Optimization Lab for Communications and Signal Processing Using MATLAB

[summer term 2021]

Prof. Dr. Anke Schmeink
ISEK Research and Teaching Group
RWTH Aachen University

Contributors

The original manuscript has been created by **F**abian Altenbach and **S**teven Corroy.

Subsequent changes have been contributed by (in alphabetical order):
Andreas Bollig (`bollig@ti.rwth-aachen.de`)
Markus Rothe (`rothe@ti.rwth-aachen.de`)
Omid Taghizadeh (`taghizadeh@ti.rwth-aachen.de`)

Preface

Signal processing and communications are widely recognized fields that are full of interesting problems. Many of these problems have been solved in the past. Others are still waiting for a smart person who brings new ideas forward. In this lab, we attempt to give you the right tools and abilities to become this person.

We believe that knowledge about (convex) optimization is the key for understanding many practical problems. However, the way optimization is treated during study is by no means practical. For example, in several classes we often encounter the Karush-Kuhn-Tucker conditions. Typically, they are used to provide analytical solutions for standard optimization problems, although most interesting problems do not even have one. Nevertheless, it seems to be some kind of tradition to expose students to those standard solutions (*e.g.*, waterfilling, entropy maximization).

That said, it may leave the impression that solving optimization problems numerically is something exceptional and should usually be avoided. In this lab, we are trying to fix that. We attempt to show that a well modeled convex optimization problem can be at least as good as any analytical solution. Fortunately, there exists a great modeling system for solving convex optimization problems, called CVX [9]. This modeling system is open source and can be used as a free toolbox for MATLAB. You will get basic knowledge in how to solve optimization problems via CVX. We will mainly concentrate on modeling problems. That's why we are also going to skip the algorithmic part of numerical optimization.

A relatively large portion of the lab is based on existing books and courses. In particular, this includes content from slides and homework problems[1]. This material is available for everyone. Therefore, we would like to gratefully thank those people, who put a lot of effort into creating this material and are still supporting the idea of open source. Here is a list of the most important references:

1. Convex Optimization [6]
2. Linear Programming [17]
3. EE263: Introduction to Linear Dynamical Systems [2],[12]
4. EE363: Linear Dynamical Systems [3]
5. Engr207b: Linear Control Systems 2 [13]
6. EE364a,b: Convex Optimization I,II [4],[5]

Note that we have tried to make this course reader as self-contained as possible. In order to pass the lab, you don't have to read this material. Of course, you are graciously invited to make your own path through the fascinating topic of optimization.

[1]To keep the presentation clear, we will omit repeated reference quoting.

References

[1] Richard Aster, Brian Borchers, and Clifford Thurber. *Parameter Estimation and Inverse Problems (International Geophysics)*. Academic Press, Jan. 2005.

[2] Stephen Boyd. *EE263: Introduction to Linear Dynamical Systems.* `http://www.stanford.edu/class/ee263/`. Stanford 2008, 2008.

[3] Stephen Boyd. *EE263: Linear Dynamical Systems.* `http://www.stanford.edu/class/ee363/`. Stanford 2008, 2008.

[4] Stephen Boyd. *EE364a: Convex Optimization I.* `http://www.stanford.edu/class/ee364a/`. Stanford 2008, 2008.

[5] Stephen Boyd. *EE364b: Convex Optimization II.* `http://www.stanford.edu/class/ee364b/`. Stanford 2008, 2008.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787.

[7] Emmanuel J. Candès, Michael B. Wakin, and Stephen P. Boyd. "Enhancing Sparsity by Reweighted l_1 Minimization". In: *The Journal of Fourier Analysis and Applications* (2008).

[8] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* 3rd. The Johns Hopkins University Press, 1996.

[9] Michael Grant, Stephen Boyd, and Yinyu Ye. "Disciplined convex programming". In: *Global Optimization: from Theory to Implementation, Nonconvex Optimization and Its Applications* (2006), pp. 155–210. URL: `http://www.stanford.edu/~boyd/disc_cvx_prog.html`.

[10] S. Joshi and S. Boyd. "Sensor selection via convex optimization". In: *IEEE Transactions on Signal Processing* 57.2 (2009), pp. 451–462.

[11] Seung-Jean Kim et al. "l_1 Trend Filtering". In: *SIAM review, problems and techniques section* (2009). `http://stanford.edu/~boyd/papers/l1_trend_filter.html`.

[12] Sanjay Lall. *EE263: Introduction to Linear Dynamical Systems.* `http://www.stanford.edu/class/ee263/`. Stanford 2009, 2009.

[13] Sanjay Lall. *Engr207b: Linear Control Systems 2.* `http://junction.stanford.edu/~lall/engr207b/`. Stanford 2009, 2009.

[14] Jia Meng et al. "Collaborative spectrum sensing from sparse observations using matrix completion for cognitive radio networks". In: *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 2010).* Dallas, TX, USA, 2010, pp. 3114–3117.

[15] Nassim N. Taleb. *Fooled by Randomness, 2nd edition.* Penguin Books, 2007.

[16] Z. Tian. "Compressed wideband sensing in cooperative cognitive radio networks". In: *IEEE Global Telecommunications Conference (GLOBECOM 2008).* New Orleans, LA, USA, Nov. 2008.

[17] Lieven Vandenberghe. *EE236A: Linear Programming.* `http://www.ee.ucla.edu/ee236a/ee236a.html`. UCLA 2008, 2008.

Contents

0 MATLAB Fundamentals

This section is meant to give you a quick and concise interactive introduction to the basics of the MATLAB computing environment. The information inside this section is a crucial prerequisite for the successful execution of the whole lab. If you already have some experience working with MATLAB you probably won't learn anything new in this session. In this case, just skim through the section quickly to make sure there is nothing you don't already know. When working through this section you should try out everything you read and experiment freely to really understand the concepts. Don't rush this. Take your time.

0.1 Directory structure and MATLAB environment

The first thing to do is to open the MATLAB environment. If you can't find it, consult your supervisor. Once the environment is opened you should make yourself familiar with it by locating the following user interface elements

- Command Window
- Workspace
- Command History
- Folder View

During this tutorial you will work with and get to know all of these UI components.

0.2 Basic variable types and data import/export

Next we want to import data from a file using the GUI as well as a MATLAB command and subsequently save data to a file. In the Folder View navigate to the folder `0_MATLAB_Fundamentals` contained in the directory structure on your desktop. Import the data contained in the file `data.mat` into the workspace by double-clicking the file. Next, explore the variables by double-clicking them in the workspace. Note, that you can manually edit variables right in this view.

To clear everything contained in the workspace, type `clear` into the command window. We now want to import the same file as above into the workspace by the means of the command window. To do so type `load data.mat` and hit enter.

To create new variables in the workspace you just assign values to them without declaring them first. As an example type

```
a = 1
b = 2
c = a + b
d = cos(a)
e = 3 + 4i
```

and observe the changes in the workspace. Note, that the last expression creates a complex variable. MATLAB's general variable type is the matrix. In MATLAB a scalar (like the ones you just created) is a 1×1 matrix, a vector is a $m \times 1$ matrix and a matrix, well a matrix in MATLAB is a $m \times n$ (you guessed it) matrix. To get a feeling for how general matrices are created in MATLAB execute the following commands and use the workspace to examine the shapes of the newly created variables.

```
f = [1 2 3 4 5]
g = 1:5
h = [3, 4, 5]
i = [3; 4; 5]
J = [1, 2; 3, 4]
k = 0:0.1:5
M = [1 2; 3 4] + [1 1; 1 1] *1i
n = 3;
```

Ending a command line with a semicolon mutes its command window output. Note that in order to display a variable's content you can also just type the variable name into the command window. To get the size of a variable type `size` followed by the variable name. The output is also a matrix.

To save the workspace to a file type `save data2.mat` and hit enter.

0.3 Functions, plotting and documentation

We have already seen a basic function call above (`d = cos(a)`). Most of MATLAB's functions are defined not only for scalar values but for vectors or matrices. As an example, type the following: `t = sin(2*pi*k)` and hit enter. Now plot the vector `t` against the index `k` by typing `plot(k, t)`. Matrices can be plotted with `surf`. Try `surf(J)` and `surf(t'*t)`.

To get help on a function type `help` or `doc` followed by the function name. Try the following:

```
help cos
doc plot
doc size
```

Using the documentation read up on the basics of the following functions: `sqrt, log, min, max, round, sum`. Play around with the functions to understand them.

You will need the documentation a lot. To browse available MATLAB functions click on the *fx* button next to the command prompt (`>>`) in the command window.

Often, MATLAB functions can be called in several different ways so it is always a good idea to check a function's documentation before coding some already present functionality oneself.

0.4 Basic operations on variables and indexing

All basic operations on scalars (`+,-,*,/`) work as expected. To get the conjugate transpose and the regular transpose of the variable `M` respectively type `M'` and `M.'`. To multiply the matrices `J` and `M`, type `J*M`. The componentwise multiplication of said matrices is computed with `J.*M`. Make sure you understand the difference between the above operations.

To get the inverse of the matrix `J` type `inv(J)`. To confirm the inverse type `inv(J) * J`. You can get the eigenvalues of `J` by typing `eig(J)` (calling the function differently will also provide you with `J`'s eigenvectors → documentation). Read up on the `diag` function in the documentation and try the following:

```
diag(J)
diag(1:5)
```

To select a matrix element or section try the following and examine the output of the commands:

```
A = rand(7)
A(2,3)
A(1:3,5:end)
```

The second line gives you the matrix entry at the second row and third column, while the third line gives you the 5th to last column of the first three lines of the matrix. To zero the first two lines of `A` type `A(1:2, :)  = 0`. To delete the second row of `A`, type `A(2, :)  = [];`. To delete multiple rows of a matrix at once (here, rows 4 and 6), try the following:

```
A = rand(7)
A([4, 6], :) = []
```

Note, that indexing in MATLAB always starts at 1 (*one*) in contrast to other programming languages most of which are zero-indexed.

0.5 Matrix generation

MATLAB possesses some functions to generate matrices of certain useful types. To get an identity matrix of size $N \times N$ type `eye(N)`. To get a matrix filled with zeros / ones of the same size respectively type `zeros(N)` or `ones(N)`. If the zeros / ones filled matrix is supposed to be non-square use `zeros(m,n)` or `ones(m,n)`.

We can also generate random matrices with MATLAB. Try out the following examples:

```
A = rand(5)
B = rand(3,2)
C = randn(5)
D = randn(3,2)
```

Read up on the difference between `rand`, `randi` and `randn` in the documentation.

0.6 MATLAB scripts, loops, conditional statements and logical operators

To open a new MATLAB script click the *New Script* button in the top left corner. This will open the editor with an empty script. You can run the script from the editor by either hitting `F5` on the keyboard or clicking on the green *Run* button. By putting the cursor on a function call and hitting `F1` in the editor you can open the documentation of the selected function.

To understand the basics about loops and conditional statements in MATLAB open the script contained in the file `example.m` in the section's folder with the editor. We can learn a couple of things from this script. The first line is a good line for initializing the script by resetting the whole MATLAB environment. The command `clear` empties the workspace while `close all` closes all open plotting windows and `clc` clears the command window.

In lines 3 to 7 we see a simple example of a for-loop. In MATLAB a for-loop loops over a pre-specified vector. You could also specify the vector inline as shown below:

```matlab
for i = 1:5
  disp(i);
end
```

Lines 9 to 16 show an example of a while-loop. The loop will repeat its content as long as the condition given after the `while` command is true, i.e., does not equal zero. The conditional statement in lines 18 to 26 should be self-explanatory. In lines 28 to 39 we see that `&&` stands for the scalar logical AND, `||` denotes the scalar logical OR and the tilde is the logical NOT-operation.

0.7 Logical indexing

Logical indexing can come in very handy when using MATLAB. Consider the following example:

```matlab
A = randi(2,3,4) -1
A(A ~= 0) = 3
A(A == 0) = 2
```

0.8 MATLAB test

If you feel like you understand and can execute all of the above ask a supervisor for the multiple choice MATLAB test.

1 Preliminaries I: Linear algebra and vector analysis

In this session, we will have a short exposure on a couple of basic mathematical concepts. We start with a short introduction to linear algebra. After that, some basic rules for first- and second order derivatives are given. These basic concepts will be needed in later sessions of the lab. In fact, they are also mandatory in order to understand convex optimization.

1.1 Motivation for linear algebra

1.1.1 Why do we have to study linear algebra?

Linear algebra is a field in mathematics which comprises the study of *vector spaces*, also called *linear spaces*. This includes linear functions f that map an input vector $\mathbf{x} \in \mathbb{R}^n$ to an output vector $\mathbf{y} \in \mathbb{R}^m$, that is,

$$\mathbf{y} = f(\mathbf{x}). \tag{1.1}$$

Given a basis, for example the standard basis in $\mathbb{R}^n$, we can uniquely represent this mapping by a simple matrix-vector multiplication

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \qquad \mathbf{A} \in \mathbb{R}^{m \times n}. \tag{1.2}$$

Here comes the cool part: The most important class of mathematical models in engineering are linear systems. That means, we can represent a huge class of systems by this matrix-vector multiplication. In the case of nonlinear systems, it is common to linearize them near a specific operating point, for example the linearization of transistor characteristics. Therefore, concepts from linear algebra can also be applied to nonlinear systems. As we will see later, this is not only a conceptual thing. Using the right theorems from linear algebra enables us to get insights into many real-world problems!

We will now give the sketch of a short proof of this important idea. A linear system obeys the law of superposition

$$f(\mathbf{x}_1 + \mathbf{x}_2) = f(\mathbf{x}_1) + f(\mathbf{x}_2), \tag{1.3}$$

and scaling

$$f(\alpha\mathbf{x}) = \alpha f(\mathbf{x}). \tag{1.4}$$

Expressing the vector $\mathbf{x}$ in terms of the standard basis gives $\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{e}_i$. The basis vectors are given by $\mathbf{e}_i$, where all entries are zero, except the ith entry which equals one. Assuming a linear system f, it follows that

$$\begin{aligned}
\mathbf{y} = f(\mathbf{x}) &= f(\sum_{i=1}^{n} x_i \mathbf{e}_i) && (1.5)\\
&= f(x_1\mathbf{e}_1) + \cdots + f(x_n\mathbf{e}_n) \quad \text{(superposition)} && (1.6)\\
&= x_1 f(\mathbf{e}_1) + \cdots + x_n f(\mathbf{e}_n) \quad \text{(scaling)}. && (1.7)
\end{aligned}$$

Using $f(\mathbf{e}_i) \in \mathbb{R}^m$, we can rewrite this as

$$\mathbf{y} = \underbrace{\begin{bmatrix} f(\mathbf{e}_1) & \cdots & f(\mathbf{e}_n) \end{bmatrix}}_{\mathbf{A} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}_{\mathbf{x} \in \mathbb{R}^n} = \mathbf{A}\mathbf{x}. \tag{1.8}$$

1.1.2 Example: Discrete-time convolution

Convolutions play a central role in many fields, *e.g.*, audio signal processing, image processing, communication, and control. Whenever we are facing the convolution between two discrete-time signals $x(k)$ and $h(k)$, we should immediately think of a matrix-vector multiplication. For simplicity, we will assume that both signals are nonzero only inside the interval $0 \leq k \leq N-1$. The discrete-time convolution is then defined as

$$y(k) = \sum_{\tau=0}^{N-1} x(k-\tau)h(\tau), \qquad k = 0, 1, \ldots \tag{1.9}$$

Rewriting this as a matrix-vector multiplication gives us

$$\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \\ y(N) \\ \vdots \\ y(2N-2) \end{bmatrix} = \begin{bmatrix} h(0) & 0 & \cdots & 0 \\ h(1) & h(0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h(N-1) & h(N-2) & \cdots & h(0) \\ 0 & h(N-1) & \cdots & h(1) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h(N-1) \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix}, \tag{1.10}$$

or equivalently $\mathbf{y} = \mathbf{H}\mathbf{x}$. On a side note, the restriction to the interval $0 \leq k \leq N-1$ can easily be extended to other finite dimensional cases by simply modifying the size of the corresponding matrix and vectors. The infinite dimensional case is not covered by this simple mapping. Therefore, it is also not covered by linear algebra. It is subject of a field called functional analysis.

1.1.3 Interpretations of $\mathbf{y} = \mathbf{A}\mathbf{x}$

So far we have not said anything about the meaning of the linear model $\mathbf{y} = \mathbf{A}\mathbf{x}$, particularly with respect to applications. We can distinguish between three categories:

1. estimation or inversion
2. control or design
3. mapping or transformation.

During this session and the lab in general, we will frequently link concepts from linear algebra to estimation and control problems.

Estimation or inversion

Given $\mathbf{y} = \mathbf{A}\mathbf{x}$, we have:

- y_i is the ith measurement or sensor reading,
- x_j is the jth parameter to be estimated,
- a_{ij} is the sensitivity of the ith sensor to the jth parameter.

Typical problems are:

- Find all vectors $\mathbf{x}$ that result in $\mathbf{y}$, *i.e.*, all vectors that are consistent with the measurements.
- If there is no $\mathbf{x}$ such that $\mathbf{y} = \mathbf{A}\mathbf{x}$, find a vector $\mathbf{x}$ that is almost consistent with the measurements, that is: $\mathbf{y} \approx \mathbf{A}\mathbf{x}$.

Control or design

Given $\mathbf{y} = \mathbf{A}\mathbf{x}$, we have:

- x_j is the jth control input or design parameter (which we can choose),
- y_i is the ith output we want to get (or want to achieve),
- a_{ij} describes the effectiveness of the jth control input to the ith output.

Typical problems are:

- Find all control input vectors $\mathbf{x}$ that result in some desired output $\mathbf{y} = \mathbf{y}_{\text{des}}$, *i.e.*, all vectors that meet certain design specifications.
- Among all vectors $\mathbf{x}$ that satisfy $\mathbf{y} = \mathbf{y}_{\text{des}}$, find a "small" one (this could correspond to a low power consumption).

1.2 Basic building blocks of linear algebra

The above section gave merely a motivation why it might be important to have some knowledge in linear algebra. This does not only play a role in this lab. In fact, when it comes to engineering, it is also important to know about this material in general. In the following subsections, we will introduce some of the most important basic building blocks of linear algebra. Although some of these blocks may look a bit technical, they all have interpretations and applications in many practical problems.

1.2.1 Linear spaces and subspaces

As stated in the first subsection, linear algebra focuses on the study of vector spaces. A vector space or linear space over the real numbers consists of a set $\mathcal{V}$, a vector sum $+ : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$, a scalar multiplication[2] $\cdot : \mathbb{R} \times \mathcal{V} \to \mathcal{V}$, and a distinguished element $\mathbf{0} \in \mathcal{V}$. These elements must fulfill the following list of properties:

1. $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$, $\forall \mathbf{x}, \mathbf{y} \in \mathcal{V}$, $+$ is commutative
2. $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$, $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{V}$, $+$ is associative
3. $\mathbf{0} + \mathbf{x} = \mathbf{x}$, $\forall \mathbf{x} \in \mathcal{V}$, $\mathbf{0}$ is the additive identity
4. $\forall \mathbf{x} \in \mathcal{V}$, $\exists (-\mathbf{x}) \in \mathcal{V}$, such that $\mathbf{x} + (-\mathbf{x}) = \mathbf{0}$, existence of the additive inverse
5. $1\mathbf{x} = \mathbf{x}$, $\forall \mathbf{x} \in \mathcal{V}$, neutral element of the scalar multiplication
6. $(\alpha\beta)\mathbf{x} = \alpha(\beta\mathbf{x})$, $\forall \alpha, \beta \in \mathbb{R}$, $\forall \mathbf{x} \in \mathcal{V}$, scalar multiplication is associative
7. $\alpha(\mathbf{x} + \mathbf{y}) = \alpha\mathbf{x} + \alpha\mathbf{y}$, $\forall \alpha \in \mathbb{R}$, $\forall \mathbf{x}, \mathbf{y} \in \mathcal{V}$, right distributive rule
8. $(\alpha + \beta)\mathbf{x} = \alpha\mathbf{x} + \beta\mathbf{x}$, $\forall \alpha, \beta \in \mathbb{R}$, $\forall \mathbf{x} \in \mathcal{V}$, left distributive rule

In what follows, we will not focus on these rules. If you would like to check whether or not a given space is a vector space, you would have to check if the above rules are fulfilled. Sometimes this has very limited practical use. For example, the way computers calculate vector additions by no means obeys the associative law. This is caused by the fact that computers work within finite precision. Nevertheless, in most cases the precision is high enough and we can think of something like approximate vector spaces (do not mention this outside the lab!).

Two basic examples for vector spaces are $\mathcal{V}_1 = \mathbb{R}^n$ together with standard component-wise vector addition and multiplication, and the set $\mathcal{V}_2 = \{\mathbf{0}\}$ (this is not the empty set!). Another one is the so-called *span* of a set of vectors which is given by

$$\mathcal{V}_3 = \operatorname{span}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k) = \{\alpha_1\mathbf{v}_1 + \cdots + \alpha_k\mathbf{v}_k | \alpha_i \in \mathbb{R}\}. \tag{1.11}$$

To put it simply, the span is the set of all linear combinations that can be constructed from $\mathbf{v}_i|_{i=1}^{k}$. We will need the concept of the span when defining a basis for a vector space (see Sec. 1.2.2).

Let $\mathcal{V}$ be a vector space. A subset $\mathcal{W} \subseteq \mathcal{V}$ is called (vector) *subspace* if and only if it satisfies the following three conditions:

1. For all $\mathbf{x}, \mathbf{y} \in \mathcal{W}$, the linear combination $\mathbf{x} + \mathbf{y}$ is also in $\mathcal{W}$.
2. For all $\mathbf{x} \in \mathcal{W}$ and $\alpha \in \mathbb{R}$, the scalar multiplication $\alpha\mathbf{x}$ is also in $\mathcal{W}$.
3. The distinguished element $\mathbf{0}$ is also in $\mathcal{W}$.

Roughly speaking, a subspace is closed under vector addition and scalar multiplication. The above vector spaces $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$ are all subspaces of $\mathbb{R}^n$.

[2]From now on, we will omit the additional notation for scalar multiplication.

1.2.2 Independence, basis and dimension

A set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ is *independent* if

$$\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_k\mathbf{v}_k = \mathbf{0} \Rightarrow \alpha_1 = \alpha_2 = \cdots = \alpha_k = 0 \tag{1.12}$$

An equivalent and well-known statement of this condition is that no vector $\mathbf{v}_i$ can be expressed as a linear combination of the other vectors $\mathbf{v}_1, \dots, \mathbf{v}_{i-1}, \mathbf{v}_{i+1}, \dots, \mathbf{v}_k$. It is important to understand that independence is an attribute of a set of vectors, not of a single member of this set. Sometimes people are using mathematical slang. They might speak of independent vectors, but what they really mean is an independent set of vectors.

A set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ is a *basis* for a vector space $\mathcal{V}$ if

1. $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ span the vector space $\mathcal{V}$, *i.e.*, $\mathcal{V} = \text{span}(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k)$.
2. The set $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ is independent.

Why is the concept of a basis important? Because we can uniquely express any vector $\mathbf{v} \in \mathcal{V}$ as a linear combination of the basis vectors

$$\mathbf{v} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_k\mathbf{v}_k. \tag{1.13}$$

For example, consider the vector space $\mathbb{R}^3$. The so-called standard basis for $\mathbb{R}^3$ consists of the three vectors

$$\mathbf{v}_1 = \begin{bmatrix}1\\0\\0\end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix}0\\1\\0\end{bmatrix}, \quad \mathbf{v}_3 = \begin{bmatrix}0\\0\\1\end{bmatrix}. \tag{1.14}$$

Any point in this space can be expressed by an appropriate scaling. For example $\mathbf{v} = \begin{bmatrix}3 & -2 & \sqrt{\pi}\end{bmatrix}^{\mathrm{T}} = 3\mathbf{v}_1 - 2\mathbf{v}_2 + \sqrt{\pi}\mathbf{v}_3$.

This leads us to the next question. How many bases does a vector space $\mathcal{V}$ have? In principle, there might be an infinite number of different bases (just change the basis vectors and the scaling). However, each basis has the same number of vectors. This is called the *dimension* of $\mathcal{V}$. For example, $\dim(\mathbb{R}^3) = 3$. Every set of n independent vectors in $\mathbb{R}^n$ is also a basis for $\mathbb{R}^n$.

1.2.3 Nullspace and range

The *nullspace* of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is defined as

$$\mathcal{N}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{A}\mathbf{x} = \mathbf{0}\}. \tag{1.15}$$

It is the set of vectors $\mathbf{x}$ that are mapped to zero by $\mathbf{y} = \mathbf{A}\mathbf{x}$. Think about the consequences of this matrix property. For instance, take a vector $\mathbf{z}$ which is element of the nullspace of $\mathbf{A}$, that is $\mathbf{z} \in \mathcal{N}(\mathbf{A})$. The equation

$$\mathbf{y} = \mathbf{A}(\mathbf{x} + \mathbf{z}) = \mathbf{A}\mathbf{x} + \underbrace{\mathbf{A}\mathbf{z}}_{=\mathbf{0}} = \mathbf{A}\mathbf{x} \tag{1.16}$$

seems to have ambiguous solutions. The "size" of the nullspace gives us something like an ambiguity measure. Be careful, the nullspace is never empty. It always contains the distinguished element $\mathbf{0}$. It is therefore fine to speak of a zero nullspace.

The *range* of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ is defined as

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} \,|\, \mathbf{x} \in \mathbb{R}^n\} \subseteq \mathbb{R}^m. \tag{1.17}$$

It is the set of vectors that can be "hit" by the linear mapping $\mathbf{y} = \mathbf{A}\mathbf{x}$. This is also the set of vectors $\mathbf{y}$ for which $\mathbf{y} = \mathbf{A}\mathbf{x}$ has a solution. Although more complicated in their structure, it is important to know that nullspace and range are also subspaces of $\mathbb{R}^m$.

Interpretations of the nullspace

Suppose we have $\mathbf{z} \in \mathcal{N}(\mathbf{A})$. In an estimation or inversion problem, $\mathbf{y} = \mathbf{A}\mathbf{x}$ represents the measurement of $\mathbf{x}$.

- $\mathbf{z}$ is undetectable by the sensors
- $\mathbf{x}$ and $\mathbf{x} + \mathbf{z}$ are indistinguishable for the sensors, because $\mathbf{A}\mathbf{x} = \mathbf{A}(\mathbf{x} + \mathbf{z})$

The nullspace $\mathcal{N}(\mathbf{A})$ characterizes the ambiguity in $\mathbf{x}$ from measurement $\mathbf{y} = \mathbf{A}\mathbf{x}$.

In the case of a control or design problem, $\mathbf{y} = \mathbf{A}\mathbf{x}$ represents the output resulting from the control input $\mathbf{x}$.

- $\mathbf{z}$ is a control input with no result or effect
- $\mathbf{x}$ and $\mathbf{x} + \mathbf{z}$ have exactly the same result or effect

The nullspace $\mathcal{N}(\mathbf{A})$ characterizes freedom of input choice for a given result. Think about this: A big nullspace in an estimation problem is bad, because we have more than one solution. A big nullspace in a control problem can be great, because we have lots of free choices for our control input.

Interpretations of the range

Suppose we have $\mathbf{v} \in \mathcal{R}(\mathbf{A})$, $\mathbf{w} \notin \mathcal{R}(\mathbf{A})$. In an estimation or inversion problem, $\mathbf{y} = \mathbf{A}\mathbf{x}$ represents the measurement of $\mathbf{x}$.

- $\mathbf{y} = \mathbf{v}$ is a possible or consistent sensor signal
- $\mathbf{y} = \mathbf{w}$ is impossible or inconsistent. It may also have happened that sensors have failed or our proposed model is wrong.

In the case of a control or design problem, $\mathbf{y} = \mathbf{A}\mathbf{x}$ represents the output resulting from control input $\mathbf{x}$.

- $\mathbf{y} = \mathbf{v}$ is a possible control output or result
- $\mathbf{y} = \mathbf{w}$ is an impossible control output or result

The range of $\mathbf{A}$ characterizes the possible results (from measurements) or achievable outputs (from a control procedure).

1.2.4 Rank and conservation of dimension

The *rank* of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is defined as

$$\operatorname{rank}(\mathbf{A}) = \dim(\mathcal{R}(\mathbf{A})). \tag{1.18}$$

This means the rank corresponds to the size or dimension of the range space. There are some important but non-trivial statements about the rank:

1. $\operatorname{rank}(\mathbf{A}) = \operatorname{rank}(\mathbf{A}^{\mathrm{T}})$
2. $\operatorname{rank}(\mathbf{A}) \leq \min(m, n)$
3. $\operatorname{rank}(\mathbf{A}) + \dim(\mathcal{N}(\mathbf{A})) = n$

First of all, the rank of a matrix does not depend on the orientation of the matrix. The second line states that the rank is upper bounded by the minimum number of columns/rows. The last statement is what we mean by *conservation of dimension.* Roughly speaking:

- n is the number of degrees of freedom in the input $\mathbf{x}$
- $\operatorname{rank}(\mathbf{A}) = \dim(\mathcal{R}(\mathbf{A}))$ is the dimension of the set of things you can hit via the mapping from $\mathbf{x}$ to $\mathbf{y} = \mathbf{A}\mathbf{x}$
- $\dim(\mathcal{N}(\mathbf{A}))$ is the number of degrees of freedom lost in the mapping from $\mathbf{x}$ to $\mathbf{y} = \mathbf{A}\mathbf{x}$
- $\operatorname{rank}(\mathbf{A})$ is the number of degrees of freedom in the output $\mathbf{y}$

The rank of a matrix has a tremendous impact on practical problems, as well as it has beautiful interpretations for those.

Let us discuss a short example of this concept. Suppose we have to make a big survey, for example about personal acceptance of wireless communication devices. We want to estimate the variables or driving factors $\mathbf{x}$ behind personal acceptance, *e.g.*, age, gender, education, political viewpoint etc. We have to set up a big measurement matrix, say $\mathbf{A} \in \mathbb{R}^{300 \times 150}$. Each column of this matrix represents a different variable and each row stands for a different measurement (or opinion). We now start our survey, spending a lot of money in order to get measurements from at least 300 different probands. What does it mean if the rank of the resulting measurement matrix turns out to be 19? Well, first of all this means our fancy 150-dimensional model can be explained in a much simpler 19-dimensional subspace or even less[3]. Secondly, in many cases people are putting a lot of effort in constructing matrices like these. Not to mention the effort for those 300 probands. They have to answer 150 questions instead of 19. Always be careful when you try to set up a complicated model, it could waste a lot of resources!

[3]As we will see later, for practical applications it sometimes makes more sense to speak about the effective rank of a matrix.

1.2.5 Invertible matrices

A square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is said to be *invertible* or *nonsingular* if $\det(\mathbf{A}) \neq 0$. There are a number of equivalent conditions on this statement, which we mention without proof:

1. The columns (rows) of $\mathbf{A}$ are a basis for $\mathbb{R}^n$.
2. $\mathbf{y} = \mathbf{A}\mathbf{x}$ has a unique solution $\mathbf{x}$ for every $\mathbf{y} \in \mathbb{R}^n$.
3. $\mathbf{A}$ has a (left and right) inverse denoted by $\mathbf{A}^{-1} \in \mathbb{R}^{n \times n}$, with $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.
4. $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$ and $\mathcal{R}(\mathbf{A}) = \mathbb{R}^n$. We say that $\mathbf{A}$ has *full rank*.
5. $\det(\mathbf{A}^{\mathrm{T}}\mathbf{A}) = \det(\mathbf{A}\mathbf{A}^{\mathrm{T}}) \neq 0$

For example, suppose we are given the output of a communication channel $\mathbf{y} = \mathbf{H}\mathbf{x}$, where $\mathbf{H}$ is a known or estimated matrix of channel gains. We would like to find the input signal $\mathbf{x}$. If $\mathbf{H}$ is invertible (square and full rank), then we can find an optimal equalizer $\mathbf{H}_{\mathrm{eq}} = \mathbf{H}^{-1}$ in order to recover the input perfectly, that is $\mathbf{x} = \mathbf{H}_{\mathrm{eq}}\mathbf{y} = \mathbf{H}^{-1}\mathbf{H}\mathbf{x}$.

Provided we have a reliable method for inverting matrices, it is now tempting to finish the section on linear algebra. Can we simply apply matrix inverses in order to solve $\mathbf{y} = \mathbf{A}\mathbf{x}$? Obviously, the answer is no. When it comes to practical problems, at least four non-exclusive issues occur:

1. It is often the case that a matrix has not full rank.
2. Even if a matrix has full rank, what does $\mathbf{A}\mathbf{x} \approx \mathbf{0}$ mean?
3. In many practical applications, matrices are not square.
4. In most cases we have to combat additional noise, which is not covered by $\mathbf{y} = \mathbf{A}\mathbf{x}$.

In the following subsections, we will discuss how to handle these issues.

1.2.6 Eigenvalue decomposition of symmetric matrices

We say that $\lambda \in \mathbb{C}$ is an *eigenvalue* of the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ if it solves the *characteristic equation*

$$\det(\lambda \mathbf{I} - \mathbf{A}) = 0. \tag{1.19}$$

This means that the matrix $(\lambda \mathbf{I} - \mathbf{A})$ has a nonzero nullspace[4]. Let $\mathbf{v} \in \mathbb{C}^n$ be a nonzero element of this nullspace. It follows that

$$(\lambda \mathbf{I} - \mathbf{A})\mathbf{v} = \mathbf{0} \quad \Leftrightarrow \quad \mathbf{A}\mathbf{v} = \lambda \mathbf{v}, \quad \mathbf{v} \neq \mathbf{0}. \tag{1.20}$$

The element $\mathbf{v}$ is called (right) *eigenvector* of $\mathbf{A}$ associated with the eigenvalue λ. Taking an eigenvector $\mathbf{v}$ as the input direction for a linear system, we see that the output is just a scaled version of this direction.

[4]Otherwise $\det(\lambda \mathbf{I} - \mathbf{A}) \neq 0$.

The characteristic equation is a polynomial of degree n. Therefore, a matrix of dimension $n \times n$ has at most n distinct eigenvalues. Watch out, although $\mathbf{A}$ is real, it can have complex eigenvectors and -values. Now suppose $\mathbf{A} \in \mathbb{R}^{n\times n}$ is symmetric, *i.e.*, $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$. Then its eigenvalues and -vectors are real. To see this, we evaluate the quadratic form

$$\mathbf{v}^{\mathrm{H}}\mathbf{A}\mathbf{v} = \mathbf{v}^{\mathrm{H}}(\mathbf{A}\mathbf{v}) = \lambda \mathbf{v}^{\mathrm{H}}\mathbf{v} = \lambda \sum_{i=1}^{n} |v_i|^2, \tag{1.21}$$

but also

$$\mathbf{v}^{\mathrm{H}}\mathbf{A}\mathbf{v} = (\mathbf{A}\mathbf{v})^{\mathrm{H}}\mathbf{v} = (\lambda \mathbf{v})^{\mathrm{H}}\mathbf{v} = \lambda^* \sum_{i=1}^{n} |v_i|^2. \tag{1.22}$$

This means we have $\lambda = \lambda^*$. A complex number equals its conjugate if the imaginary part is zero. Therefore $\lambda \in \mathbb{R}$. The proof for the eigenvectors is similar.

Real eigenvalues are not the only important property of symmetric matrices. It can be shown, that any real symmetric matrix $\mathbf{A}$ can be factored as

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}}. \tag{1.23}$$

The above factorization is called *eigenvalue decomposition* (EVD) of $\mathbf{A}$, where $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is *orthogonal.* An orthogonal matrix requires $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I} \Leftrightarrow \mathbf{Q}^{\mathrm{T}} = \mathbf{Q}^{-1}$. We could also say that the rows of $\mathbf{Q}^{\mathrm{T}}$ and the columns of $\mathbf{Q}$ form an orthonormal[5] set of vectors. Like independence and basis, orthonormality for vectors is a property of a set, not of a single element.

Positive definite matrices and their invertibility

A real symmetric matrix $\mathbf{A}$ is called *positive definite*, written $\mathbf{A} > 0$, if

$$\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0 \quad \text{for all nonzero } \mathbf{x} \in \mathbb{R}^n, \tag{1.24}$$

that is, its quadratic form is strictly positive. A matrix is positive definite if and only if it has positive eigenvalues only. To see this, take the nth eigenvector, which corresponds to the smallest eigenvalue $\lambda_n = \lambda_{\min}(\mathbf{A})$. Plugging this into the quadratic form gives

$$0 < \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{v}_n^{\mathrm{T}}\mathbf{A}\mathbf{v}_n = \lambda_n \mathbf{v}_n^{\mathrm{T}}\mathbf{v}_n = \lambda_n \tag{1.25}$$

Since all eigenvalues are real, we have $\lambda_n \leq \cdots \leq \lambda_1$, that is, all eigenvalues are positive. This gives us a test for positive definiteness of a matrix. Whenever the smallest eigenvalue is non-positive, we can directly conclude that it can't be positive definite.

We will now show that any real symmetric positive definite matrix is invertible. Assume we have $\mathbf{A} = \mathbf{A}^{\mathrm{T}} > 0$. Evaluating the determinant gives

$$\begin{aligned}
\det(\mathbf{A}) &= \det(\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}}) && \text{(eigenvalue decomposition of } \mathbf{A}\text{)}\\
&= \det(\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}}\mathbf{Q}) && \text{(cyclic permutation within } \det(\cdot)\text{)}\\
&= \det(\mathbf{\Lambda}) && \text{(}\mathbf{Q}\text{ is orthogonal).}
\end{aligned}$$

[5]Unfortunately, orthogonality for matrices corresponds to orthonormality for vectors.

The determinant of a diagonal matrix is simply the product of its diagonal elements. We then have

$$\det(\mathbf{A}) = \det(\mathbf{\Lambda}) = \prod_{i=1}^{n} \lambda_i(\mathbf{A}) \neq 0 \tag{1.26}$$

The product term can't be zero, because all eigenvalues are positive. Therefore, $\det(\mathbf{A}) \neq 0$, and this implies $\mathbf{A}$ is invertible.

A real symmetric matrix $\mathbf{A}$ is called *positive semidefinite*, written $\mathbf{A} \geq 0$, if

$$\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} \geq 0 \quad \text{for all nonzero } \mathbf{x} \in \mathbb{R}^n, \tag{1.27}$$

that is, its quadratic form is nonnegative. This means, there might be some eigenvalues that are zero. When this is the case, the determinant becomes zero and the matrix is singular and hence is not invertible. Be careful with this observation, especially in the case of (almost) positive semidefinite covariance matrices!

Gain of a matrix in a direction

At the end of subsection 1.2.5, we have raised several concerns about the meaning of $\mathbf{y} = \mathbf{A}\mathbf{x}$. We will now shed some light on this issue. Consider the squared Euclidean norm or squared ℓ_2-norm, which is given by

$$\|\mathbf{x}\|_2^2 = \mathbf{x}^{\mathrm{T}}\mathbf{x} = \sum_{i=1}^{n} |x_i|^2. \tag{1.28}$$

How large is the output $\mathbf{y} = \mathbf{A}\mathbf{x}$ of a linear system compared to the input $\mathbf{x}$? In terms of the ℓ_2-norm, we could use the quantity $\|\mathbf{y}\|_2^2 = \|\mathbf{A}\mathbf{x}\|_2^2$. However, any scaling of $\mathbf{x}$ would lead to a higher output in $\mathbf{y}$. What we really want to find out is the influence of the matrix $\mathbf{A}$. Therefore, we normalize this relation by the ℓ_2-norm of the input

$$\frac{\|\mathbf{y}\|_2^2}{\|\mathbf{x}\|_2^2} = \frac{\|\mathbf{A}\mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2}, \quad \mathbf{x} \neq \mathbf{0}. \tag{1.29}$$

The quantity $\|\mathbf{A}\mathbf{x}\|_2^2/\|\mathbf{x}\|_2^2$ gives the amplification factor or *gain* of the matrix $\mathbf{A}$ in the direction $\mathbf{x}$. We can immediately raise several questions about the gain, for example

1. What is the maximum gain and the maximum gain direction of $\mathbf{A}$?

2. What is the minimum gain and the minimum gain direction of $\mathbf{A}$?

3. How does the gain of $\mathbf{A}$ vary with direction?

We now also have a nice way to say what an element in the nullspace of a matrix actually is. It is a direction in which the matrix gain is zero. Now suppose a matrix has a zero nullspace and therefore has full rank. In spite of this mathematical fact, there might still be directions with a very small gain, which is why it is sometimes useful to adopt the notion of an effective or numerical matrix rank.

Matrix norm

The maximum gain of a matrix is called *matrix norm* or *spectral norm* and is given by

$$\|\mathbf{A}\|_2 = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2}. \tag{1.30}$$

The above expression is already some kind of optimization problem. Note that we don't require the matrix $\mathbf{A}$ to be square. Using the so-called Rayleigh-Ritz theorem, we can rewrite this as

$$\max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2} = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x}}{\mathbf{x}^{\mathrm{T}}\mathbf{x}} = \lambda_{\max}(\mathbf{A}^{\mathrm{T}}\mathbf{A}). \tag{1.31}$$

So we have

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^{\mathrm{T}}\mathbf{A})} \tag{1.32}$$

Note that $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is symmetric and positive semidefinite. Therefore, all eigenvalues are nonnegative. The above expression is a nice generalization of the scalar case, where $|a| = \sqrt{a^2}$ for scalars a. Similarly, the minimum gain is given by

$$\min_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sqrt{\lambda_{\min}(\mathbf{A}^{\mathrm{T}}\mathbf{A})}. \tag{1.33}$$

To summarize, the maximum/minimum gain direction of a matrix can be stated as

1. The maximum gain input direction is $\mathbf{v}_1$, which is the eigenvector of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ associated with $\lambda_{\max}$.
2. The minimum gain input direction is $\mathbf{v}_n$, which is the eigenvector of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ associated with $\lambda_{\min}$.

In problem 1.3, we will discuss a nice application of the matrix norm.

Matrix square roots

As in the scalar case, there exists the notion of a square root in the matrix sense. Let $\mathbf{A}$ be a symmetric positive semidefinite matrix. Its eigenvalue decomposition is given by

$$\begin{aligned}
\mathbf{A} &= \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}} && (1.34)\\
&= \mathbf{Q}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{Q}^{\mathrm{T}}\mathbf{Q}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{Q}^{\mathrm{T}} && (1.35)\\
&= (\mathbf{Q}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{Q}^{\mathrm{T}})^2, && (1.36)
\end{aligned}$$

where $\mathbf{\Lambda}^{\frac{1}{2}} = \mathrm{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots, \sqrt{\lambda_n})$. We call the matrix $\mathbf{A}^{\frac{1}{2}} = \mathbf{Q}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{Q}^{\mathrm{T}}$ the (symmetric) *matrix square root* of $\mathbf{A}$. As in the scalar case, we have $\mathbf{A} = \mathbf{A}^{\frac{1}{2}}\mathbf{A}^{\frac{1}{2}}$. Note that the matrix square root requires that $\mathbf{A}$ is positive semidefinite.

1.2.7 Singular value decomposition

Using the matrix norm, we can quantify the most sensitive/insensitive direction of a matrix. We will now see that the *singular value decomposition* (SVD) gives us a way to pick out the essential features of any linear mapping $\mathbf{y} = \mathbf{A}\mathbf{x}$, and simplify it by removing the unessential ones. The singular value decomposition of a matrix $\mathbf{A}$ is given by

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}}, \tag{1.37}$$

where

- $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathrm{rank}(\mathbf{A}) = r \leq \min(m, n)$
- $\mathbf{U} \in \mathbb{R}^{m\times r}$, $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$
- $\mathbf{V} \in \mathbb{R}^{n\times r}$, $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$
- $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$, where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$.

The vectors $\mathbf{v}_i$ are the right or input singular vectors, the vectors $\mathbf{u}_i$ are the left or output singular vectors of $\mathbf{A}$. The diagonal elements σ_i are called singular values. Any matrix $\mathbf{A}$ can be written in this form, *i.e.*, the singular value decomposition always exists.

Relation to the eigenvalue decomposition

There is a direct link between SVD and EVD. Consider

$$\begin{aligned}
\mathbf{A}^{\mathrm{T}}\mathbf{A} &= (\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}})^{\mathrm{T}}(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}) && && (1.38)\\
&= \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^{\mathrm{T}} && (\mathbf{U} \text{ is orthogonal}) && (1.39)\\
&= \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathrm{T}}, && (\text{equals EVD of } \mathbf{A}^{\mathrm{T}}\mathbf{A}) && (1.40)
\end{aligned}$$

so we have $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^{\mathrm{T}}\mathbf{A})}$ and $\lambda_i(\mathbf{A}^{\mathrm{T}}\mathbf{A}) = 0$ for $i > r$. Similarly,

$$\begin{aligned}
\mathbf{A}\mathbf{A}^{\mathrm{T}} &= (\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}})(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}})^{\mathrm{T}} && (1.41)\\
&= \mathbf{U}\boldsymbol{\Sigma}^2\mathbf{U}^{\mathrm{T}} && (1.42)\\
&= \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathrm{T}}, && (1.43)
\end{aligned}$$

where $\sigma_i = \sqrt{\lambda_i(\mathbf{A}\mathbf{A}^{\mathrm{T}})}$ and $\lambda_i(\mathbf{A}\mathbf{A}^{\mathrm{T}}) = 0$ for $i > r$. The singular values of $\mathbf{A}$ are the square roots of the nonzero eigenvalues of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ or $\mathbf{A}\mathbf{A}^{\mathrm{T}}$. In particular, it follows

$$\|\mathbf{A}\|_2 = \sigma_1, \tag{1.44}$$

i.e., the matrix norm of a matrix equals the maximum singular value of that matrix.

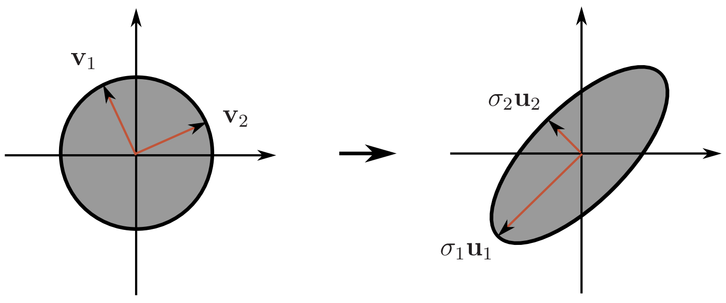


Figure 1.1: Geometry of linear maps.

Geometrical interpretation

We will start with an important fact from linear algebra. Suppose we have a vector $\mathbf{x} \in \mathbb{R}^n$ in the unit ball set, that is

$$S = \{\mathbf{x} \in \mathbb{R}^n \,|\, \|\mathbf{x}\|_2 \leq 1\}. \tag{1.45}$$

This is a fairly general representation, since we could place any finite ℓ_2-norm vector in this unit ball set by appropriate scaling. Here is the important part: Every matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ maps the unit ball in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$ (Fig. 1.1)

$$AS = \{\mathbf{Ax} \,|\, \mathbf{x} \in S\}. \tag{1.46}$$

Because we have a linear mapping $\mathbf{y} = \mathbf{Ax}$, we now also know what a linear system does. The question is, what are the quantitative characteristics of such a mapping? The answer lies in the singular value decomposition

$$\mathbf{A} = \mathbf{U\Sigma V}^{\mathrm{T}} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}}. \tag{1.47}$$

This operation can be summarized as (Fig. 1.2):

1. rotate by $\mathbf{V}^{\mathrm{T}}$
2. stretch along the axes by σ_i ($\sigma_i = 0$ for $i > r$)
3. zero-pad (if $m > n$) or truncate (if $m < n$) to get an m-vector
4. rotate by $\mathbf{U}$

In particular, observe that the scaling of the input/output directions is given by the singular values. Small values σ_i will shrink the input direction $\mathbf{v}_i$ corresponding to the output direction $\mathbf{u}_i$, while large ones result in an amplification.

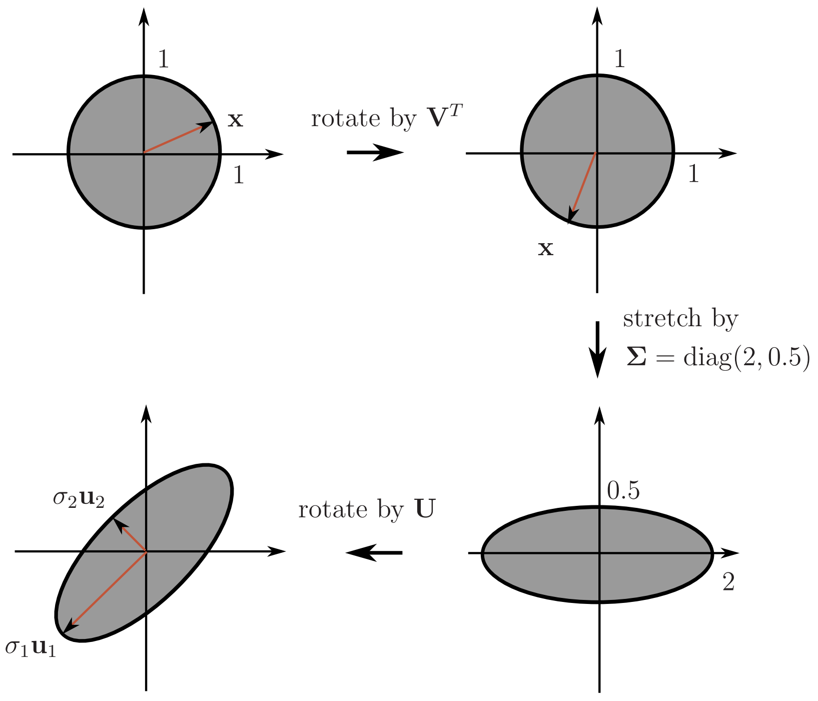


Figure 1.2: SVD example.

Interpretation for estimation and control

The SVD gives us a clearer picture of the matrix gain as a function of input/output directions. Consider for example the SVD of a 4×4 matrix, where the singular values are given by $\mathbf{\Sigma} = \text{diag}(10, 7, 0.1, 0.05)$. The input components along $\mathbf{v}_1$ and $\mathbf{v}_2$ are amplified and come out mostly along the plane spanned by $\mathbf{u}_1$ and $\mathbf{u}_2$. Those directions result in a high gain plane. The same argumentation holds for the third and fourth component and the resulting low gain plane. The matrix gain varies between 10 and 0.05, which means we have a ratio of 200 : 1.

In a control problem, singular values measure actuator efficiency. We can't really control the output direction $\mathbf{u}_4$, because any change of the input signal won't change much. However, in an estimation problem the singular values measure the sensor sensitivity or uncertainty. Sensors appear to be 200 times more sensitive in $\mathbf{v}_1$ direction than in $\mathbf{v}_4$ direction.

This also gives us the quantitative reason why it is sometimes fine to speak of the effective or numerical rank of a matrix. Clearly, in the above example, the 4×4 matrix $\mathbf{A}$ has four nonzero singular values. Therefore, its rank is 4, *i.e.*, the matrix has full rank. However, for some applications it is safe to say the effective rank is only 2. This is because all actions taken by us basically affect only a two-dimensional subspace of $\mathbb{R}^4$. The SVD might serve as a “true-rank-revealer”. See also problem 1.2 for an application of this principle.

1.3 Derivatives of functions

The general notation for a function is given by

$$f : A \to B. \tag{1.48}$$

This means f is a function on the set A into the set B. For example $f : \mathbb{R}^n \to \mathbb{R}^m$ maps all n-vectors into m-vectors.

However, we will frequently use a small exception to this general definition. We will not always require that the function f is defined for the whole set A, but only for an appropriate subset $C \subseteq A$. As an example consider the logarithmic function $f(x) = \log x$. Ignoring the complex case, the domain of this function is given by the set of strictly positive real numbers, that is, $x \in \mathbb{R}_{++} \subseteq \mathbb{R}$. We can also think of an analogy from programming languages, where you can define a function by its syntax. This does not mean that the defined function produces valid output for arbitrary input data.

1.3.1 First derivative and gradient

Suppose we have a function $f : \mathbb{R}^n \to \mathbb{R}^m$. On a side note, the function f itself could also represent a complex function, since the mapping from $\mathbb{C}^n \to \mathbb{R}^{2n}$ is isomorph. The first derivative of f is defined as

$$Df(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \frac{\partial f_m(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{bmatrix}, \tag{1.49}$$

where $Df(\mathbf{x}) \in \mathbb{R}^{m \times n}$ is the so-called *Jacobian matrix*. For the sake of simplicity we have omitted technical conditions on the existence of the derivative. If f is real-valued, that is, $f : \mathbb{R}^n \to \mathbb{R}$, the first derivative or *gradient* of the function is given by the vector

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix} = [Df(\mathbf{x})]^{\mathrm{T}}. \tag{1.50}$$

Each row consists of the partial derivatives of f. This is equal to the first row of the Jacobian matrix transposed. The first-order approximation of f at a point $\mathbf{z}$ is given by

$$f(\mathbf{x}) \approx f(\mathbf{z}) + \nabla f(\mathbf{z})^{\mathrm{T}}(\mathbf{x} - \mathbf{z}). \tag{1.51}$$

This relation is also known as the Taylor approximation with a linear term only. In session 3 we will establish an important relation between convex functions and the first-order approximation of a function.

Consider the following simple example. We are given a quadratic form $f : \mathbb{R}^n \to \mathbb{R}$

$$f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} x_i x_j, \tag{1.52}$$

where $\mathbf{P} \in \mathbb{R}^{n \times n}$. This matrix is not necessarily symmetric. Let us have a look at its partial derivatives for index combinations $i = k$ and $j = l$. When the index is given by $k = l$, it follows that

$$\frac{\partial}{\partial x_k}(p_{kk}x_k^2) = 2p_{kk}x_k, \quad k = 1, \ldots n. \tag{1.53}$$

Having a different index combination $k \neq l$ we can write

$$\frac{\partial}{\partial x_k}(p_{kl}x_kx_l + p_{lk}x_kx_l) = (p_{kl} + p_{lk})x_l, \quad k, l = 1, \ldots n. \tag{1.54}$$

Note, that the matrix entries p_{kl} and p_{lk} are just exchanged. This exactly corresponds to a matrix transpose. Putting this together, we can write the gradient of a quadratic form $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$ with a non-symmetric real matrix $\mathbf{P}$ as

$$\nabla f(\mathbf{x}) = (\mathbf{P} + \mathbf{P}^{\mathrm{T}})\mathbf{x}. \tag{1.55}$$

In many practical cases we can assume the matrix $\mathbf{P}$ to be symmetric. In that case the gradient simplifies to $\nabla f(\mathbf{x}) = 2\mathbf{P}\mathbf{x}$, which is just an extension of the scalar case.

Chain rule for the first derivative

As in the scalar case, there exists a chain rule in order to compute the first derivative. Consider the composite function $h(\mathbf{x}) = g(f(\mathbf{x}))$, where we have $f : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^m \to \mathbb{R}^p$, and $h : \mathbb{R}^n \to \mathbb{R}^p$. The general chain rule can be stated as

$$Dh(\mathbf{x}) = Dg(f(\mathbf{x}))Df(\mathbf{x}). \tag{1.56}$$

Roughly speaking, one could read this expression as "outer derivative times inner derivative".

Let's focus on two important examples of this general chain rule. The first one occurs when f is real-valued ($f : \mathbb{R}^n \to \mathbb{R}$) and g is scalar ($g : \mathbb{R} \to \mathbb{R}$). The first derivative or gradient of this scalar composite function is then given by

$$\nabla h(\mathbf{x}) = g'(f(\mathbf{x}))\nabla f(\mathbf{x}). \tag{1.57}$$

As a second example, consider the affine[6] composition $h(\mathbf{x}) = g(\mathbf{A}\mathbf{x} + \mathbf{b})$, with real-valued $g : \mathbb{R}^m \to \mathbb{R}$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, and $\mathbf{b} \in \mathbb{R}^m$. Using the chain rule (1.56), we then have

$$\nabla h(\mathbf{x}) = [Dh(\mathbf{x})]^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}}\nabla g(\mathbf{A}\mathbf{x} + \mathbf{b}). \tag{1.58}$$

An application of the above affine composition rule is the gradient of the so-called *log-sum-exp*-function. This function arises in the context of duality of the channel capacity problem and plays a central role in the field of geometric programming. It is given by

$$f(\mathbf{x}) = \log \sum_{i=1}^{m} \exp\left(\mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i\right), \tag{1.59}$$

[6]For our purposes affine means linear plus constant.

where $\mathbf{a}_i \in \mathbb{R}^m$, and $b_i \in \mathbb{R}$. Substituting $y_i = \mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i$ leads to $g(\mathbf{y}) = \log \sum_{i=1}^m \exp y_i$. Using the chain rule (1.56), it follows that

$$\nabla g(\mathbf{y}) = \frac{1}{\sum_{i=1}^m \exp y_i} \begin{bmatrix} \exp y_1 \\ \vdots \\ \exp y_m \end{bmatrix}. \tag{1.60}$$

We will now define a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with rows $\mathbf{a}_1^{\mathrm{T}}, \dots, \mathbf{a}_m^{\mathrm{T}}$. Observe that the argument $\mathbf{y}$ can be represented by the affine expression $\mathbf{Ax} + \mathbf{b}$

$$g(\mathbf{y}) = g(y_1, \dots, y_m) = \log \sum_{i=1}^m \exp y_i, \tag{1.61}$$

$$\Rightarrow g(\mathbf{Ax} + \mathbf{b}) = g(\mathbf{a}_1^{\mathrm{T}}\mathbf{x} + b_1, \dots, \mathbf{a}_m^{\mathrm{T}}\mathbf{x} + b_m) = \log \sum_{i=1}^m \exp\left(\mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i\right) \tag{1.62}$$

We already know the gradient of equation (1.61). We can now directly apply the affine composition rule and eventually find

$$\nabla f(\mathbf{x}) = \mathbf{A}^{\mathrm{T}} \nabla g(\mathbf{Ax} + \mathbf{b}) \tag{1.63}$$

$$= \mathbf{A}^{\mathrm{T}} \frac{1}{\sum_{i=1}^m \exp\left(\mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i\right)} \begin{bmatrix} \exp\left(\mathbf{a}_1^{\mathrm{T}}\mathbf{x} + b_1\right) \\ \vdots \\ \exp\left(\mathbf{a}_m^{\mathrm{T}}\mathbf{x} + b_m\right) \end{bmatrix}. \tag{1.64}$$

1.3.2 Second derivative

Consider the real-valued function $f : \mathbb{R}^n \to \mathbb{R}$. Again, we will omit technical details about differentiability and the domain of this function. Then, the second derivative or *Hessian matrix* is given by

$$D\nabla f(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}. \tag{1.65}$$

The Hessian describes the local curvature of a multi-variable function. Using Clairaut's theorem (the order of partial derivatives can be exchanged) we can directly conclude that the Hessian matrix is a symmetric matrix. The second-order approximation of a function f at a point $\mathbf{z}$ is defined by

$$f(\mathbf{x}) \approx f(\mathbf{z}) + \nabla f(\mathbf{z})^{\mathrm{T}}(\mathbf{x} - \mathbf{z}) + (1/2)(\mathbf{x} - \mathbf{z})^{\mathrm{T}} \nabla^2 f(\mathbf{x})(\mathbf{x} - \mathbf{z}). \tag{1.66}$$

We have already mentioned a relation between gradient and convexity of a function. Not surprisingly, there also exists a relation between the Hessian of a function and its convexity property. This relation has a significant impact on convex optimization theory and practice. Keep in mind that we will look at these things in later sessions!

The gradient of a quadratic form with a real non-symmetric matrix was shown to be $\nabla f(\mathbf{x}) = (\mathbf{P} + \mathbf{P}^{\mathrm{T}})\mathbf{x}$. Calculating its Hessian results in $\nabla^2 f(\mathbf{x}) = \mathbf{P} + \mathbf{P}^{\mathrm{T}}$. Again, if $\mathbf{P}$ is symmetric, the Hessian simplifies to $\nabla^2 f(\mathbf{x}) = 2\mathbf{P}$, which appears to be similar to the scalar version of second-order derivatives.

Chain rule for second derivative

As in the gradient case, there exist chain rules for calculating the Hessian of composite functions. Due to the product structure of the first derivative, it is tedious to find the second derivative in most cases. Therefore, we will focus on two simple examples, which often occur in practice.

We start with the composition with a scalar function, that is $f : \mathbb{R}^m \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$, and $h(\mathbf{x}) = g(f(\mathbf{x}))$. The gradient is calculated as

$$\nabla h(\mathbf{x}) = g'(f(\mathbf{x}))\nabla f(\mathbf{x}). \tag{1.67}$$

Evaluating the second order derivative using the chain rule again, we find the composition rule with a scalar function

$$\nabla^2 h(\mathbf{x}) = g''(f(\mathbf{x}))\nabla f(\mathbf{x})\nabla f(\mathbf{x})^{\mathrm{T}} + g'(f(\mathbf{x}))\nabla^2 f(\mathbf{x}). \tag{1.68}$$

We can also extend the affine composition case. Suppose $f : \mathbb{R}^m \to \mathbb{R}$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, and $\mathbf{b} \in \mathbb{R}^m$. We define the real-valued function $g : \mathbb{R}^n \to \mathbb{R}$ by $g(\mathbf{x}) = f(\mathbf{A}\mathbf{x} + \mathbf{b})$. The gradient and Hessian read as

$$\nabla g(\mathbf{x}) = \mathbf{A}^{\mathrm{T}}\nabla f(\mathbf{A}\mathbf{x} + \mathbf{b}), \tag{1.69}$$

$$D\nabla g(\mathbf{x}) = \nabla^2 g(\mathbf{x}) = \mathbf{A}^{\mathrm{T}}\nabla^2 f(\mathbf{A}\mathbf{x} + \mathbf{b})\mathbf{A}. \tag{1.70}$$

1.4 Problem section

1. ***Relevance of linear systems***

   Think back to other courses in your field of study. Can you give examples of $\mathbf{y} = \mathbf{A}\mathbf{x}$?

2. ***Sensor failure detection with noise***[7]

   Suppose we have a linear model $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}$, with a known measurement matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m > n)$, the known sensor readings $\mathbf{y} \in \mathbb{R}^m$, and an unknown noise vector $\mathbf{n} \in \mathbb{R}^m$. We want to find $\mathbf{x}$. Unfortunately, several sensors may have failed. That means the sensor readings are inconsistent, *i.e.*, $\mathbf{y} \notin \mathcal{R}(\mathbf{A})$. Our job is to find out which sensor(s) has/have failed.

   (a) We will first assume that only one sensor has failed and no noise is present, i.e., $\mathbf{n} = \mathbf{0}$. Load the file `single_sensor_failure`, which contains $\mathbf{A}$ and $\mathbf{y}_1$. To check if a given vector $\mathbf{y} \in \mathcal{R}(\mathbf{A})$, you can basically use the simple MATLAB code `rank([A y]) == rank(A)`. Which sensor has failed? [Template: `s1p2a_template.m`]

   (b) We will now consider additive noise. The file `single_sensor_failure` also contains a noise vector $\mathbf{n}$. Create a noisy sensor reading by simply adding this noise vector to your previous sensor readings, that is $\mathbf{y}_{\mathrm{n}1} = \mathbf{y}_1 + \mathbf{n}$. Using the method from (a), try to figure out again which sensor has failed. Why is this method not working anymore? Give an alternative method which exploits the fact that the noise is much weaker than the signal using the singular value decomposition (`svd` in MATLAB) and explain the result. [Template: `s1p2b_template.m`]

   (c) **[bonus problem]**
   Now suppose that two sensors have failed. The file `double_sensor_failure` contains the corresponding matrix $\mathbf{A}$, together with the vectors $\mathbf{y}_2$ and $\mathbf{n}$. Propose an extended method for a double sensor failure identification based on the methods from (a) and (b). Test your method for the noiseless and noisy case. You might find the MATLAB function `combnk` useful. Which sensor pair has failed? Do you have any kind of objections for this extended method? [Template: `s1p2c_template.m`]

[7] Adopted from [12], homework problem 2.4.

3. ***Optimal time compression equalizer***[8]

Consider the communication system in Fig. 1.3, where $h(k)$ is the finite impulse response (FIR) of the channel and $g(k)$ is an FIR equalizer. We assume that $g(k)$ and $h(k)$ are nonzero only inside the interval $0 \leq k \leq N-1$. The relation between channel input $x(k)$ and output $y(k)$ is given by the discrete-time convolution

$$y(k) = g(k) * h(k) * x(k) = h_{\text{eq}}(k) * x(k), \tag{1.71}$$

where $h_{\text{eq}}(k)$ is the equalized channel response. Our goal is to find optimal equalizer coefficients $g(k)$, such that $h_{\text{eq}}(k)$ meets specific design criteria.

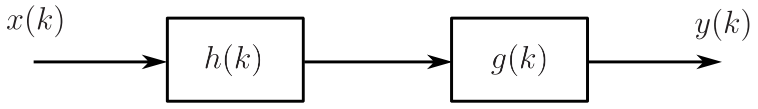


Figure 1.3: Communication system.

Note
Part (a) and (b) should be done on paper. You will only need MATLAB for part (c).

(a) The discrete-time convolution

$$h_{\text{eq}}(k) = h(k) * g(k) = \sum_{\tau=0}^{N-1} h(k-\tau)g(\tau), \quad k = 0, 1, \dots, 2N-2 \tag{1.72}$$

can be expressed as a simple matrix-vector product: $\mathbf{h}_{\text{eq}} = \mathbf{H}\mathbf{g}$. Find $\mathbf{h}_{\text{eq}}$, $\mathbf{H}$ and $\mathbf{g}$.

(b) Our design criterion is to choose $g(k)$ such that most of the energy of the equalized impulse response $h_{\text{eq}}(k)$ is concentrated around $k = N-1, N, N+1$. To state this formally, we define the total energy by

$$E_{\text{tot}} = \sum_{k=0}^{2N-2} h_{\text{eq}}^2(k), \tag{1.73}$$

and the desired concentrated energy by

$$E_{\text{des}} = \sum_{k=N-1}^{N+1} h_{\text{eq}}^2(k). \tag{1.74}$$

The *desired to total energy ratio* (DTE) is given by DTE $= E_{\text{des}}/E_{\text{tot}}$, where $E_{\text{tot}} > 0$. Find a method that maximizes the DTE. To achieve this, take the following steps:

[8]Taken from [12], homework problem 3.6.

1) Rewrite the DTE with only matrices and vectors.

2) Show, that $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ is positive semidefinite ($\mathbf{x}^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{x} \geq 0$ for all $\mathbf{x}$).

3) With the help of the matrix square root and a suitable substitution, reformulate the denominator of your DTE expression, such that it looks like the denominator of the matrix norm.

4) Applying the substitution of 3) as well as a second substitution to the numerator of the DTE, reformulate the numerator, such that the resulting DTE expression looks like the matrix norm.

5) By applying the relation given in (1.31) and back-substituting, finally find the FIR equalizer $\mathbf{g}$, which maximizes the DTE.

(c) Implement and test your method in MATLAB. A set of channel coefficients $h(k)$ can be loaded from the data file `channel_imp_response`. Use the MATLAB function `toeplitz` to create the channel matrix $\mathbf{H}$. Plot your solution $g(k)$, the equalized response $h_{\mathrm{eq}}(k)$, and give the value of your DTE. The MATLAB function that returns the square root of a matrix is `sqrtm`. The inverse of a matrix is computed with `inv`. [Template: `s1p3_template.m`]

2 Preliminaries II: Convex optimization

2.1 Convex optimization problems

In this section we define the term *convex optimization problem*, which is the kind of problem we will solve during the lab. Among all possible optimization problems only a small subset is convex, so why should we be interested in such problems? This is because most convex problems are solvable in polynomial time (or faster) with standard algorithms and software, readily available and often free. Therefore, in this lab we do not focus on solvers but rather on modeling. If you have an optimization problem to solve, it is often quite simple to find a poor model which is unsolvable in a reasonable time. Clearly it makes no sense to state an unsolvable problem. The art of convex optimization is to find a model which enables us to represent our problem as a convex problem so that we can solve it fast.

2.1.1 Terminology

An optimization problem has the form

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & f_0(\mathbf{x}) \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \\ & h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, p, \end{array} \tag{2.1}$$

where we define the following elements:

- $\mathbf{x} \in \mathbb{R}^n$ is the *optimization variable*,
- $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*,
- $f_i(\mathbf{x}) \leq 0$ are *inequality constraints* and $f_i : \mathbb{R}^n \to \mathbb{R}$ are *inequality constraint functions*,
- $h_i(\mathbf{x}) = 0$ are *equality constraints* and $h_i : \mathbb{R}^n \to \mathbb{R}$ are *equality constraint functions.*

If there are no constraints, the problem (2.1) is called *unconstrained.* The goal of the problem (2.1) is to find the vector $\mathbf{x}$ which minimizes $f_0(\mathbf{x})$ and verifies the constraints $f_i(\mathbf{x}) \leq 0$ for $i = 1, \ldots, m$ and $h_j(\mathbf{x}) = 0$ for $j = 1, \ldots, p$. The problem form (2.1) is very general. An optimization problem might not directly have this form, but in most cases it is very easy to transform it into a problem equivalent to (2.1),*i.e.*, it has the same solution. For example, the constraint $2x_1 \geq 3$ can be rewritten as $-2x_1+3 \leq 0$.

We define the *domain* of the problem (2.1) as the set

$$\mathcal{D} = \bigcap_{i=0}^{m} \text{dom } f_i \cap \bigcap_{j=1}^{p} \text{dom } h_j, \tag{2.2}$$

i.e., the set of points for which the objective and all constraint functions are defined. A point $\mathbf{x} \in \mathcal{D}$ is *feasible* if it satisfies all constraints. The problem (2.1) is feasible (otherwise unfeasible) if there exists at least one feasible point. The set of all feasible points is the *feasible set.*

We define the *optimal value* $p^\star$ of the problem (2.1) as

$$p^\star = \inf \{f_0(\mathbf{x}) \mid f_i(\mathbf{x}) \leq 0,\ i = 1, \dots, m,\ h_j(\mathbf{x}) = 0,\ j = 1, \dots, p\}. \tag{2.3}$$

If the problem is infeasible, $p^\star = \infty$. If $p^\star = -\infty$, the problem is *unbounded below.* The point $\mathbf{x}^\star$ such that $f_0(\mathbf{x}^\star) = p^\star$ is called the *optimal point.* If there exists an optimal point, the problem is called *solvable.* To get some intuition, imagine we want to build a product and for that we need to buy some materials. f_0 is our cost function, *i.e.*, we want to minimize the price we will pay in total for constructing our product. The functions f_i are constraints on how much materials we need. If $p^\star = \infty$, it means that we did unbelievably bad. We cannot get the product finished even by spending an infinite amount of money. If $p^\star = -\infty$ however our model must be wrong since it would mean that we generate an infinite amount of money while buying materials.

If the objective function f_0 is identically zero, we have

$$\begin{array}{ll} p^\star = 0 & \text{if the problem is feasible} \\ p^\star = \infty & \text{otherwise.} \end{array} \tag{2.4}$$

Such a problem is called a *feasibility problem* which can be written as

$$\begin{array}{lll} \text{find} & \mathbf{x} & \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, & i = 1, \dots, m \\ & h_j(\mathbf{x}) = 0, & j = 1, \dots, p. \end{array} \tag{2.5}$$

2.1.2 Definition of convex optimization problems

A *convex optimization problem* has the form

$$\begin{array}{lll} \underset{\mathbf{x}}{\text{minimize}} & f_0(\mathbf{x}) & \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, & i = 1, \dots, m \\ & \mathbf{a}_j^{\mathrm{T}}\mathbf{x} = b_j, & j = 1, \dots, p, \end{array} \tag{2.6}$$

where $f_i(\mathbf{x})|_{i=0}^{m}$ are convex functions. In other words, a convex optimization problem is an optimization problem with following additional constraints:

- the objective function f_0 must be convex,
- the inequality constraint functions f_i must be convex,
- the equality constraint functions must be affine, *i.e.*, $h_j = \mathbf{a}_j^{\mathrm{T}}\mathbf{x} - b_j$.

All along this lab our goal will be to write problems in this form so that we can solve them efficiently with MATLAB. Sometimes the convex formulation is simple to obtain but sometimes it takes a lot of work to find it. This is the most interesting part of convex optimization and actually this is what research is about : given a problem that we want to solve, how can we write it in a convex form? Sometimes it is not possible but we can find very good solutions by approximating our problem by a convex problem. This is called *convex relaxation* and we will see this during the lab.

2.2 Convex sets

To model and solve convex problems efficiently, it will be very important to recognize that a problem is actually convex. In order to achieve that, we need to develop the theory of convex sets (this section) and convex functions (the next section).

2.2.1 Definition

A set $\mathcal{C}$ is *convex* if all line segments starting in $\mathcal{C}$ and ending in $\mathcal{C}$ are included in the set. We visualize this geometrical interpretation in Fig. 2.1.

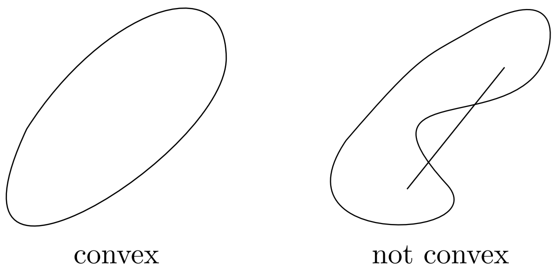


Figure 2.1: Geometrical interpretation of convexity

Mathematically, a line segment between two points $\mathbf{x}_1$ and $\mathbf{x}_2$ has the form

$$\{\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2 \mid \theta \in \mathbb{R},\ 0 \leq \theta \leq 1\}. \tag{2.7}$$

A set $\mathcal{C}$ is *convex* if for any two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}$

$$\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2 \in \mathcal{C}, \tag{2.8}$$

with $\theta \in \mathbb{R}$, $0 \leq \theta \leq 1$.

We can give a more general definition of convex sets using convex combinations. The convex combination of the points $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathcal{C}$ is given by

$$\sum_{i=1}^{k} \theta_i \mathbf{x}_i, \tag{2.9}$$

where $\theta_1, \ldots, \theta_k \in \mathbb{R}$, $\sum_{i=1}^{k} \theta_i = 1$. We then define a set as convex if all convex combinations of points of this set are included in the set.

Furthermore, for any set $\mathcal{S}$ (convex or not), we define the *convex hull* of $\mathcal{S}$ as the set of all convex combinations of points in $\mathcal{S}$. We visualize an example of a convex hull in Fig. 2.2. The convex hull of a set $\mathcal{S}$ is the smallest convex set that contains $\mathcal{S}$. If a set is equal to its convex hull, it is obviously convex.

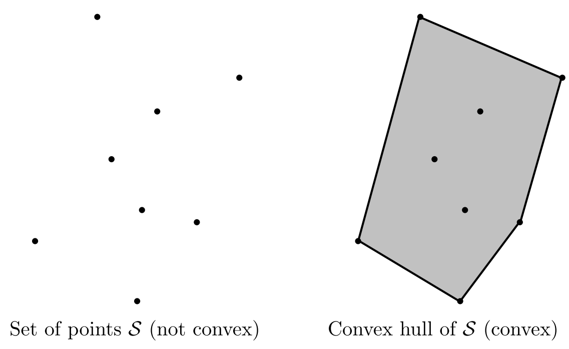

Set of points $\mathcal{S}$ (not convex)
Convex hull of $\mathcal{S}$ (convex)

Figure 2.2: Convex hull (in gray) of a set composed of 8 points.

2.2.2 Examples of convex sets

In this section we present some examples of convex sets, which will be important since they are used to model many problems.

Hyperplanes and halfspaces

A *hyperplane* is a set of the form

$$\mathcal{H} = \{\mathbf{x} \mid \mathbf{a}^{\mathrm{T}}\mathbf{x} = b\}, \tag{2.10}$$

with $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{a} \neq \mathbf{0}$ and $b \in \mathbb{R}$. We can readily see that a hyperplane $\mathcal{H}$ is convex by taking two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{H}$. These points verify $\mathbf{a}^{\mathrm{T}}\mathbf{x_1} = b$ and $\mathbf{a}^{\mathrm{T}}\mathbf{x_2} = b$. Now we have to show that all the points on the line segment between $\mathbf{x}_1$ and $\mathbf{x}_2$ are in $\mathcal{H}$. We compute

$$\begin{aligned}
\mathbf{a}^{\mathrm{T}}(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) &= \theta\mathbf{a}^{\mathrm{T}}\mathbf{x}_1 + (1-\theta)\mathbf{a}^{\mathrm{T}}\mathbf{x}_2 \\
&= \theta b + (1-\theta)b \\
&= b,
\end{aligned}$$

which shows that $\theta\mathbf{x_1} + (1-\theta)\mathbf{x_2}$ is in $\mathcal{H}$ and that $\mathcal{H}$ is convex. A hyperplane $\mathcal{H}$ has another representation of the form

$$\mathcal{H} = \{\mathbf{x} \mid \mathbf{a}^{\mathrm{T}}(\mathbf{x} - \mathbf{x_0}) = 0\}, \tag{2.11}$$

where $\mathbf{x_0}$ is any point on the hyperplane, *i.e.* $\mathbf{a}^{\mathrm{T}}\mathbf{x_0} = b$.

A *halfspace* is a set of the form

$$\mathcal{H}_s = \{\mathbf{x} \mid \mathbf{a}^{\mathrm{T}}\mathbf{x} \leq b\}. \tag{2.12}$$

A hyperplane divides $\mathbb{R}^n$ into two halfspaces, see Fig. 2.3.

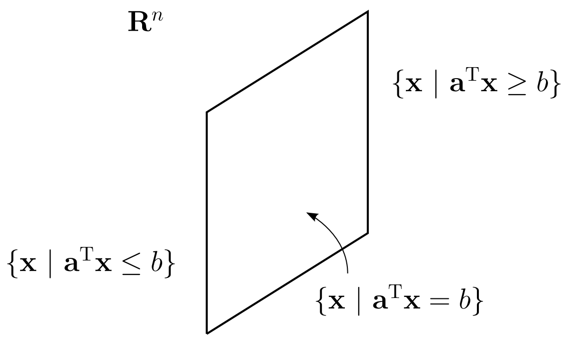

$\mathbf{R}^n$
$\{\mathbf{x} \mid \mathbf{a}^{\mathrm{T}}\mathbf{x} \geq b\}$
$\{\mathbf{x} \mid \mathbf{a}^{\mathrm{T}}\mathbf{x} \leq b\}$
$\{\mathbf{x} \mid \mathbf{a}^{\mathrm{T}}\mathbf{x} = b\}$

Figure 2.3: A hyperplane separating $\mathbb{R}^n$ into two hyperspaces.

Balls and ellipsoids

A *ball* is a set of the form

$$\mathcal{B}(\mathbf{x_c}, r) = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x_c}\| \leq r\}, \tag{2.13}$$

with $\mathbf{x_c} \in \mathbb{R}^n$ being the center of the ball and $r \in \mathbb{R}$ the radius of the ball. The expression $\|\mathbf{y}\|$ represents the vector norm of $\mathbf{y}$. Many different vector norms exist, for example

- $\|\mathbf{y}\|_1 = \sum_{i=1}^{n} |y_i|$,
- $\|\mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{n} y_i^2}$,
- $\|\mathbf{y}\|_\infty = \max\,\{y_1, \ldots, y_n\}$.

In Fig. 2.4, we illustrate different norm balls.

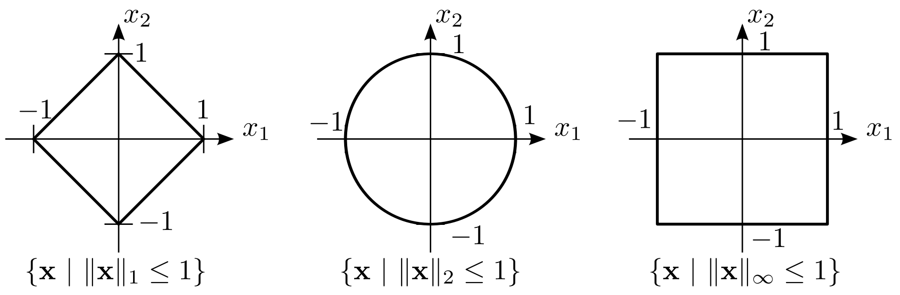

x_2
1
-1
1
x_1
-1
$\{\mathbf{x} \mid \|\mathbf{x}\|_1 \leq 1\}$
$\{\mathbf{x} \mid \|\mathbf{x}\|_2 \leq 1\}$
$\{\mathbf{x} \mid \|\mathbf{x}\|_\infty \leq 1\}$

Figure 2.4: 1-, 2-, ∞-norm balls with center $(0, 0)$ and radius 1

Alternatively a ball can be written as

$$\mathcal{B}(\mathbf{x_c}, r) = \{\mathbf{x_c} + r\mathbf{u} \mid \|\mathbf{u}\| \leq 1\}. \tag{2.14}$$

An *ellipsoid* is a convex set which can always be represented as

$$\mathcal{E} = \{\mathbf{x_c} + \mathbf{A}\mathbf{u} \mid \|\mathbf{u}\| \leq 1\}, \tag{2.15}$$

with $\mathbf{A}$ square and nonsingular. Equivalently $\mathcal{E}$ can be written as

$$\mathcal{E} = \{\mathbf{x} \mid (\mathbf{x} - \mathbf{x_c})^{\mathrm{T}}\mathbf{P}^{-1}(\mathbf{x} - \mathbf{x_c}) \leq 1\}, \tag{2.16}$$

with $\mathbf{P} = \mathbf{P}^{\mathrm{T}}$ (*i.e.*, $\mathbf{P}$ is symmetric) and $\mathbf{P} \succ 0$ (*i.e.*, $\mathbf{P}$ is positive definite). The length of the semiaxes of the ellipsoid are $\sqrt{\lambda_i(\mathbf{P})}$ with $\lambda_i(\mathbf{P})$ the eigenvalues of $\mathbf{P}$. We can easily go from the first representation to the second one : a vector $\mathbf{x}$ is in $\mathcal{E}$ if $\mathbf{x} = \mathbf{x_c} + \mathbf{A}\mathbf{u}$ and $\|\mathbf{u}\| \leq 1$, in the case of the Euclidean norm it follows

$$\begin{aligned}
\|\mathbf{u}\|_2 &\leq 1 \\
\|\mathbf{A}^{-1}(\mathbf{x} - \mathbf{x_c})\|_2 &\leq 1 \\
(\mathbf{x} - \mathbf{x_c})^{\mathrm{T}}\mathbf{A}^{-T}\mathbf{A}^{-1}(\mathbf{x} - \mathbf{x_c}) &\leq 1 \\
(\mathbf{x} - \mathbf{x_c})^{\mathrm{T}}(\mathbf{A}\mathbf{A}^{\mathrm{T}})^{-1}(\mathbf{x} - \mathbf{x_c}) &\leq 1 \\
(\mathbf{x} - \mathbf{x_c})^{\mathrm{T}}\mathbf{P}^{-1}(\mathbf{x} - \mathbf{x_c}) &\leq 1
\end{aligned}$$

with $\mathbf{A} = \mathbf{P}^{1/2}$.

Polyhedra

A *polyhedron* is a set of the form

$$\mathcal{P} = \{\mathbf{x} \mid \mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i,\ i = 1, \ldots, m,\ \mathbf{c}_j^{\mathrm{T}}\mathbf{x} = d_j,\ j = 1, \ldots, p\}, \tag{2.17}$$

with $\mathbf{a}_i, \mathbf{c}_j \in \mathbb{R}^n$ and $b_i, d_j \in \mathbb{R}$. A polyhedron can be written more compactly as

$$\mathcal{P} = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{C}\mathbf{x} = \mathbf{d}\}, \tag{2.18}$$

with $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{C} \in \mathbb{R}^{p \times n}$ and $\mathbf{b}, \mathbf{d} \in \mathbb{R}^n$. Note that a polyhedron is an intersection of hyperplanes and halfspaces.

2.2.3 Cones

Definition

A set $\mathcal{C}$ is a *cone* if for any $\mathbf{x} \in \mathcal{C}$ and $\theta \geq 0$ we have

$$\theta\mathbf{x} \in \mathcal{C}. \tag{2.19}$$

In Fig. 2.5, we illustrate examples of cones. We will see later that cones are very interesting sets because they can define generalized inequalities. Note, that cones are not convex by definition. A *convex cone* is a set $\mathcal{C}$ such that for any two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}$, $\theta_1, \theta_2 \geq 0$,

$$\theta_1\mathbf{x}_1 + \theta_2\mathbf{x}_2 \in \mathcal{C}. \tag{2.20}$$

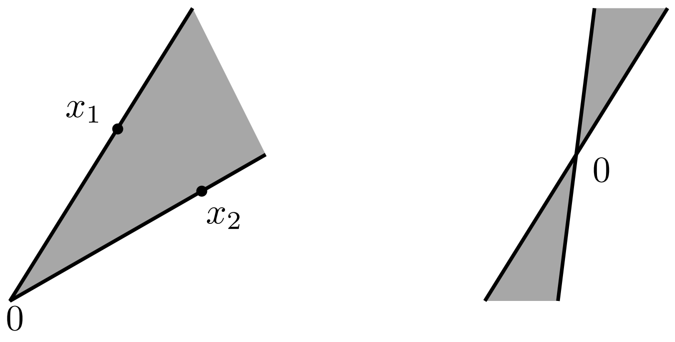


Figure 2.5: 2 cones (gray area)

The cone on the left hand side of Fig. 2.5 is convex, the cone on the right hand side is not. A famous convex cone is the nonnegative orthant $\mathbb{R}^n_+$. It actually defines the usual elementwise inequality as we will see in section 2.2.4.

We call the *conic hull* of a set $\mathcal{S}$ the set all conic combinations, *i.e.*,

$$\sum_{i=1}^{k} \theta_i \mathbf{x}_i,\ \theta_i \geq 0,\ i = 1, \ldots, k$$

of points in $\mathcal{S}$. The conic hull is the smallest convex cone that contains $\mathcal{S}$. In the next sections we give two examples of convex cones which will be very useful.

Norm cones

A *norm cone* is a set $\mathcal{C}$ of the form

$$\mathcal{C} = \{\mathbf{x} \mid \|\mathbf{x}\| \leq t\} \subset \mathbb{R}^{n+1}, \tag{2.21}$$

where $\|\mathbf{x}\|$ can be any vector norm. The *second-order cone* or *Lorentz* cone is a norm cone defined with the euclidean norm, *i.e.*, of the form

$$\mathcal{C} = \{\mathbf{x} \mid \|\mathbf{x}\|_2 \leq t\} \subset \mathbb{R}^{n+1}, \tag{2.22}$$

which is illustrated in Fig 2.6.

Positive semidefinite cone

We denote $\mathbb{S}^n$ the set of $n \times n$ symmetric matrices given by

$$\mathbb{S}^n = \{\mathbf{X} \in \mathbb{R}^{n \times n} \mid \mathbf{X} = \mathbf{X}^{\mathrm{T}}\}. \tag{2.23}$$

We denote $\mathbb{S}^n_{++}$ the set of $n \times n$ symmetric positive definite matrices given by

$$\mathbb{S}^n_{++} = \{\mathbf{X} \in \mathbb{S}^n \mid \mathbf{X} \succ 0\}. \tag{2.24}$$

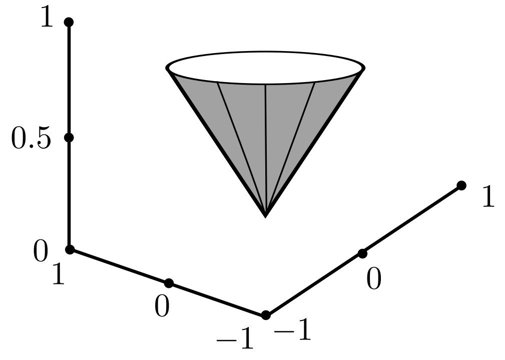


Figure 2.6: Lorentz cone: $\sqrt{x_1^2 + x_2^2} \leq t$

We denote $\mathbb{S}_+^n$ the set of $n \times n$ symmetric positive semidefinite matrices given by

$$\mathbb{S}_+^n = \{\mathbf{X} \in \mathbb{S}^n \mid \mathbf{X} \succcurlyeq 0\}. \tag{2.25}$$

The set $\mathbb{S}_+^n$ is a convex cone. To see this, let us use the definition for positive semidefinite matrices : $\mathbf{X} \succcurlyeq 0 \Leftrightarrow \mathbf{y}^{\mathrm{T}}\mathbf{X}\mathbf{y} \geq 0, \forall \mathbf{y} \in \mathbb{R}^n$. Given $\mathbf{X}_1 \succcurlyeq 0$, $\mathbf{X}_2 \succcurlyeq 0$ and $\theta_1, \theta_2 \geq 0$ we compute

$$\begin{aligned} \mathbf{y}^{\mathrm{T}}(\theta_1\mathbf{X}_1 + \theta_2\mathbf{X}_2)\mathbf{y} &= \theta_1 \underbrace{\mathbf{y}^{\mathrm{T}}\mathbf{X}_1\mathbf{y}}_{\geq 0} + \theta_2 \underbrace{\mathbf{y}^{\mathrm{T}}\mathbf{X}_2\mathbf{y}}_{\geq 0} \\ &\geq 0, \end{aligned}$$

i.e., $\theta_1\mathbf{X}_1 + \theta_2\mathbf{X}_2 \succcurlyeq 0$, which shows that $\mathbb{S}_+^n$ is a convex cone. Analogously, this can be shown for positive definite matrices.

2.2.4 Generalized inequalities

Most of the time, thinking about inequalities means thinking about the total ordering of $\mathbb{R}$ under $\leq$, *i.e.*, $x \leq y \Leftrightarrow x - y \leq 0$. There exist other inequalities, called *generalized inequalities.* They are very useful since we can use them to define new kinds of optimization problems, *e.g.*, second-order cone problems and semidefinite problems. These inequalities are based on *proper cones.*

A cone $\mathcal{K} \subseteq \mathbb{R}^n$ is a *proper cone* if

1. $\mathcal{K}$ is convex,
2. $\mathcal{K}$ is closed,
3. $\mathcal{K}$ has nonempty interior,
4. $\mathcal{K}$ contains no line, *i.e.*, $\mathbf{x} \in \mathcal{K}, -\mathbf{x} \in \mathcal{K} \Leftrightarrow \mathbf{x} = \mathbf{0}$.

We can define a generalized inequality using a proper cone associated to a partial ordering in $\mathbb{R}^n$ as follow:

$$\mathbf{x} \preccurlyeq_{\mathcal{K}} \mathbf{y} \Leftrightarrow \mathbf{y} - \mathbf{x} \in \mathcal{K}. \tag{2.26}$$

We can also define a generalized inequality using a proper cone associated to a strict ordering in $\mathbb{R}^n$ as

$$\mathbf{x} \prec_{\mathcal{K}} \mathbf{y} \Leftrightarrow \mathbf{y} - \mathbf{x} \in \text{int } \mathcal{K}, \tag{2.27}$$

where int $\mathcal{K}$ is the interior of $\mathcal{K}$. The nonnegative orthant $\mathbb{R}^n_+$ is a proper cone and we can define an inequality based on it. We have $\mathbf{x} \preccurlyeq_{\mathbb{R}^n_+} \mathbf{y} \Leftrightarrow \mathbf{y} - \mathbf{x} \in \mathbb{R}^n_+$, where $\preccurlyeq_{\mathbb{R}^n_+}$ is the standard componentwise inequality for vectors, *i.e.*, if all components of a vector $\mathbf{y}$ are greater than those of a vector $\mathbf{x}$, then $\mathbf{y} - \mathbf{x}$ is in the nonnegative orthant.

The semidefinite cone is a proper cone and defines so called matrix inequalities. We have $\mathbf{X} \preccurlyeq_{\mathbb{S}^n_+} \mathbf{Y} \Leftrightarrow \mathbf{Y} - \mathbf{X} \succcurlyeq 0$. In the following we simply drop the subscript and write $\mathbf{X} \preccurlyeq \mathbf{Y}$ meaning that $\mathbf{Y} - \mathbf{X}$ is positive semidefinite.

Most of the properties of standard inequalities hold for generalized inequalities. However, a few properties do not hold due to the fact that the ordering is not linear, *i.e.*, two values are not necessarily comparable, e.g., $[2 \;\; -1]^{\mathrm{T}}$ is neither greater nor smaller than $[1 \; 3]^{\mathrm{T}}$ in the $\mathbb{R}^n_+$ cone.

2.2.5 Operations that preserve convexity

To check whether a set $\mathcal{S}$ is convex or not, we can use the basic definition of convex sets (2.8). This definition is not always simple to apply and the following method is often more powerful. Let us write $\mathcal{S}$ as a function f of another convex set $\mathcal{C}$ which we know is convex, *i.e.*, $\mathcal{S} = f(\mathcal{C})$. If $\mathcal{C}$ is convex and the function f preserves convexity, we know that $\mathcal{S}$ is convex. The following two operations preserve convexity

Intersection

Intersection preserves convexity. Given two convex sets $\mathcal{C}_1$ and $\mathcal{C}_2$, $\mathcal{C}_1 \cap \mathcal{C}_2$ is convex.

Affine functions

Affine functions preserve convexity. Given f of the form

$$f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}, \tag{2.28}$$

with $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. If a set $\mathcal{C}$ is convex, then its image under f,

$$f(\mathcal{C}) = \{f(\mathbf{x}) \mid \mathbf{x} \in \mathcal{C}\}, \tag{2.29}$$

is convex.

2.3 Convex functions

2.3.1 Definition

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if its domain is a convex set and if for any $\mathbf{x}, \mathbf{y} \in \mathrm{dom}\ f$ and θ with $0 \leq \theta \leq 1$ we have

$$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y}). \tag{2.30}$$

We illustrate a convex function in Fig. 2.7.

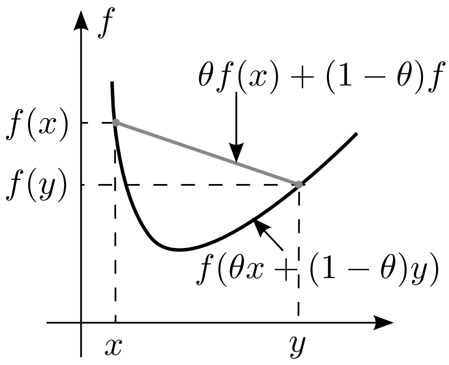


Figure 2.7: Convex function (in black) and a line segment between $\mathbf{x}$ and $\mathbf{y}$ (in gray)

A function f is *concave* if the function $-f$ is convex.

The first order condition for a function f to be convex is the following. If f is *differentiable*, then f is convex if and only if its domain is convex and for any $\mathbf{x}, \mathbf{y} \in \mathrm{dom}\ f$

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^{\mathrm{T}}(\mathbf{y} - \mathbf{x}), \tag{2.31}$$

where ∇ is the gradient as defined in (1.50). We omit the proof of this theorem. Interestingly, if $\nabla f(\mathbf{x}) = 0$ then $f(\mathbf{y}) \geq f(\mathbf{x})$, in other words $\mathbf{x}$ minimizes the function f. This is a very important property of convex functions. From a local observation, one can deduct a global fact on f, *i.e.*, $\mathbf{x}$ is a global minimizer of f.

The second order condition for a function f to be convex is the following. If f is twice differentiable, then f is convex if and only if its domain is convex and for any $\mathbf{x} \in \mathrm{dom}\ f$ we have

$$\nabla^2 f(\mathbf{x}) \succcurlyeq 0, \tag{2.32}$$

where $\nabla^2 f(\mathbf{x})$ is the Hessian of f as defined in (1.65) and $\succcurlyeq$ is a matrix inequality, *i.e.*, $\nabla^2 f(\mathbf{x}) \in \mathbb{S}^n_+$. We omit the proof of this theorem.

Finally we give a last useful condition for convexity. A function f is convex if and only if it is convex when restricted to any line that intersects its domain, *i.e.*,

$$g(t) = f(\mathbf{x} + t\mathbf{v}) \tag{2.33}$$

is convex for all t such that $\mathbf{x}+t\mathbf{v} \in \text{dom } f$. This property is practically very important. If you want to check if a set is not convex, just draw a line which intersects the domain and look at the shape of the function along the line. If the function does not have nonnegative curvature then it is not convex.

2.3.2 Examples of convex and concave functions

Note, that convexity is simple to show for some of these examples but quite difficult for some others.

- e^{ax} is convex on $\mathbb{R}$, $\forall a \in \mathbb{R}$
- x^a is convex on $\mathbb{R}_{++}$, for $a \geq 1$ or $a \leq 0$
- x^a is concave on $\mathbb{R}_{++}$, for $0 \leq a \leq 1$
- $|x|^p$ is convex on $\mathbb{R}$, for $p \geq 1$
- $\log x$ is concave on $\mathbb{R}_{++}$
- $x\log x$ is convex on $\mathbb{R}_{+}$ $(0\log 0 = 0)$
- $\|\mathbf{x}\|$ is convex on $\mathbb{R}^n$
- $f(\mathbf{x}) = \max\{x_1, \dots, x_n\}$ is convex on $\mathbb{R}^n$
- $f(x, y) = x^2/y$ with $\text{dom } f = \mathbb{R} \times \mathbb{R}_{++}$ is convex
- $f(\mathbf{x}) = \log(e^{x_1} + \cdots + e^{x_n})$ is convex on $\mathbb{R}^n$
- $f(\mathbf{x}) = \left(\prod_{i=1}^n x_i\right)^{1/n}$ is concave on $\mathbb{R}^n_{++}$
- $f(\mathbf{X}) = \log \det(\mathbf{X})$ is concave on $\mathbb{S}^n_{++}$

2.3.3 Sublevel sets

The α-sublevel set of $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\mathcal{C}_\alpha = \{x \in \text{dom } f \mid f(x) \leq \alpha\}. \tag{2.34}$$

An example of sublevel sets is illustrated in Fig. 2.8. It holds that the sublevel sets of a convex function are convex.

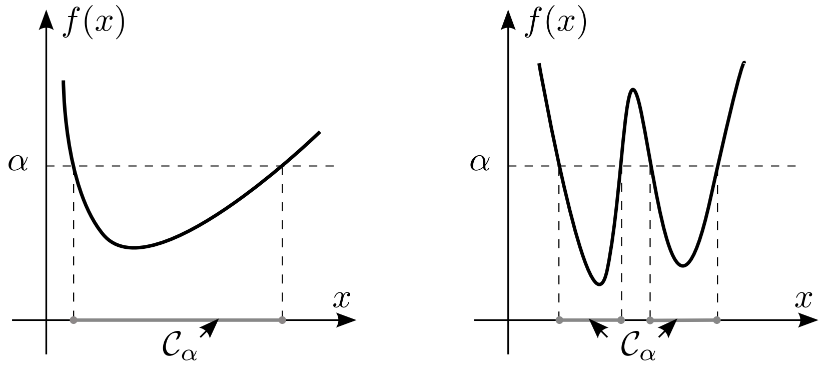


Figure 2.8: Sublevel set (in gray) for a convex function (left) and a nonconvex function (right)

2.3.4 Epigraph

The *graph* of a function $f : \mathbb{R}^n \to \mathbb{R}$ is

$$\{(x, f(x)) \mid x \in \text{dom } f\}. \tag{2.35}$$

The *epigraph* of a function $f : \mathbb{R}^n \to \mathbb{R}$ is

$$\text{epi } f = \{(x, t) \mid x \in \text{dom } f,\ f(x) \leq t\}. \tag{2.36}$$

The epigraph gives us a direct link between convex sets and convex functions. A function is convex if and only if its epigraph is a convex set. This will be useful to prove that a function is convex. Equivalently, the *hypograph* of a function $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$\text{hypo } f = \{(x, t) \mid x \in \text{dom } f,\ f(x) \geq t\}. \tag{2.37}$$

A function is concave if and only if its hypograph is a convex set.

2.3.5 Operations that preserve convexity

As for convex sets, it is very useful to know which operations preserve convexity.

Nonnegative weighted sum

A nonnegative weighted sum f of convex functions $f_1, \ldots f_m$ given by

$$f = w_1 f_1 + \cdots + w_m f_m, \tag{2.38}$$

with $w_1, \ldots, w_m \geq 0$ is convex.

Composition with an affine mapping

Assume the function $g : \mathbb{R}^n \to \mathbb{R}$ defined as

$$g(\mathbf{x}) = f(\mathbf{A}\mathbf{x} + \mathbf{b}), \tag{2.39}$$

with $f : \mathbb{R}^m \to \mathbb{R}$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$. The domain of g is $\{\mathbf{x} \mid \mathbf{A}\mathbf{x} + \mathbf{b} \in \text{dom } f\}$. If f is convex, then g is convex. If f is concave, then g is concave.

Pointwise maximum and supremum

Given the function f, the *pointwise maximum* of several functions $f_1, \dots, f_m$ defined as

$$f(x) = \max\{f_1(x), \dots, f_m(x)\}, \tag{2.40}$$

if the functions $f_1, \dots, f_m$ are all convex then f is convex.

Example: The *piecewise-linear* function

$$f(\mathbf{x}) = \max\{\mathbf{a}_1^{\mathrm{T}}\mathbf{x} + b_1, \dots, \mathbf{a}_k^{\mathrm{T}}\mathbf{x} + b_k\}$$

is convex. You can also verify this fact immediately since f can be written as $\|\mathbf{A}\mathbf{x}+\mathbf{b}\|_\infty$, which is obviously convex.

Given the function g, the *pointwise supremum* of a function f is defined as

$$g(x) = \sup_{y\in\mathcal{A}} f(x, y). \tag{2.41}$$

If f is convex in x for any y in $\mathcal{A}$, then g is convex. The domain of g is

$$\{x \mid (x, y) \in \text{dom } f \ \forall y \in \mathcal{A}, \sup_{y\in\mathcal{A}} f(x, y) < \infty\}.$$

Example: The *maximum eigenvalue of a symmetric matrix* defined as

$$f(\mathbf{X}) = \lambda_{\max}(\mathbf{X}), \ \mathbf{X} \in \mathbb{S}^n, \tag{2.42}$$

is convex. To see this we write f as

$$f(\mathbf{X}) = \sup\{\mathbf{y}^{\mathrm{T}}\mathbf{X}\mathbf{y}, \ \|y\|_2 = 1\}.$$

This is a pointwise supremum of a function linear in $\mathbf{X}$, and thus convex. Note that we can write f like this since λ is an eigenvalue of $\mathbf{X}$ if

$$\begin{aligned} & \exists\mathbf{y} : \mathbf{X}\mathbf{y} = \lambda\mathbf{y} \\ \Leftrightarrow \quad & \mathbf{y}^{\mathrm{T}}\mathbf{X}\mathbf{y} = \lambda\mathbf{y}^{\mathrm{T}}\mathbf{y}. \end{aligned}$$

If $\|\mathbf{y}\|_2 = 1$ then $\mathbf{y}^{\mathrm{T}}\mathbf{X}\mathbf{y} = \lambda$.

Composition

We define $h : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}$ and $f : \mathbb{R}^n \to \mathbb{R}$ such that $f = h \circ g$, *i.e.*,

$$f(x) = h(g(x)), \ \operatorname{dom} f = \{x \in \operatorname{dom} g \mid g(x) \in \operatorname{dom} h\}. \tag{2.43}$$

- If h is convex, nondecreasing and g is convex then f is convex
- If h is convex, nonincreasing and g is concave then f is convex
- If h is concave, nondecreasing and g is concave then f is concave
- If h is concave, nonincreasing and g is convex then f is concave

Minimization

Given a function f and a convex nonempty set $\mathcal{C}$ we define the function g as

$$g(x) = \inf_{y \in \mathcal{C}} f(x, y). \tag{2.44}$$

If f is convex and $g(x) > -\infty$ for some x then g is convex.

2.4 Problem section

1. ***Solving convex optimization problems with CVX.***

   Designed by Michael Grant and Stephen Boyd, CVX is a free modeling system for *disciplined convex programming* (DCP). It enables us to use the classic MATLAB syntax in order to solve convex optimization problems satisfying the DCP ruleset. DCP is a methodology for constructing convex optimization problems proposed by Michael Grant, Stephen Boyd, and Yinyu Ye [9]. It consists of a number of rules (DCP ruleset) which have to be applied so that an implemented optimization problem is automatically convex. In other words, if CVX accepts an optimization problem, this problem must be convex. On the contrary, if CVX rejects a problem it still could be convex. Hence every convex optimization problem has to be formulated in accordance to DCP guidelines in order to be processed by CVX. The complete DCP ruleset can be found in the CVX user manual at `http://web.cvxr.com/cvx/doc`.

   The strongest point of CVX is that it immensely simplifies the programming of a convex optimization problem. We take the following example

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & 2x_1 - 3x_2 + x_3 \\ \text{subject to} & x_1 + x_2 - x_3 \leq 7 \\ & x_1 + 4x_2 + 2x_3 = 3 \\ & -2x_1 + x_2 + 4x_3 = 1 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, \end{array} \tag{2.45}$$

   where $\mathbf{x} := [x_1; x_2; x_3]$. This problem can be programmed in MATLAB using CVX as

```
c = [2; -3; 1];
G = [1 1 -1; -1 0 0; 0 -1 0; 0 0 -1];
h = [7; 0; 0; 0];
A = [1 4 2; -2 1 4];
b = [3; 1];

cvx_begin
    variable x(3);
    minimize (c'*x);
    subject to
        G*x <= h;
        A*x == b;
cvx_end
```

   (a) In the problem (2.45), identify the functions f_i and h_j (refer to the script, equation 2.1).

(b) The feasible set of an optimization problem is the intersection of the feasible sets resulting from each of the constraints. Identify the type of the corresponding sets resulting from each of the constraints in (2.45). Are they convex sets? What is the type of the objective function?

(c) Is this problem convex?

(d) Solve the problem (2.45) using CVX by typing and executing the above problem into your script.

- Note: After executing a problem via CVX, the solution status (whether the problem has been successfully solved or whether it is infeasible, or has been solved with reduced accuracy, ...) is stored in a CVX-specific variable `cvx_status`. Furthermore, the obtained optimal objective value is stored in `cvx_optval`.
- Check the solution status and the obtained optimal objective value by checking the value of `cvx_status` and `cvx_optval`.
- What is $\mathbf{x}^\star$? Calculate the resulting value of the objective using the obtained $\mathbf{x}^\star$. Does this value match the CVX variable `cvx_optval`?

[Template: `s2p1_template.m`]

2. ***Convex sets.***

(a) A very practical way to show that a set $\mathcal{C}$ is convex is to show that its intersection with any line of the form $\{\mathbf{u} + t\mathbf{v} \mid t \in \mathbb{R}\}$ is convex. This will be very helpful because it is sometimes difficult to use the definition of convex sets to prove that a set is convex. Show that a set is convex if and only if its intersection with any line is convex.

(b) The set of solutions of a quadratic inequality $\mathcal{C} \subseteq \mathbb{R}^n$ is given by

$$\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + c \leq 0\},$$

with $\mathbf{A} \in \mathbb{S}^n$, $b \in \mathbb{R}^n$ and $c \in \mathbb{R}$. Show that $\mathcal{C}$ is a convex set if $\mathbf{A} \succcurlyeq 0$. Hint: use the result of the previous question and look at the intersection of $\mathcal{C}$ with any line of the form $\{\mathbf{u} + t\mathbf{v} \mid t \in \mathbb{R}\}$.

(c) Using MATLAB, visualize that the set $\mathcal{C}$ of question (b) is not necessarily convex if the symmetric matrix $\mathbf{A}$ is not positive semidefinite. Note: Avoid using loops for this problem.

- Choose $n = 2$.
- Generate the symmetric matrix $\mathbf{A}$, the vector $\mathbf{b}$ and the scalar c using a random generator (take all values distributed randomly between -5 and 5, use the function `rand`). Use the fact that $\mathbf{A} = \frac{1}{2}(\mathbf{B} + \mathbf{B}^T)$ is symmetric for any matrix $\mathbf{B}$.
- Recall that a symmetric matrix is positive semi-definite if all of its eigenvalues are non-negative. Obtain the eigenvalues of the random matrix $\mathbf{A}$.

- How do you classify $\mathbf{A}$ (as positive semi-definite, negative semi-definite, or neither)?
- Plot the set $\mathcal{C}$ in the range of [-20,20] for each element of $\mathbf{x}$. The functions `surf`, `combn` (useful to generate all combinations with length 2), `sign`, `diag` and `reshape` might be useful.
- Run your code for the former steps multiple times and observe that $\mathcal{C}$ is convex only when the generated $\mathbf{A}$ is positive semi-definite.

[Template: `s2p2c_template.m`]

(d) Try the matrix

$$\mathbf{A} = \begin{bmatrix} -2 & 0 \\ 0 & -3 \end{bmatrix}$$

with $\mathbf{b} = \mathbf{0}$ and $c = -\sqrt{2}$. Is the matrix $\mathbf{A}$ positive semidefinite? Is $\mathcal{C}$ convex? What if we choose $c = 100$?

[Template: `s2p2d_template.m`]

3. ***Convex functions.***
[Template: `s2p3_template.m`]

(a) Using MATLAB, check if the function

$$f(\mathbf{X}) = \operatorname{tr}(\mathbf{X}^{-1})$$

is convex for a general invertible matrix $\mathbf{X}$. Use the fact that a function f is convex if and only if it is convex when restricted to any line that intersects its domain (see section 2.3.1). In this case a line has the form $\{\mathbf{U} + t\mathbf{V} \mid t \in \mathbb{R}\}$ where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n\times n}$ and $\mathbf{U} + t\mathbf{V}$ is in the domain of f.

(b) Check the convexity of f assuming that $\mathbf{X} \in \mathbb{S}^n_+$. Note: the positive semi-definite set of matrices is a convex set. Hence, to generate a line segment in $\mathbb{S}^n_+$, randomly choose two positive semi-definite matrices (make use of the fact that any matrix with structure $\mathbf{Z}^{\mathrm{T}}\mathbf{Z}$ belongs to $\mathbb{S}^n_+$) to represent the start - and the end point of the line segment. Due to convexity, all intermediate matrices on this line are also positive semi-definite.

4. ***Avoiding loops (bonus problem)***

(a) Go back to question 3. Modify your code such that no loops are necessary. The MATLAB function `kron` might be useful. Try how `kron` behaves if one of its arguments is the identity matrix. You might also be interested in the fact that

$$\left(\begin{bmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{B} \end{bmatrix}\right)^{-1} = \begin{bmatrix} \mathbf{A}^{-1} & 0 \\ 0 & \mathbf{B}^{-1} \end{bmatrix}.$$

Make sure, that the version with loops and the version without loops result in the same outcome.

3 Linear programming I

3.1 Definition

A linear program (LP) is an optimization problem where the objective function f_0 is affine and all constraint functions f_i and h_i are affine. A linear problem has the form

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} + \mathbf{d} \\ \text{subject to} \quad & \mathbf{G}\mathbf{x} \leq \mathbf{h} \\ & \mathbf{A}\mathbf{x} = \mathbf{b}, \end{aligned} \tag{3.1}$$

where $\mathbf{G} \in \mathbb{R}^{m \times n}$ and $\mathbf{A} \in \mathbb{R}^{p \times n}$. In other words we minimize an affine function over a polyhedral set. LPs are among the first optimization problems that were solved. Nevertheless they have no analytical solution in general. One of the most famous algorithms to solve LPs is the *simplex algorithm* developed by George Dantzig in 1947.

3.2 Network Optimization

An important application of linear programming in communication systems is network optimization. Indeed, it is quite simple to model a network structure using matrices. We represent a network by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$. We assume that $|\mathcal{V}| = m$ and $|\mathcal{E}| = n$. The vertices are enumerated from 1 to m and the edges are represented by ordered pairs (i, j), i.e., there is an edge going from vertex i to vertex j. Next, we assume that there is no more than one edge from vertex i to vertex j and that there is no edge from vertex i to vertex i (i.e., no self-loops). An example network is illustrated in Figure 3.1.

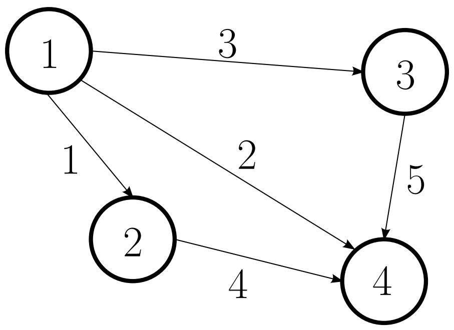


Figure 3.1: A simple network.

To mathematically represent a network like this, we can use an incidence matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ given by

$$[A]_{ij} = \begin{cases} 1 & \text{edge } j \text{ starts at vertex } i, \\ -1 & \text{edge } j \text{ ends at vertex } i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

For example the incidence matrix of the network in Figure 3.1 is given by

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & -1 & 0 & -1 & -1 \end{bmatrix}. \tag{3.3}$$

Each row of this matrix represents a vertex, while each column of the matrix represents an edge.

We can represent the flow in the network by a non-negative vector $\mathbf{x} \in \mathbb{R}^n$ where x_j is the flow through edge j. The flow is positive if it is in direction of the vertex and negative otherwise. It is possible to calculate the total flow leaving a vertex i as

$$\mathbf{a}_i\mathbf{x} = \sum_{j=1}^{n} [A]_{ij} x_j, \tag{3.4}$$

where $\mathbf{a}_i$ is the ith row of $\mathbf{A}$.

Now we know how to represent flows inside the network. We still need to model how quantities can enter and leave the network. We call $\mathbf{b} \in \mathbb{R}^m$ the supply vector (see Figure 3.2 for an illustration).

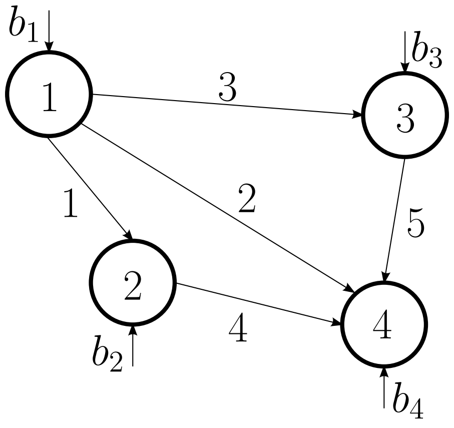


Figure 3.2: A simple network with external supply.

If b_i is negative, it means that something leaves the network, in other words a negative supply component is a demand from outside. If the supply is at the equilibrium we must have

$$\sum_{i=1}^{m} b_i = 0. \tag{3.5}$$

We say that the network is balanced if the quantities in the network stay constant. The balance equation is

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{3.6}$$

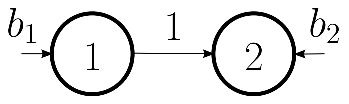


Figure 3.3: Example of balanced network.

Let's take a simple example in Figure 3.3 to wrap up these concepts. In that case, if the supply equals the demand, we have $b_1 = -b_2 = \alpha$, then we have

$$\mathbf{A} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \alpha \\ -\alpha \end{bmatrix}.$$

Using the balance equation (3.6), we find that we must (obviously) have $x_1 = \alpha$.

3.3 Problem section

1. ***A simple diet problem***

   The most famous LP is the so-called diet problem. You have to compose a diet from m different nutrients in quantities larger than or equal to $b_1, b_2, \ldots, b_m$. The nutrients are not available in isolation. They are contained in different types of food. You can choose among n different types of food. One unit of food j contains the amount a_{ij} of nutrient i and costs c_j. The problem is to choose the amount of each type of food that generates the smallest cost possible while providing a minimum amount of nutrients. The diet problem can be cast as an LP and written as

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0. \end{array} \tag{3.7}$$

   We consider a slightly different diet problem. The diet must be composed of 3 nutrients whereas the two first nutrients have to be in quantities of at least 50 units and the last nutrient has to be in the quantity of exactly 100 units. There are 5 different types of food, all of which are available in a limited amount, listed in the table below

type of food	**1**	**2**	**3**	**4**	**5**
nutrient 1	2	3	7	2	5
nutrient 2	3	6	7	3	5
nutrient 3	5	2	8	2	4
avail. amount	4	6	8	5	5
cost	5	5	10	5	7

   Solve this diet problem with CVX.

2. ***Minimum cost network flow problem***

A direct application of network optimization is the minimization of the cost of transporting resources from supply points to demand points. In the context of mobile communication we want to study a relay network composed of one transmitter, one receiver and several relays. The specific relay network we will try to optimize is given in Figure 3.4. The main question is: how can the transmitter

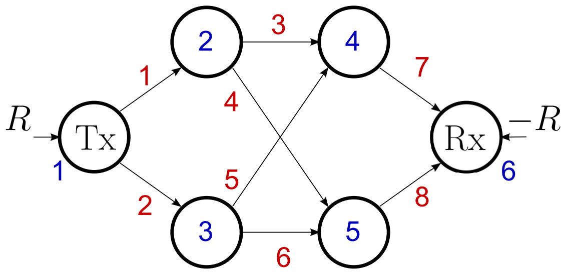


Figure 3.4: Relay network.

send at a rate $R = 10$ to the receiver using as little power as possible in the whole network? How should information be routed? Consider the relay network in Figure 3.4.

(a) Model the network with a matrix $\mathbf{A}$.

(b) What is the vector $\mathbf{b}$?

(c) We first assume a very simple model for representing the power consumed by using an edge i. We define a cost vector

$$\mathbf{c}^{\mathrm{T}} = [2\ 3\ 4\ 2\ 3\ 3\ 4\ 2\ ].$$

The power consumed in the network is equal to

$$\mathbf{c}^{\mathrm{T}}\mathbf{r},$$

with $\mathbf{r} \in \mathbb{R}^n$. The symbol r_i represents the rate transmitted on edge i. Using a linear program, find the minimum power that allows us to transmit at rate R through the network.

(d) Now assume that no edge can be used with a rate larger than 6. Using a linear program, find the minimum power that allows us to transmit at rate R through the network.

If we study wireless networks, the previous cost model for communication is highly unrealistic. In the following we assume that each link is noisy with distribution $\mathcal{N}(0, \sigma^2)$. Each link has a channel gain h_i with $i = 1, \dots, n$, which represents fading. The rate r_i, which is achieved between two nodes by using a power p_i, over the vertex i can be calculated as

$$r_i = \log_2(1 + \frac{h_i^2 p_i}{\sigma^2}). \tag{3.8}$$

So if we define $\mathbf{r} \in \mathbb{R}^n$ as the rate transmitted on each link and if we want to minimize the total power used in the network we might use the following objective function

$$\text{Total consumed power} = \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} \frac{\sigma^2}{h_i^2}(2^{r_i} - 1), \tag{3.9}$$

which can be achieved by solving (3.8) for p_i. This objective function is clearly convex and CVX can solve this problem. However, the logarithm makes the solution very slow. In this part we show that we can solve this problem, using linear programming, much faster by approximating the log functions by several affine functions.

In general, it is possible to approximate a convex function $f(\mathbf{x})$ by a piecewise-linear function given by

$$f(\mathbf{x}) \approx \max_{i=1,\ldots,m} (\mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i). \tag{3.10}$$

where m is the number of linear functions used for the approximation. It is possible to transform the minimization problem

$$\underset{\mathbf{x}}{\text{minimize}} \max_{i=1,\ldots,m} (\mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i), \tag{3.11}$$

into the following problem

$$\begin{aligned} &\underset{\mathbf{x},t}{\text{minimize}} && t \\ &\text{subject to} && \max_{i=1,\ldots,m} (\mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i) \leq t, \end{aligned} \tag{3.12}$$

where t is an additional epigraph variable. The problems (3.11) and (3.12) are equivalent. Now if the maximum of all functions $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i$ is smaller than t, then clearly each of them must be smaller than t and our minimization problem is equivalent to

$$\begin{aligned} &\underset{\mathbf{x},t}{\text{minimize}} && t \\ &\text{subject to} && \mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i \leq t, \quad i = 1, \ldots, m. \end{aligned} \tag{3.13}$$

This problem is a linear program with variables $\mathbf{x}$ and t. With this theoretical tool, we can now minimize the function in (3.9) very efficiently using a linear program.

(e)
- Initialize the random number generator of MATLAB using `randn('state', 17)`. This provides the ability of generating random realizations, while the used data and the results remain tractable for other executions.
- Generate a vector $\mathbf{h} \in \mathbb{R}^n$, at random with each entry distributed according to $\mathcal{N}(0, 1)$, containing the channel gains of each edge of the network. Furthermore assume $\sigma^2 = 1$.

- For each edge, approximate the power function (3.9) by a piecewise-linear function containing 5 affine functions. Use {0,3,6,..,15} as supporting points. Observe that we must have

$$\frac{\sigma^2}{h_i^2}(2^{r_i} - 1) \leq f(r_i, h_i),$$

so that the result we will get using this affine approximation is a lower bound on the result of the true problem. The approximation is illustrated in Fig 3.5.

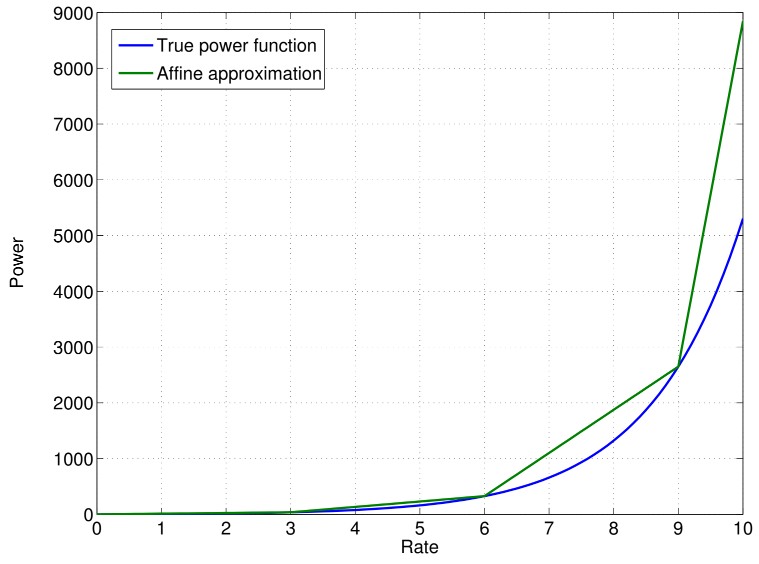


Figure 3.5: Affine approximation of the power function.

(f)
- Using a linear program, find the minimum power allowing us to transmit at rate R through the network. Hint: Use a similar formulation as in (3.12) for each edge. The total consumed power is the sum of the approximated power values for all edges.
- Launch the calculation several times with different random number generator states, do not consider any restriction on how much rate can flow through an edge. Why does the transmitter sometimes use its two output edges, and does not give all of the available power to a single path?

(g) Now take as objective function, the non-linear convex function (3.9). Solve the problem with this new function, using CVX.

(h) Compare the results obtained by the LP and by the non-linear problem. Describe the tradeoff between speed and precision using the LP?

4 Linear programming II

4.1 Chebyshev Bounds

The Chebyshev bounds are used to upper bound the probability of the event that a realization of a random variable X lies in a certain set. To evaluate this upper bound, we only need to know the expectation of certain functions of X (e.g., its expectation and its variance). This is very useful since it enables us to upper bound the probability of a physical phenomenon without needing to know its probability distribution. From a practical view, these bounds are very interesting since they can be used in linear programming.

Examples of Chebyshev bounds

Two famous examples of Chebyshev bounds are Markov's inequality for a positive random variable X ($X \in \mathbb{R}_+$) with mean μ:

$$\mathrm{P}(X \geq t) \leq \frac{\mu}{t}, \tag{4.1}$$

which for $t = 1$ simplifies to

$$\mathrm{P}(X \geq 1) \leq \mu,$$

and Chebyshev's inequality for a random variable $X \in \mathbb{R}$ with mean μ and variance σ^2

$$\mathrm{P}(|X - \mu| \geq t) \leq \frac{\sigma^2}{t^2}, \tag{4.2}$$

which for $t = 1$ simplifies to

$$\mathrm{P}(|X - \mu| \geq 1) \leq \sigma^2.$$

In both cases the inequality holds no matter how X is distributed.

Generalization

Now, consider a random variable X defined on a set $\mathcal{S} \subseteq \mathbb{R}^m$ and a subset $\mathcal{C}$ such that $\mathcal{C} \subset \mathcal{S}$. Our goal is to upper bound the probability that X lies in $\mathcal{C}$, i.e., we want to upper bound $\mathrm{P}(X \in \mathcal{C})$. To do this, we first define the function $1_\mathcal{C}$, which is the indicator function of the set $\mathcal{C}$:

$$1_\mathcal{C}(z) = \begin{cases} 1 & \text{if } z \in \mathcal{C} \\ 0 & \text{if } z \notin \mathcal{C}. \end{cases} \tag{4.3}$$

This function has the interesting property that

$$\mathbb{E}[1_\mathcal{C}(X)] = \mathrm{P}(X \in \mathcal{C}), \tag{4.4}$$

that is, the expectation of the function $1_\mathcal{C}(X)$ over all $X \in \mathcal{S}$ equals the probability that $X \in \mathcal{C}$.

Example

To understand this, let's consider an example where the sets $\mathcal{S}$ and $\mathcal{C}$ are discrete sets. Let's say $\mathcal{S}$ contains $n_\mathcal{S}$ elements. Of these $n_\mathcal{S}$ elements, $n_\mathcal{C}$ elements are also in $\mathcal{C}$ and $n_{\not\mathcal{C}}$ elements are only in $\mathcal{S}$ but not in $\mathcal{C}$, such that $n_\mathcal{S} = n_\mathcal{C} + n_{\not\mathcal{C}}$. The expectation of the function $1_\mathcal{C}(X)$ over all elements in $\mathcal{S}$ is given by

$$\begin{aligned}\mathbb{E}[1_\mathcal{C}(X)] =& \frac{1}{n_\mathcal{S}} \sum_{X\in\mathcal{S}} 1_\mathcal{C}(X) && (4.5)\\ =& \frac{1}{n_\mathcal{S}} \left(\sum_{X\in\mathcal{S}, X\in\mathcal{C}} 1 + \sum_{X\in\mathcal{S}, X\notin\mathcal{C}} 0 \right) && (4.6)\\ =& \frac{1}{n_\mathcal{S}} \sum_{i=1}^{n_\mathcal{C}} 1 && (4.7)\\ =& \frac{n_\mathcal{C}}{n_\mathcal{S}} && (4.8)\\ =& \mathrm{P}(X \in \mathcal{C}) && (4.9)\end{aligned}$$

Expected values of functions of the random variable

Recall that we don't know the probability distribution of X, but only know the expectation of certain functions f_i of X, i.e., we know

$$\mathbb{E}[f_i(X)] = a_i, \quad i = 1, \dots, n \tag{4.10}$$

with $f_i : \mathbb{R}^m \to \mathbb{R}$.

We always choose the function f_0 to be constant with value one ($f_0(z) = 1$). Since f_0 is constant, its expected value is also constant, i.e.,

$$\mathbb{E}[f_0(X)] = a_0 = 1. \tag{4.11}$$

Example

To give you a better picture of how an actual case would look, let's consider the following typical example. We want to upper bound the probability that the random variable X is in a certain set, but we don't know its probability distribution. What we do know is that X is zero-mean and that X has the variance σ^2, i.e., we know the expected values of two different functions of X, namely

$$\begin{aligned} f_1(X) &= X && \text{with} \quad \mathbb{E}[f_1(X)] = 0 \\ f_2(X) &= (X - \mathbb{E}[X])^2 \overset{\mathbb{E}[X]=0}{=} X^2 && \text{with} \quad \mathbb{E}[f_2(X)] = \sigma^2 \end{aligned} \tag{4.12}$$

We now define the function f as a linear combination of the functions f_i given by

$$f(z) = \sum_{i=0}^{n} y_i f_i(z), \tag{4.13}$$

with the weights $\mathbf{y} \in \mathbb{R}^n$. In the following, we will use the expected value of $f(X)$, which is given by

$$\mathbb{E}[f(X)] = \mathbb{E}\left[\sum_{i=0}^{n} y_i f_i(X)\right] = \sum_{i=0}^{n} y_i \mathbb{E}[f_i(X)] = \sum_{i=0}^{n} y_i a_i \tag{4.14}$$

to upper bound $\mathrm{P}(X \in \mathcal{C})$. To get the *tightest* upper bound, we will optimize over $\mathbf{y}$, i.e., choose the values for $y_i|_{i=0}^{n}$, which result in the *tightest* upper bound.

For the upper bound, let's consider the inequalities

$$f(z) \geq 1_{\mathcal{C}}(z), \quad \forall z \in \mathcal{S}. \tag{4.15}$$

If (4.15) holds, then

$$\mathbb{E}[f(X)] = \mathbf{a}^{\mathrm{T}}\mathbf{y} \geq \mathbb{E}[1_{\mathcal{C}}(X)] = \mathrm{P}(X \in \mathcal{C}). \tag{4.16}$$

In other words, if we can find a function f that fulfills (4.15), then we have found an upper bound for $\mathrm{P}(X \in \mathcal{C})$, namely $\mathbb{E}[f(X)]$. The condition in (4.15) can be represented as

$$\begin{aligned} f(z) &\geq 1, \quad \text{for } z \in \mathcal{S}, z \in \mathcal{C} \\ f(z) &\geq 0, \quad \text{for } z \in \mathcal{S}, z \notin \mathcal{C}. \end{aligned} \tag{4.17}$$

To get the best (tightest) upper bound possible, we want to find the function f that produces the smallest $\mathbb{E}[f(X)]$. This is realized by minimizing $\mathbb{E}[f(X)]$ with respect to $\mathbf{y}$. The optimization problem we need to solve to achieve this is given by

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \mathbf{a}^{\mathrm{T}}\mathbf{y} \\ \text{subject to} \quad & f(z) \geq 1, \quad \text{for } z \in \mathcal{S}, z \in \mathcal{C} \\ & f(z) \geq 0, \quad \text{for } z \in \mathcal{S}, z \notin \mathcal{C}. \end{aligned} \tag{4.18}$$

The objective of the optimization problem is the upper bound on $\mathrm{P}(X \in \mathcal{C})$, which is to be minimized, while the constraints make sure that our solution obeys (4.15). Formulating the problem in scalar notation results in

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \sum_{i=0}^{n} a_i y_i \\ \text{subject to} \quad & \sum_{i=0}^{n} y_i f_i(z) \geq 1, \quad \text{for } z \in \mathcal{S}, z \in \mathcal{C} \\ & \sum_{i=0}^{n} y_i f_i(z) \geq 0, \quad \text{for } z \in \mathcal{S}, z \notin \mathcal{C}, \end{aligned} \tag{4.19}$$

which is obviously a linear program in $\mathbf{y}$. The optimal value p^* of this problem is an upper bound of $\mathrm{P}(X \in \mathcal{C})$.

4.2 Problem section

1. ***Probability of Correct Signal Detection***

   In this section we will solve a simple signal detection problem using the Chebyshev bounds. We assume the following model. A transmitter sends a signal $\mathbf{x} \in \mathcal{P}$ to a receiver where $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\} \subseteq \mathbb{R}^n$ is the signal constellation and $\mathbf{p}_i|_{i=1}^m$ is a modulation point in the constellation. For example a 4-PAM (pulse amplitude modulation) system uses 4 possible levels, e.g., -3V, -1V, 1V, 3V. The receiver samples the signal $\mathbf{y}$ such that

   $$\mathbf{y} = \mathbf{x} + \mathbf{n}, \tag{4.20}$$

   where $\mathbf{n}$ is a noise vector. The goal at the receiver is to find the correct value of the transmitted signal $\mathbf{x}$ from the received samples $\mathbf{y}$. Generally, a minimum distance detector is used, i.e., the receiver chooses the point $\mathbf{p}_j$ that is closest to the received $\mathbf{y}$ in the Euclidean sense. Formally, the receiver chooses $\mathbf{p}_j$ such that

   $$\|\mathbf{y} - \mathbf{p}_j\|_2 < \|\mathbf{y} - \mathbf{p}_i\|_2, \quad \forall i \neq j. \tag{4.21}$$

   Note that this equation defines a Voronoi region. A correct detection occurs if the modulation point that is closest to the received signal is the actual transmitted point, i.e., $\mathbf{x} = \mathbf{p}_j$. By replacing $\mathbf{y}$ by $\mathbf{x} + \mathbf{n}$ in Equation 4.21, and then choosing $\mathbf{x} = \mathbf{p}_j$, we get the condition on $\mathbf{n}$, which ensures correct detection of $\mathbf{p}_j$

   $$\begin{array}{rcl} \|\mathbf{x} + \mathbf{n} - \mathbf{p}_j\|_2 & < & \|\mathbf{x} + \mathbf{n} - \mathbf{p}_i\|_2, \quad \forall i \neq j \\ \|\mathbf{n}\|_2 & < & \|\mathbf{p}_j + \mathbf{n} - \mathbf{p}_i\|_2 \\ \mathbf{n}^{\mathrm{T}}\mathbf{n} & < & (\mathbf{p}_j + \mathbf{n} - \mathbf{p}_i)^{\mathrm{T}}(\mathbf{p}_j + \mathbf{n} - \mathbf{p}_i) \\ 2(\mathbf{p}_i^{\mathrm{T}}\mathbf{p}_j + \mathbf{p}_i^{\mathrm{T}}\mathbf{n} - \mathbf{p}_j^{\mathrm{T}}\mathbf{n}) & < & \mathbf{p}_i^{\mathrm{T}}\mathbf{p}_i + \mathbf{p}_j^{\mathrm{T}}\mathbf{p}_j \\ 2(\mathbf{p}_i^{\mathrm{T}}\mathbf{p}_j + \mathbf{p}_i^{\mathrm{T}}\mathbf{n} - \mathbf{p}_j^{\mathrm{T}}\mathbf{n} - \mathbf{p}_j^{\mathrm{T}}\mathbf{p}_j) & < & \mathbf{p}_i^{\mathrm{T}}\mathbf{p}_i - \mathbf{p}_j^{\mathrm{T}}\mathbf{p}_j \\ 2(\mathbf{p}_i - \mathbf{p}_j)^{\mathrm{T}}(\mathbf{p}_j + \mathbf{n}) & < & \|\mathbf{p}_i\|_2^2 - \|\mathbf{p}_j\|_2^2, \quad \forall i \neq j \end{array} \tag{4.22}$$

   Let's now consider a scenario with two-dimensional signals, i.e., $n = 2$ and a situation, where we know that the noise has the mean vector $\mathbb{E}[\mathbf{n}] = \mathbf{0}_{2\times 1}$ and the covariance matrix $\mathbb{E}[\mathbf{n}\mathbf{n}^{\mathrm{T}}] = \sigma^2\mathbf{I}_{2\times 2}$ with $\sigma^2 = 1$. We further know that the transmitted signal is a 16-QAM signal with $\mathbf{x} \in \{-3, -1, 1, 3\}^2$, i.e., $\mathbf{x}$ has two entries, each of which can take a value from the set $\{-3, -1, 1, 3\}$. Finally, the noise vector $\mathbf{n}$ can only take values in $\{-2, 0, 2\}^2$.

   The question arises how we could **upper bound the probability of correct detection** for **each** symbol $\mathbf{p}_j|_{j=1}^{16}$ although we do not know the probability distribution of the noise. To achieve this we want to use linear programming and the Chebyshev bounds.

   (a) Note that the problem will be solved separately for each of the constellation points. Hence, we only deal with the statistics of the noise for each symbol. Recall that while the distribution of the noise is unknown to us, the expected value and the variance of the noise is known. Now, think about how this problem relates to the one from section 4.1 of the reading assignment. You should find the answers to the following questions:

- Which of the variables from the problem description of this exercise is the one that has the set $\mathcal{S}$ as its support?
- What does it mean if said variable is element of $\mathcal{C}$ and what does it mean if it is not? Explain that in connection to our current problem, regarding noise and symbol detection.

(b) We will need the set $\mathcal{S}$ as well as the set $\mathcal{C}_j$ corresponding to the symbols $\mathbf{p}_j|_{j=1}^{16}$. Write down the set definitions for $\mathcal{S}$ and $\mathcal{C}_j$ on paper. **Let one of the lab supervisors check your solution before progressing.**

(c) Write a MATLAB script that generates the sets $\mathcal{C}_j$ and $\{\mathcal{S} \setminus \mathcal{C}_j\}$ for all $j = 1..16$. For each set your script should generate a matrix in which each column corresponds to a point in the set. Make use of the function `combn`.

(d) From the problem statement, we can extract five (scalar) functions of the variable which is supported by the set $\mathcal{S}$, together with their expected values. Write down the functions $f_i|_{i=1}^{5}$ and their respective expected values on paper. Also write down their linear combination f. **Let one of the lab supervisors check your solution before progressing.**

(e) Write down a linear program that finds the tightest upper bound on the probability of correct detection of a symbol. It should make use of the sets from (b), the function f from (d) and the indicator function of the set $\mathcal{C}_j$. Follow the same concept as presented in (4.19).

(f) Implement the optimization problem from (e) using CVX and run it for each symbol $\mathbf{p}_j|_{j=1}^{16}$. What are the probabilities of correct detection for the symbols $\{-3, -1, 1, 3\}^2$?

		x_1			
		-3	-1	1	3
	-3				
	-1				
x_2	1				
	3				

(g) Modify your program such that it gives an upper bound on the probability of **erroneous** detection for each point (instead of **correct** detection).

				x_1	
		-3	-1	1	3
	-3				
	-1				
x_2	1				
	3				

(h) The resulting values of the last two parts of the problem are the approximations (upper-bounds) regarding the correct/incorrect detection probabilities. Can you think of a simple way to evaluate their accuracy / tightness ?

(i) **[bonus problem]** Write a Monte Carlo simulation and estimate the real probabilities of correct / erroneous detection. Compare these results to the computed bounds from (f) and (g).

5 Quadratic programming

5.1 Introduction

We have already encountered linear programs (LPs). They have been shown to be useful in modeling various problems, e.g., network flow, diet, or detection problems. Although LPs provide a fairly general structure, they may not reflect all kinds of real-world problems we are interested in. For example, consider the following LP

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{5.1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. In this diet problem, the vector $\mathbf{c}$ corresponds to a fixed cost function. However, costs may vary with time. We will model this variation by a *random variable* mean value $\mathrm{E}[\mathbf{c}] = \bar{\mathbf{c}}$ and covariance matrix $\mathbf{\Sigma} = \mathrm{E}[(\mathbf{c}-\bar{\mathbf{c}})(\mathbf{c}-\bar{\mathbf{c}})^{\mathrm{T}}]$.

How can we incorporate this information into our model? We could replace the objective $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ by the *expected cost* $\mathrm{E}[\mathbf{c}^{\mathrm{T}}\mathbf{x}] = \bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x}$. The modified linear program now reads as

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{5.2}$$

This approach alone will not do the job. There are at least two reasons for this. First of all, we should never skip additional information, which comes without additional cost, that is, the covariance matrix $\mathbf{\Sigma}$. Second, let us assume we are solving the optimization problem (5.2) over many realizations of $\mathbf{c}$. If we are lucky, some solutions result in a small cost. Due to the randomness of $\mathbf{c}$, we might also have some bad luck and get several very costly solutions. In other words, the *cost variance* $\mathrm{Var}[\mathbf{c}^{\mathrm{T}}\mathbf{x}]$ might be too high.

One way to address this issue is to minimize a linear combination of the expected cost and the cost variance, i.e.,

$$\mathrm{E}[\mathbf{c}^{\mathrm{T}}\mathbf{x}] + \gamma \mathrm{Var}[\mathbf{c}^{\mathrm{T}}\mathbf{x}], \tag{5.3}$$

where $\gamma \geq 0$. This expression is called the *risk-sensitive cost*, since it offers a trade-off between small expected cost and small cost variance. The *risk-aversion parameter* γ controls this trade-off. Higher values correspond to higher expected cost and lower cost variance and vice versa. After a few calculations, the resulting optimization problem is given by

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x} + \gamma \mathbf{x}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{5.4}$$

Due to the quadratic form $\mathbf{x}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{x}$ in the objective, this optimization problem is not a linear program anymore. It belongs to a class of optimization problems called *quadratic programs*. How to identify, model and solve quadratic problems will be the main focus of this session.

5.2 Quadratic programs

Consider the following optimization problem

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & (1/2)\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} + \mathbf{q}^{\mathrm{T}}\mathbf{x} + r \\ \text{subject to} & \mathbf{G}\mathbf{x} \leq \mathbf{h} \\ & \mathbf{A}\mathbf{x} = \mathbf{b}, \end{array} \tag{5.5}$$

where $\mathbf{P} \in \mathbb{S}^n_+$, $\mathbf{G} \in \mathbb{R}^{m\times n}$, and $\mathbf{A} \in \mathbb{R}^{p\times n}$. An optimization problem of this particular form is called *convex quadratic program* or in short *quadratic program* (QP). In a quadratic program, we minimize a convex quadratic function (the matrix $\mathbf{P}$ is positive semidefinite) over a polyhedron (the constraint functions are affine).

Taking $\mathbf{P} = \mathbf{0}$, we can directly see that quadratic programs include linear programs as a special case. In general, linear programs don't offer analytical solutions. Interestingly, this is not true for all quadratic programs. For example, the analytical solution to the famous least-squares problem comes up in so many different fields that most people actually don't recognize it as the solution of a quadratic program.

5.2.1 Relation to least-squares problems

We will study least-squares problems in greater detail in later sessions. That said, it is noteworthy to point out the connection between QPs and least-squares problems. An unconstrained *least-squares approximation* problem is given by

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2, \tag{5.6}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is full rank $(m \geq n)$. Recall that the squared ℓ_2-norm of a real vector $\mathbf{x}$ is given by $\|\mathbf{x}\|_2^2 = \mathbf{x}^{\mathrm{T}}\mathbf{x} = \sum_{i=1}^n x_i^2$. Using this, we can rewrite the objective of (5.6) as

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = \mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} - 2\mathbf{b}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{b}.$$

Therefore, an unconstrained least-squares problem is essentially an unconstrained quadratic program. In this simple form, the very well-known (and often overrated) analytical solution is $\mathbf{x}^* = \mathbf{A}^\dagger\mathbf{b}$, where $\mathbf{A}^\dagger = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}$ is the pseudo-inverse of $\mathbf{A}$. Even when adding affine constraints to the unconstrained least-squares problem, the problem will remain a constrained QP. However, in this case there might be no analytical solution anymore.

Example: Regression or curve fitting

Suppose we are given a bunch of m data points $(t_i, f(t_i))$. Those points are illustrated through blue markers in Figure 5.1. We could think of many examples in practice about the origin of this data, e.g., from a ballistic trajectory, an opinion survey, or an investment strategy payoff. Our job is to find a model, which explains the observed data and is able to predict future outcomes. We will restrict ourselves to a linear combination of functions given by

$$f(t) = x_1 f_1(t) + x_2 f_2(t) + \cdots + x_n f_n(t),$$

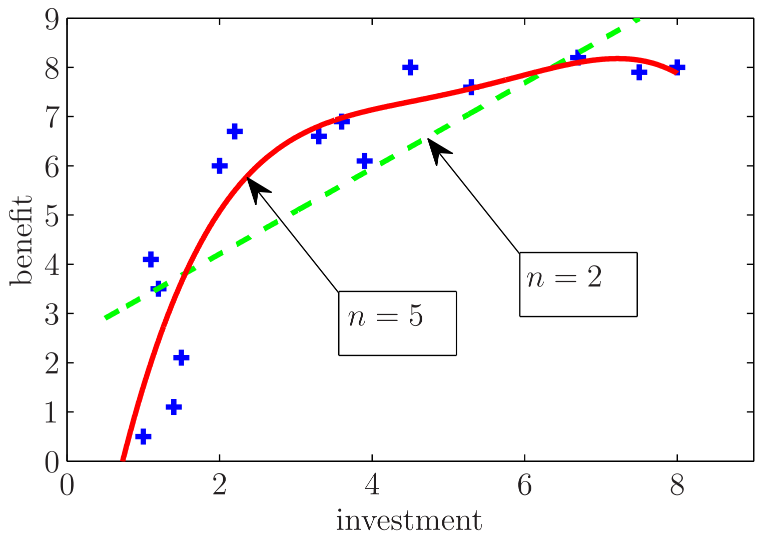


Figure 5.1: Least-squares fitting.

where $x_1, \ldots, x_n$ are called the unknown parameters, hidden variables or regression coefficients. Linearity means in this case that we have a linear combination of possibly non-linear functions. Collecting the data points $y_i = f(t_i)$ for all m measurements, we end up with a set of linear equations

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} f_1(t_1) & \cdots & f_n(t_1) \\ f_1(t_2) & \cdots & f_n(t_2) \\ \vdots & \ddots & \vdots \\ f_1(t_m) & \cdots & f_n(t_m) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

or equivalently $\mathbf{y} = \mathbf{Ax}$. Assume we want to use a simple polynomial model, that is

$$f(t) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1}.$$

Our data matrix $\mathbf{A}$ then becomes a so-called *Vandermonde matrix* and reads as

$$\mathbf{A} = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_m & t_m^2 & \cdots & t_m^{n-1} \end{bmatrix},$$

where $\mathbf{A}$ has full rank if $m \geq n$ and all t_i are distinct. In an estimation or regression problem, we should have more measurements than hidden variables[9], that is, $m > n$. Therefore, we can't invert the matrix $\mathbf{A}$ exactly. In order to get an approximate

[9]In later sessions, we will consider problems in control theory and compressive sampling. The requirement $m > n$ will then change radically!

solution, we can use a least-squares approximation. That is to say, we have to solve the optimization problem

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2. \tag{5.7}$$

In Fig.5.1 we can observe two different curve fittings ($n = 2$ and $n = 5$). Note that the fifth order polynomial provides a better fit than the first order polynomial. Is it therefore always good to increase the value of n?

Example: Vertical seismic profiling

Consider the vertical seismic profiling problem depicted in Figure 2(a) ([1]). Depending on the layer of earth, we are interested in some properties of the material surrounding the borehole, e.g., density, amount of water etc. The wave propagation speed in a specific medium is directly linked with the corresponding material properties. Therefore, we would like to create a vertical velocity profile using a source that creates sound waves. The observed wavefront travel time $t(z)$ at vertical position z can be stated as

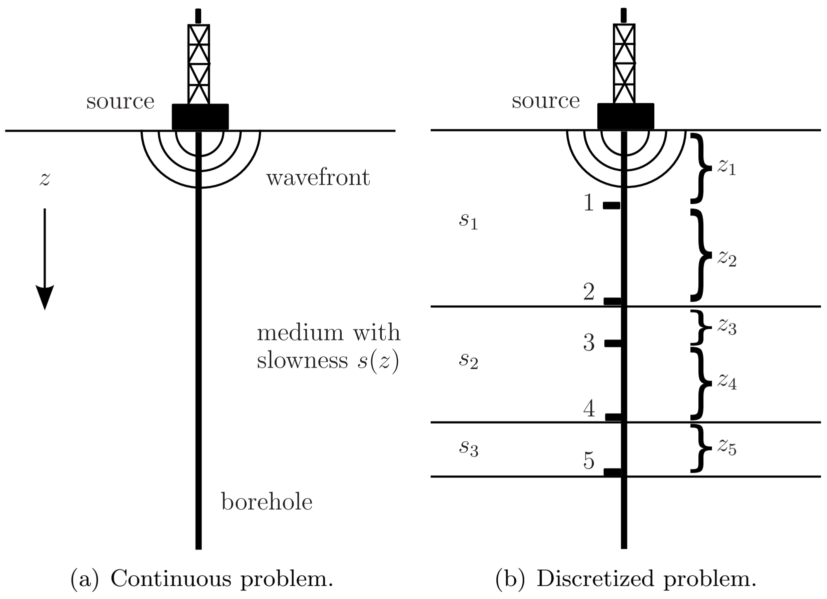


Figure 5.2: The vertical seismic profiling problem.

$$t(z) = \int_0^z s(r)dr + w(z) = \int_0^\infty u(z-r)s(r)dr + w(z), \tag{5.8}$$

where $s(r)$ is the vertical slowness and $w(z)$ is measurement noise. The step function $u(y)$ is then defined by

$$u(y) = \begin{cases} 1, & \text{if } y \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

At first glance, we can solve the above problem by simple differentiation of time over distance $t'(z) = s(z)$. This approach, however, does not work in practice due to the nature of the additive noise. As we know, a noise signal with high variations leads to a high magnitude of differentiation, which decreases the estimation accuracy (the opposite argumentation holds for averaging and integration operators which attenuate the effect of additive noise). As an alternative approach to estimate the vertical velocity, we place $m = 5$ sensors at different levels in the borehole (Figure 2(b)). The vertical distance between the sensor j and $j-1$ (if existent) is denoted as z_j. This leads to a discrete approximation of the $s_i, ; 1 \leq i \leq 3$ (see Figure 2(b)) based on the measured time instances $t_i, ; 1 \leq i \leq m$. The linear dependence of the measured time instances with the slowness and distance values can be then formulated as

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{bmatrix} = \begin{bmatrix} z_1 & 0 & 0 \\ z_1 + z_2 & 0 & 0 \\ z_1 + z_2 & z_3 & 0 \\ z_1 + z_2 & z_3 + z_4 & 0 \\ z_1 + z_2 & z_3 + z_4 & z_5 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix},$$

or equivalently $\mathbf{t} = \mathbf{Z}\mathbf{s} + \mathbf{w}$. Our goal is to estimate the slowness parameter vector $\mathbf{s}$ from a number of noisy observations $\mathbf{t}$. In least-squares estimation, this can be done by minimizing the residual error term $\mathbf{w}$ with respect to the ℓ_2-norm, that is, solving the convex quadratic optimization problem

$$\underset{\mathbf{s}}{\text{minimize}} \quad \|\mathbf{t} - \mathbf{Z}\mathbf{s}\|_2^2. \tag{5.9}$$

One last remark on this example: Least-squares problems are present in many different fields. It is fair to say that the possibility to solve these problems efficiently has triggered whole research areas. This research has resulted in very efficient and specialized solutions, e.g., sphere decoders, Kalman filters, and linear prediction algorithms. It is therefore important to point out that in this lab, we only study least-squares problems superficially. We focus on small models. For advanced concepts, it is mandatory to study subject specific literature.

5.2.2 Solving quadratic programs via CVX

Let us have a look at some example code below. Solving a quadratic program via CVX obeys the same basic rules as we have already encountered in linear programs. We have to define the optimization variable $\mathbf{x}$ of proper dimension n. After that, we have to plug in the objective and define constraints. In this particular case, only inequality constraints are included.

```
% problem statement
m = 50; n = 60; P = randn(n,n); P = P'*P;
q = randn(n,1); G = randn(m,n); h = randn(m,1); r = randn(1);
% CVX solution
cvx_begin
```

```
    variable x(n)
    minimize( x'*P*x + q'*x)
    subject to
        G*x <= h;
cvx_end
```

Note that we can simply omit the constant factor r, which is independent of the optimization variable. However, for calculating the actual optimal value

$$p^* = \underset{\mathbf{x}}{\operatorname{argmin}}\{\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} + \mathbf{q}^{\mathrm{T}}\mathbf{x} + r\} = \underset{\mathbf{x}}{\operatorname{argmin}}\{\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} + \mathbf{q}^{\mathrm{T}}\mathbf{x}\} + r$$

we add the scalar value r again. We can also conveniently solve least-squares problems (with constraints) with CVX using the expression `norm(A * x - b, 2)`.

Two final comments are in order. The first is that there is no ambiguity about the mathematical definition of the ℓ_2-norm. However, this might change when it comes to numerical calculation. For example, MATLAB already has its own predefined `norm`-function. When working within the CVX boundaries (between `cvx_begin` and `cvx_end`), CVX uses an extended version of the standard norm function. The second is that CVX will automatically transform a least-squares problem with equality and inequality constraints into some solver-specific form. The point here is that the user does not have to know about such things when using CVX as a high level modeling system.

5.3 Problem section

1. ***Diet problem with random cost.***

   You are given a fixed data set for this problem, including the matrices $\mathbf{G}$ and $\mathbf{\Sigma}$ and the vectors $\mathbf{h}$ and $\bar{\mathbf{c}}$.

   (a) Consider the diet problem with random cost as stated in (5.2)

   $$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & \mathbf{G}\mathbf{x} \geq \mathbf{h}, \mathbf{x} \geq \mathbf{0}. \end{array}$$

   Solve this optimization problem. After that, use the following code to create 1000 different realizations of the random cost vector:

   ```
   c = c_bar + sqrtm(Sigma) * randn(n, 1);
   ```

   For each realization $\mathbf{c}_k$, $1 \leq k \leq 1000$, store the value of the actual cost $\mathbf{c}_k^{\mathrm{T}}\mathbf{x}^*$ in a vector cost_LP (one vector of dimension 1000×1).

   (b) Now consider the risk-sensitive diet problem (5.4). Show that the objective can be expressed as

   $$\mathrm{E}[\mathbf{c}^{\mathrm{T}}\mathbf{x}] + \gamma \mathrm{Var}[\mathbf{c}^{\mathrm{T}}\mathbf{x}] = \bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x} + \gamma \mathbf{x}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{x}.$$

   Hint: $\mathrm{Var}[\mathbf{c}^{\mathrm{T}}\mathbf{x}] = \mathrm{E}[(\mathbf{c}^{\mathrm{T}}\mathbf{x} - \bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x})^2]$.

   (c) What is the meaning of $\gamma \geq 0$? Can you give an intuitive interpretation for $\gamma < 0$? How is convexity related to γ?

   (d) Solve the risk-sensitive optimization problem

   $$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x} + \gamma \mathbf{x}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{x} \\ \text{subject to} & \mathbf{G}\mathbf{x} \geq \mathbf{h}, \mathbf{x} \geq \mathbf{0}. \end{array}$$

   Take the following three different values of γ: $\gamma_1 = 0.1$, $\gamma_1 = 0.85$ and $\gamma_3 = 5$. Use the same 1000 realizations of the random cost vector as in (a). Store the values of the actual costs in a matrix cost_QP (dimension 1000×3).

   (e) Compare the results of both optimization problems (a) and (d) in terms of mean and variance of the actual cost. You may want use a nice presentation (e.g., actual cost with highlighted mean and variance) to make the difference clear to a possibly broader audience.

   Hint: The following MATLAB functions might be useful:
   `mean`, `var`

2. ***FIR filter design with low coefficient variation.*** **[This is a bonus problem. Hence, it is not required for passing the LAB]**

We consider the design of an FIR bandpass filter (Fig. 5.3). The Fourier transform of the filter impulse response is given by

$$H(\omega) = \sum_{n=0}^{N-1} h_n e^{-j\omega n},$$

where N is the filter length. Our goal is to find the optimal filter coefficients $h_0, \dots, h_{N-1}$, such that the deviation from a desired frequency response $H_{\text{des}}(\omega)$ is minimized with respect to the ℓ_2-norm. Additionally, we would like to achieve a low coefficient variation, which will be described in detail later. The general structure of the filter design problem we want to solve can be expressed as[10]

$$\begin{array}{ll} \underset{h_0,\dots,h_{N-1}}{\text{minimize}} & \|H(\omega) - H_{\text{des}}(\omega)\|_2^2 \\ \text{subject to} & \text{low coefficient variation.} \end{array}$$

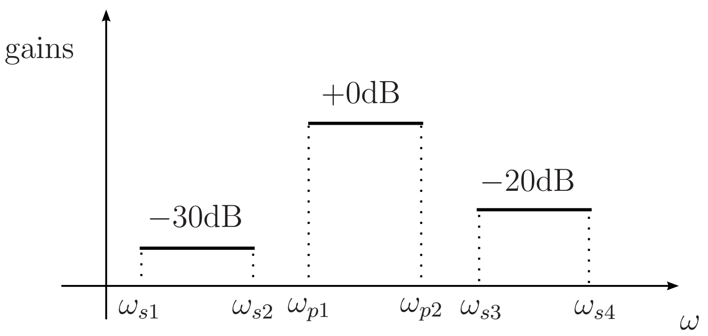


Figure 5.3: Filter design specifications in the frequency domain.

(a) We will now construct the objective function. In order to solve the optimization problem, we have to discretize the frequency domain at K frequency points

$$H(\omega_k), H_{\text{des}}(\omega_k) \quad \text{given at} \quad \omega_0, \dots, \omega_{K-1}.$$

Express the filter design objective as a least-squares optimization problem. You should end up with the general formulation $\|\mathbf{Ax} - \mathbf{b}\|_2^2$. Identify the relation between $\mathbf{A}, \mathbf{x}, \mathbf{b}$, and the quantities and variables in the filter design problem.

Hint: Try to find a convenient vector notation for $H(\omega_0), \dots, H(\omega_{K-1})$ first. Furthermore, you can assume that the values of $H_{\text{des}}(\omega_k)$ are known problem data.

[10]Note that the objective contains a norm in an infinite-dimensional space. However, after discretization this norm will result in a ℓ_2 vector norm.

(b) In order to attain a low coefficient variation, we will introduce the constraints

$$|h_n - h_{n-1}| \leq c, \quad n = 1, \ldots, N-1,$$

where c is a given maximum variation. This constraint basically means that adjacent filter coefficients may not vary by more than the value of c. Express these single constraints in terms of one matrix and two vectors.

(c) We will now solve the filter design problem numerically. Take the filter length as $N=100$ and the phase delay as $D=50$. The three frequency bands (Fig.5.3) cover the following range of frequencies

$$(\omega_{s1}, \omega_{s2}) = (0, 0.3), \qquad (\omega_{p1}, \omega_{p2}) = (0.35, 0.5), \qquad (\omega_{s3}, \omega_{s4}) = (0.55, 1).$$

In order to solve the problem, you can use a predefined function, which is called `[A,b] = DesiredFrequencyResponse(N,D,f_range,gains)`. This function will help you to generate the problem specific data, that is, the matrix **A** and the vector **b**. Assume the coefficient variation is given by $c = 0.18$. use the function `norm(.)` from cvx toolbox.

(d) Plot the frequency and the phase response of the optimally designed FIR filter using the MATLAB function `freqz(h)`. In addition, plot the filter impulse response in the time domain using the function `stem(h)`. Does your filter design satisfy every prescribed design specifications? If not, why are some design specifications only fulfilled approximately?

6 Duality

Duality theory is very important in optimization. In particular, it makes it possible to efficiently find a lower bound for optimization problems (which don't necessarily have to be convex) and to certify that a feasible point is optimal.

6.1 Lagrangian and Lagrange dual function

We consider the optimization problem

$$\begin{array}{lll} \underset{\mathbf{x}}{\text{minimize}} & f_0(\mathbf{x}) & \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, & i = 1, \ldots, m \\ & h_j(\mathbf{x}) = 0, & j = 1, \ldots, p, \end{array} \tag{6.1}$$

with $\mathbf{x} \in \mathbb{R}^n$ (domain of the involved functions) as defined in section 2. We call this problem the *primal problem.*

The *Lagrangian* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ (domain of Lagrangian) is a function given by

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^{p} \nu_i h_i(\mathbf{x}). \tag{6.2}$$

The domain of L is $\mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$ with $\mathcal{D} = \left(\bigcap_{i=0}^{m} \text{dom } f_i \right) \cap \left(\bigcap_{j=1}^{p} \text{dom } h_j \right)$. Note that the domain $\mathcal{D}$ is not the feasible set of (6.1). We say that

- λ_i is the *Lagrange multiplier* associated with the i-th inequality constraint $f_i(\mathbf{x}) \leq 0$,
- ν_i is the *Lagrange multiplier* associated with the i-th equality constraint $h_i(\mathbf{x}) = 0$.

The vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ are called *dual variables* or *Lagrange multiplier vectors* associated with the primal problem (6.1). We can interpret the meaning of the dual variables as follows. In the problem (6.1), the constraints f_i and h_i cannot be violated. Indeed, if a point $\tilde{\mathbf{x}}$ violates one constraint then this point is not feasible and we reject it. In a sense, violating the constraints produces an infinite amount of irritation. If a constraint is violated we are infinitely unhappy.

What happens with the Lagrangian, is that we pull together the objective function as well as the constraints multiplied with the dual variables in a single function L. Using the Lagrangian, if a point violates at least one constraint, we have $\boldsymbol{\lambda} \neq \mathbf{0}$ and/or $\boldsymbol{\nu} \neq \mathbf{0}$ and L is not equal to $f_0(\mathbf{x})$. The difference between L and $f_0(\mathbf{x})$ is proportional to $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$. In other words, the irritation that is produced by violating the constraints is growing linearly with $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$. So in a way, L is a version of the problem (6.1) where the constraints are soft, i.e., if the constraints are only slightly violated we are only a little bit annoyed.

The *Lagrange dual function* $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ is defined as the minimum value of L over $\mathbf{x}$ with $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\nu} \in \mathbb{R}^p$ given by

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\mathbf{x} \in \mathcal{D}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}). \tag{6.3}$$

If the function L is unbounded below then $g = -\infty$. The function g is the pointwise infimum of a family of affine (i.e., concave) functions. As seen is Section 2, such a function is always concave in $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$. This is extremely interesting and useful, given any optimization problem (convex or not), the function g is always concave. We will see soon why this fact is so important. For now you could ask why we would introduce the Lagrange dual function - what is it good for? Here comes the main point about g: $\forall \boldsymbol{\lambda} \geq \mathbf{0}$ it holds that

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^\star. \tag{6.4}$$

In other words $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ is a lower bound on the optimal value of the problem (6.1). So even if we face a nonconvex problem, which we cannot solve efficiently, we can calculate a lower bound on the optimal value of this problem by calculating $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ for any $\boldsymbol{\nu}$ and $\boldsymbol{\lambda} \geq \mathbf{0}$. Let us see why this is true. Assume we have a feasible point $\tilde{\mathbf{x}}$ and $\boldsymbol{\lambda} \geq \mathbf{0}$, then

$$\sum_{i=1}^{m} \underbrace{\lambda_i}_{\geq 0 \ (a)} \underbrace{f_i(\tilde{\mathbf{x}})}_{\leq 0 \ (b)} + \sum_{i=1}^{p} \nu_i \underbrace{h_i(\tilde{\mathbf{x}})}_{=0 \ (c)} \leq 0,$$

(a) : $\boldsymbol{\lambda} \geq \mathbf{0}$,
(b) and (c) : $\tilde{\mathbf{x}}$ is a feasible point.

Therefore $\sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^{p} \nu_i h_i(\mathbf{x})$ is nonpositive, it follows directly that

$$L(\tilde{\mathbf{x}}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f_0(\tilde{\mathbf{x}}).$$

Finally, we have

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \underset{(a)}{\leq} L(\tilde{\mathbf{x}}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f_0(\tilde{\mathbf{x}})$$

(a) : $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ is the infimum of $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$ over $\mathbf{x}$ thorough the whole domain, while $\tilde{\mathbf{x}}$ is necessarily located in the feasible set of our primal optimization problem.

Now since this inequality holds for any feasible point, it also holds for the solution $\mathbf{x}^\star$ and we have

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f_0(\mathbf{x}^\star) = p^\star.$$

Example 1: Least-norm problem

We consider the convex optimization problem

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \|\mathbf{x}\|_2^2 = \mathbf{x}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b}, \end{array} \tag{6.5}$$

with $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $m \leq n$. The Lagrangian of the problem (6.5) is

$$L(\mathbf{x}, \boldsymbol{\nu}) = \mathbf{x}^{\mathrm{T}}\mathbf{x} + \boldsymbol{\nu}^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}).$$

The Lagrange dual function is

$$g(\boldsymbol{\nu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\nu}) = \inf_{\mathbf{x}} \{\mathbf{x}^{\mathrm{T}}\mathbf{x} + \boldsymbol{\nu}^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b})\}.$$

To find the infimum of $L(\mathbf{x}, \boldsymbol{\nu})$ we calculate its gradient with respect to $\mathbf{x}$ and set it to zero

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\nu}) = 2\mathbf{x} + \mathbf{A}^{\mathrm{T}}\boldsymbol{\nu} = \mathbf{0}.$$

It follows that $L(\mathbf{x}, \boldsymbol{\nu})$ reaches its infimum for

$$\mathbf{x} = -\frac{1}{2}\mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}.$$

By plugging this value of $\mathbf{x}$ in $L(\mathbf{x}, \boldsymbol{\nu})$ we find

$$g(\boldsymbol{\nu}) = -\frac{1}{4}\boldsymbol{\nu}^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\boldsymbol{\nu} - \mathbf{b}^{\mathrm{T}}\boldsymbol{\nu}.$$

Now from (6.4) we know that for all values of $\boldsymbol{\nu}$ it holds that

$$p^\star \geq -\frac{1}{4}\boldsymbol{\nu}^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\boldsymbol{\nu} - \mathbf{b}^{\mathrm{T}}\boldsymbol{\nu}.$$

To illustrate this fact we generate a matrix $\mathbf{A}$ and a vector $\mathbf{b}$ with $m = 1$ at random. We plot $p^\star$ and $g(\boldsymbol{\nu})$ in Figure 6.1. We clearly see that $p^\star \geq g(\boldsymbol{\nu})$ for all values of $\boldsymbol{\nu}$. Interestingly for $\boldsymbol{\nu} = -0.5$ we have $p^\star = g(\boldsymbol{\nu})$, which means that the lower bound might be tight. We will come back to this in the following.

Example 2: Two-way partitioning problem

In the former example, the optimization problem was convex and we could simply have solved it using CVX. In this example, we consider a problem that is not convex to show the power of the Lagrangian concept. We consider the following problem

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{x} \\ \text{subject to} & x_i = \{-1, 1\},\ i = 1, \ldots, n, \end{array} \tag{6.6}$$

with $\mathbf{W} \in \mathbb{R}^{n \times n}$. This problem is not convex for two reasons, first $\mathbf{W}$ is not necessarily positive semidefinite and second the integer constraint $x_i = \{-1, 1\}$ is by no means convex. This problem is called *two-way partitioning.* Indeed, assume we have to partition n persons into two disjoint sets. The variable x_i takes the value -1 if you put the person i in the first set and $x_i = 1$ if you put said person in the other set.

The entries W_{ij} of the matrix $\mathbf{W}$ tell you how much person i would dislike to be in the same set with person j. Evidently, a negative value for W_{ij} means that the two persons

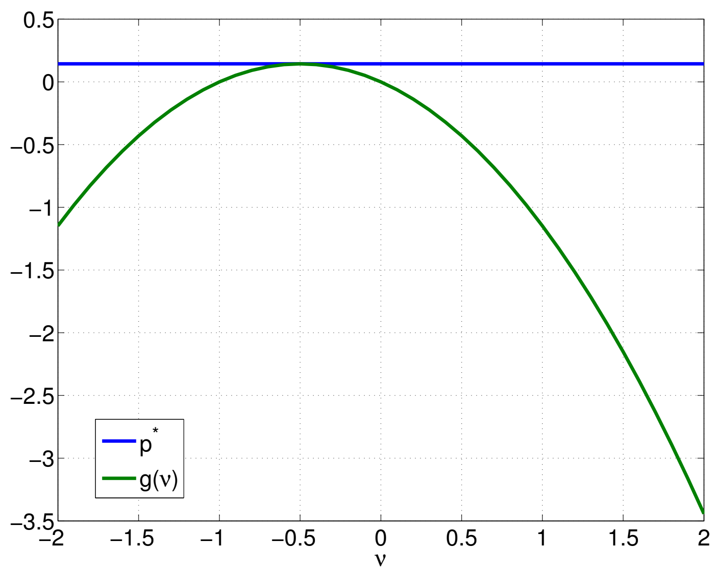


Figure 6.1: $p^\star$ and $g(\boldsymbol{\nu})$

appreciate each other. Your goal is to distribute the people into two groups such that a minimum total amount of displeasure is caused. This problem can be reformulated as

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{x} \\ \text{subject to} & x_i^2 = 1,\ i = 1, \ldots, n. \end{array} \tag{6.7}$$

The Lagrangian of the problem (6.7) is

$$\begin{array}{rcl} L(\mathbf{x}, \boldsymbol{\nu}) & = & \mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{x} + \sum_{i=1}^{n} \nu_i (x_i^2 - 1) \\ & = & \mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{x} + \mathbf{x}^{\mathrm{T}}\mathrm{diag}(\boldsymbol{\nu})\mathbf{x} - \mathbf{1}^{\mathrm{T}}\boldsymbol{\nu} \\ L(\mathbf{x}, \boldsymbol{\nu}) & = & \mathbf{x}^{\mathrm{T}}(\mathbf{W} + \mathrm{diag}(\boldsymbol{\nu}))\mathbf{x} - \mathbf{1}^{\mathrm{T}}\boldsymbol{\nu} \end{array}$$

The Lagrange dual function is

$$g(\boldsymbol{\nu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\nu}) = \inf_{\mathbf{x}} \{\mathbf{x}^{\mathrm{T}}(\mathbf{W} + \mathrm{diag}(\boldsymbol{\nu}))\mathbf{x} - \mathbf{1}^{\mathrm{T}}\boldsymbol{\nu}\}.$$

To find the infimum of $L(\mathbf{x}, \boldsymbol{\nu})$ observe the following facts

- if $\mathbf{x}^{\mathrm{T}}(\mathbf{W} + \mathrm{diag}(\boldsymbol{\nu}))\mathbf{x}$ can be negative then $g(\boldsymbol{\nu})$ is unbounded below,
- if $\mathbf{x}^{\mathrm{T}}(\mathbf{W} + \mathrm{diag}(\boldsymbol{\nu}))\mathbf{x}$ is always positive, i.e., $\mathbf{W} + \mathrm{diag}(\boldsymbol{\nu}) \succcurlyeq 0$, then the infimum is achieved when $\mathbf{x}^{\mathrm{T}}(\mathbf{W} + \mathrm{diag}(\boldsymbol{\nu}))\mathbf{x} = 0$ and $g(\boldsymbol{\nu}) = -\mathbf{1}^{\mathrm{T}}\boldsymbol{\nu}$.

So the Lagrange dual function $g(\boldsymbol{\nu})$ is given by

$$g(\boldsymbol{\nu}) = \begin{cases} -\mathbf{1}^{\mathrm{T}}\boldsymbol{\nu} & \text{if } \mathbf{W} + \mathrm{diag}(\boldsymbol{\nu}) \succcurlyeq 0 \\ -\infty & \text{otherwise.} \end{cases}$$

Again we know for sure that

$$p^\star \geq -\mathbf{1}^\mathrm{T}\boldsymbol{\nu} \qquad \text{if } \mathbf{W} + \operatorname{diag}(\boldsymbol{\nu}) \succcurlyeq 0.$$

For example, let us take $\boldsymbol{\nu} = -\lambda_{\min}(\mathbf{W})\mathbf{1}$. In that case $\mathbf{W}+\operatorname{diag}(\boldsymbol{\nu}) = \mathbf{W}-\lambda_{\min}(\mathbf{W})\mathbf{I}_n \succcurlyeq 0$. Therefore, in that case we find the following lower bound for $p^\star$

$$p^\star \geq -n\lambda_{\min}(\mathbf{W}).$$

We see that depending on the way we choose $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$, we will have a different lower bound on $p^\star$. In the next section we will explain how to choose the best lower bound in a systematic way.

6.2 Lagrange dual problem

Now that we know how to calculate $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$, we want to find $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ that maximize $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$, i.e., our lower bound on $p^\star$. The optimization problem we want to solve is

$$\begin{array}{ll} \underset{\boldsymbol{\lambda},\boldsymbol{\nu}}{\text{maximize}} & g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\ \text{subject to} & \boldsymbol{\lambda} \geq 0. \end{array} \tag{6.8}$$

This problem is called the *Lagrange dual problem* associated with the *primal problem* (6.1). We say that $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ are *dual feasible* if for $\boldsymbol{\lambda} \geq 0$, we have $g(\boldsymbol{\lambda}, \boldsymbol{\nu}) > -\infty$. Similarly we say that $(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ are *dual optimal* if for all $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ it holds $g(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star) \geq g(\boldsymbol{\lambda}, \boldsymbol{\nu})$. An important point to see is that the problem (6.8) is always a convex optimization problem, indeed, as we have seen already, $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ is a concave function and $\boldsymbol{\lambda} \geq 0$ is an affine constraint. In other words, in most cases, we can solve the dual problem efficiently.

Example: Linear program

Let us consider the following LP

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{c}^\mathrm{T}\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{array} \tag{6.9}$$

with $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$. The Lagrangian of the problem (6.9) is

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = \mathbf{c}^\mathrm{T}\mathbf{x} - \boldsymbol{\lambda}^\mathrm{T}\mathbf{x} + \boldsymbol{\nu}^\mathrm{T}(\mathbf{A}\mathbf{x} - \mathbf{b}) = -\boldsymbol{\nu}^\mathrm{T}\mathbf{b} + (\mathbf{A}^\mathrm{T}\boldsymbol{\nu} + \mathbf{c} - \boldsymbol{\lambda})^\mathrm{T}\mathbf{x}.$$

The Lagrange dual function is

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = -\boldsymbol{\nu}^\mathrm{T}\mathbf{b} + \inf_{\mathbf{x}}\{(\mathbf{A}^\mathrm{T}\boldsymbol{\nu} + \mathbf{c} - \boldsymbol{\lambda})^\mathrm{T}\mathbf{x}\}.$$

To evaluate the infimum, observe that if $\mathbf{A}^\mathrm{T}\boldsymbol{\nu} + \mathbf{c} - \boldsymbol{\lambda} \neq \mathbf{0}$ then the infimum is $-\infty$. If $\mathbf{A}^\mathrm{T}\boldsymbol{\nu} + \mathbf{c} - \boldsymbol{\lambda} = \mathbf{0}$ then $g(\boldsymbol{\nu}) = -\boldsymbol{\nu}^\mathrm{T}\mathbf{b}$. We can write $g(\boldsymbol{\nu})$ as

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \begin{cases} -\boldsymbol{\nu}^\mathrm{T}\mathbf{b} & \text{if } \mathbf{A}^\mathrm{T}\boldsymbol{\nu} + \mathbf{c} - \boldsymbol{\lambda} = \mathbf{0} \\ -\infty & \text{otherwise.} \end{cases}$$

The Lagrange dual problem associated with (6.9) is

$$\begin{array}{ll} \underset{\boldsymbol{\lambda},\boldsymbol{\nu}}{\text{maximize}} & g(\boldsymbol{\lambda},\boldsymbol{\nu}) = \begin{cases} -\boldsymbol{\nu}^{\mathrm{T}}\mathbf{b} & \text{if } \mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}+\mathbf{c}-\boldsymbol{\lambda}=\mathbf{0} \\ -\infty & \text{otherwise.} \end{cases} \\ \text{subject to} & \boldsymbol{\lambda} \geq 0. \end{array}$$

Since the lower bound $-\infty$ is not very interesting, we shall only try to maximize $g(\boldsymbol{\lambda},\boldsymbol{\nu})$ for the case that $\mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}+\mathbf{c}-\boldsymbol{\lambda}=\mathbf{0}$, i.e., we incorporate $\mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}+\mathbf{c}-\boldsymbol{\lambda}=\mathbf{0}$ as constraint in the problem

$$\begin{array}{ll} \underset{\boldsymbol{\lambda},\boldsymbol{\nu}}{\text{maximize}} & -\boldsymbol{\nu}^{\mathrm{T}}\mathbf{b} \\ \text{subject to} & \mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}+\mathbf{c}-\boldsymbol{\lambda}=\mathbf{0} \\ & \boldsymbol{\lambda} \geq 0. \end{array}$$

This problem is equivalent to solving

$$\begin{array}{ll} \underset{\boldsymbol{\lambda},\boldsymbol{\nu}}{\text{maximize}} & -\boldsymbol{\nu}^{\mathrm{T}}\mathbf{b} \\ \text{subject to} & \mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}+\mathbf{c} \geq \mathbf{0}. \end{array}$$

Note that the dual problem associated with an LP is also an LP. Do not expect to find an analytical solution for an LP via duality theory.

6.3 Weak duality

We call $d^\star$ the optimal value of the dual problem. In other words $d^\star$ is the best lower bound on the solution of the primal problem that one can achieve by solving the dual problem. The following inequality holds

$$d^\star \leq p^\star. \tag{6.10}$$

This inequality is called *weak duality.*

- If $p^\star = -\infty$ (the primal problem is unbounded below) then the dual problem is infeasible.
- If $d^\star = +\infty$ (the dual problem is unbounded above) then the primal problem is infeasible.

We call the quantity $p^\star - d^\star$ the *duality gap.*

Recall the two-way partitioning problem. The dual problem associated with this problem is

$$\begin{array}{ll} \underset{\boldsymbol{\nu}}{\text{maximize}} & -\mathbf{1}^{\mathrm{T}}\boldsymbol{\nu} \\ \text{subject to} & \mathbf{W}+\operatorname{diag}(\boldsymbol{\nu}) \succcurlyeq 0. \end{array}$$

Now here is where duality is incredibly powerful. The primal problem is not convex and not solvable when the matrix $\mathbf{W}$ is large. On the contrary the dual problem is always convex and can be solved efficiently. As an example, for $n = 4$, we have generated 10000

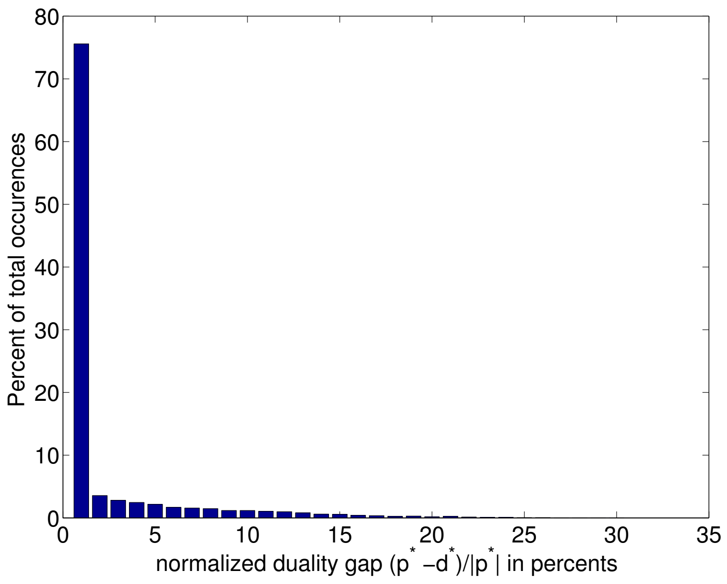


Figure 6.2: Histogram of the normalized duality gap

matrices $\mathbf{W}$, calculated $p^\star$ by brute force and calculated d^* by solving the dual problem with CVX for each of them. The outcome is illustrated in Figure 6.2. The results are astonishing! In more than 75% of the cases the dual solution is less than 1% away from the primal solution. In other words, for practical purpose we can find the solution of the primal problem by solving the dual problem. Note that we can find $p^\star$ using duality but we still need a heuristic to solve the primal problem. Duality helps us, in this case, to evaluate the performance of our heuristic.

6.4 Strong duality and Slater's constraint qualification

If the following equality holds

$$d^\star = p^\star, \tag{6.11}$$

then we say that *strong duality* holds. Probably the most important point here is that for most convex problems, strong duality holds. There exists a technical condition to guarantee that strong duality holds for a convex problem, it is called *Slater's condition.* This condition is very simple: given a convex problem, if there exists a point $\mathbf{x}$ in the domain $\mathcal{D}$, not situated on the border of $\mathcal{D}$, such that

$$f_i(\mathbf{x}) < 0, \quad i = 1, \dots, m, \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{6.12}$$

in other words, if there exists a strictly feasible point, then strong duality holds. Note that this condition is not very restrictive. Indeed assume you must design a product and f_i are specifications. If for one constraint k all feasible points are such that $f_k(\mathbf{x}) = 0$, it suffices to change the specification as $f_k(\mathbf{x}) - \epsilon \leq 0$ where ϵ is very small and now

Slater's condition holds. So for practical purpose, Slater's condition always holds and almost all convex problems have strong duality.

6.5 Optimality conditions

Certificate of suboptimality

One of the very interesting aspects of the duality theory is that the dual variables are certificates for the optimality of the primal variable. To see this, consider a feasible point $\mathbf{x}$ and its associated dual point $(\boldsymbol{\lambda}, \boldsymbol{\nu})$, it holds that

$$\begin{array}{rl} & p^\star \geq g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\ \Leftrightarrow & f_0(\mathbf{x}) - p^\star \leq f_0(\mathbf{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\nu}). \end{array}$$

The value $f_0(\mathbf{x}) - p^\star$ is the difference between how good we are doing and the best we could do, in other words $\epsilon = f_0(\mathbf{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ measures the suboptimality of the point $\mathbf{x}$. This statement is very powerful because we do not need to know $p^\star$ to know how far we are from it. The value ϵ is called the duality gap associated with $\mathbf{x}$ and $(\boldsymbol{\lambda}, \boldsymbol{\nu})$. The values $p^\star$ and $d^\star$ are both in the interval $[g(\boldsymbol{\lambda}, \boldsymbol{\nu}), f_0(\mathbf{x})]$. If $\epsilon = 0$, then $f_0(\mathbf{x}) = p^\star$, i.e., $\mathbf{x} = \mathbf{x}^\star$ and $(\boldsymbol{\lambda}, \boldsymbol{\nu}) = (\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$. In other words, if one can find a feasible point $\mathbf{x}$ and a dual point $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ such that $\epsilon = 0$, then $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ are certificates for the optimality of $\mathbf{x}$.

Complementary slackness

Assume $p^\star = d^\star$, then the following holds:

$$\begin{array}{rcl} f_0(\mathbf{x}^\star) & = & g(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star) \\ & = & \inf\limits_{\mathbf{x}} \left(f_0(\mathbf{x}) + \sum_{i=1}^m \lambda_i^\star f_i(\mathbf{x}) + \sum_{i=1}^p \nu_i^\star h_i(\mathbf{x})\right) \\ & \leq & f_0(\mathbf{x}^\star) + \sum_{i=1}^m \lambda_i^\star f_i(\mathbf{x}^\star) + \sum_{i=1}^p \nu_i^\star h_i(\mathbf{x}^\star) \\ & \leq & f_0(\mathbf{x}^\star). \end{array}$$

From the last line we find that all inequalities in this derivation are actually equalities. One conclusion from this is that $\mathbf{x}^\star$ minimizes $L(\mathbf{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$. The most important conclusion is that

$$\sum_{i=1}^m \lambda_i^\star f_i(\mathbf{x}^\star) = 0.$$

Since we know that for all i, $\lambda_i^\star f_i(\mathbf{x}^\star) \leq 0$, it follows that

$$\lambda_i^\star f_i(\mathbf{x}^\star) = 0. \tag{6.13}$$

This fact is called *complementary slackness*. It basically says that

$$\begin{cases} \text{if } \lambda_i^\star > 0 & \text{then} \quad f_i(\mathbf{x}^\star) = 0 \\ \text{if } f_i(\mathbf{x}^\star) > 0 & \text{then} \quad \lambda_i^\star = 0 \end{cases} \tag{6.14}$$

It is important to understand that complementary slackness holds for any $p^\star$ and $(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ if strong duality holds.

KKT optimality conditions

We all know that a differentiable function f has a minimum at $\mathbf{x}^\star$ if

$$\nabla_{\mathbf{x}} f(\mathbf{x}^\star) = 0.$$

We introduce here the *Karush-Kuhn-Tucker* (KKT) conditions which are equivalent conditions for optimization problems with differentiable objective and constraint functions.

1) KKT conditions for nonconvex problems

Assume that we have a primal optimal point $\mathbf{x}^\star$ and a dual optimal point $(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ with $p^\star = d^\star$. We have shown in the previous section that $\mathbf{x}^\star$ minimizes $L(\mathbf{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$. It follows that

$$\nabla_{\mathbf{x}} L(\mathbf{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star) = 0,$$

i.e.,

$$\nabla_{\mathbf{x}} f_0(\mathbf{x}^\star) + \sum_{i=1}^{m} \lambda_i^\star \nabla_{\mathbf{x}} f_i(\mathbf{x}^\star) + \sum_{i=1}^{p} \nu_i^\star \nabla_{\mathbf{x}} h_i(\mathbf{x}^\star) = 0.$$

Thus it holds

$$\begin{array}{rcll} f_i(\mathbf{x}^\star) & \leq & 0 & \text{for } i = 1, \ldots, m \quad \text{(the inequality constraints hold)} \\ h_i(\mathbf{x}^\star) & = & 0 & \text{for } i = 1, \ldots, p \quad \text{(the equality constraints hold)} \\ \lambda_i^\star & \geq & 0 & \text{for } i = 1, \ldots, m \quad \text{(the dual inequality constraints hold)} \\ \lambda_i^\star f_i(\mathbf{x}^\star) & = & 0 & \text{for } i = 1, \ldots, m \quad \text{(from complementary slackness)} \end{array} \tag{6.15}$$

and

$$\nabla_{\mathbf{x}} f_0(\mathbf{x}^\star) + \sum_{i=1}^{m} \lambda_i^\star \nabla_{\mathbf{x}} f_i(\mathbf{x}^\star) + \sum_{i=1}^{p} \nu_i^\star \nabla_{\mathbf{x}} h_i(\mathbf{x}^\star) = 0.$$

These five inequalities are called the KKT conditions. Given an optimization problem with differentiable objective and constraint functions for which strong duality holds, any point $\mathbf{x}^\star$, $(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ must satisfy the KKT conditions, that is KKT conditions are necessary.

2) KKT conditions for convex problems

If the optimization problem is convex then the KKT conditions are sufficient to guarantee that a point is optimal. Given any point $\tilde{\mathbf{x}}$, $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}})$, if these points satisfy KKT, i.e.,

$$\begin{array}{rcll} f_i(\tilde{\mathbf{x}}) & \leq & 0 & \text{for } i = 1, \ldots, m \\ h_i(\tilde{\mathbf{x}}) & = & 0 & \text{for } i = 1, \ldots, p \\ \tilde{\lambda}_i & \geq & 0 & \text{for } i = 1, \ldots, m \\ \tilde{\lambda}_i f_i(\tilde{\mathbf{x}}) & = & 0 & \text{for } i = 1, \ldots, m \\ \nabla_{\mathbf{x}} f_0(\tilde{\mathbf{x}}) + \sum_{i=1}^{m} \tilde{\lambda}_i \nabla_{\mathbf{x}} f_i(\tilde{\mathbf{x}}) + \sum_{i=1}^{p} \tilde{\nu}_i \nabla_{\mathbf{x}} h_i(\tilde{\mathbf{x}}) & = & 0, & \end{array} \tag{6.16}$$

then $\tilde{\mathbf{x}}$ and $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}})$ are dual and primal optimal with zero duality gap. To see this we first note that the two first inequalities state that $\tilde{\mathbf{x}}$ is primal feasible. Because $\tilde{\lambda}_i \geq 0$ and f_i are convex functions in $\mathbf{x}$, the function $L(\mathbf{x}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}})$ is convex since it is a positive weighted sum of convex functions. It comes from the last condition that the gradient of $L(\mathbf{x}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}})$ vanished at $\tilde{\mathbf{x}}$, which means that $\tilde{\mathbf{x}}$ minimizes $L(\mathbf{x}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}})$. Now we have

$$\begin{aligned} g(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}) &= \inf_{\mathbf{x}} L(\mathbf{x}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}) \\ &= L(\tilde{\mathbf{x}}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}}) \\ &= f_0(\tilde{\mathbf{x}}) + \underbrace{\sum_{i=1}^{m} \tilde{\lambda}_i f_i(\tilde{\mathbf{x}})}_{=0,\text{ complementary slackness}} + \underbrace{\sum_{i=1}^{p} \tilde{\nu}_i h_i(\tilde{\mathbf{x}})}_{=0,\text{ primal feasibility}} \\ &= f_0(\tilde{\mathbf{x}}). \end{aligned}$$

Therefore we have zero duality gap and $\tilde{\mathbf{x}}$ and $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\nu}})$ are primal and dual optimal. Let us sum-up this derivation. Given a convex problem with differentiable objective and constraint functions, if for a primal and dual point KKT holds, then this point is primal and dual optimal with zero duality gap.

Now the final observation of this section is: given a convex problem with differentiable objective and constraint functions, if Slater holds then a point is optimal if and only if KKT holds. Indeed, Slater tells us that we do have strong duality, i.e., there exist primal and dual points with zero duality gap, therefore, KKT must hold for this point. Conversely if KKT holds then this point is optimal. So now we have all tools at hand to prove that a given point is optimal.

6.6 Problem section

1. ***Max-flow min-cut problem***

   One of the most important network optimization problems is the *max-flow* problem. It consists of calculating the maximum amount of resources that can be transported between two vertices of a network. Consider the network in Figure 6.3. Each edge has a maximum capacity written in black. The transmitter sends

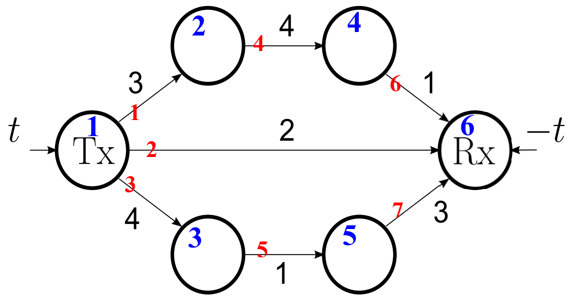


Figure 6.3: Example network. Blue numbers represent the index of the vertices. The red numbers represent the index of edges. Black numbers represent the maximum capacity of each edge.

   at a rate t to a receiver. The question is: what is the maximum rate t that can be transmitted through this network. Actually this problem can be modeled as an LP. We define a vector $\mathbf{e} \in \mathbb{R}^m$ given by

$$\mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix}.$$

   We can express the max-flow problem as follows

$$\begin{array}{ll} \underset{\mathbf{x},t}{\text{maximize}} & t \\ \text{subject to} & \mathbf{A}\mathbf{x} = t\mathbf{e} \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{d}, \end{array}$$

   where $\mathbf{d} \in \mathbb{R}^n$ is a capacity vector and d_i is the capacity of the edge i.

   (a) Using CVX, give the max-flow of the network illustrated in Figure 6.3.

   (b) Rewrite the max-flow problem such that the only optimization variable is

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ t \end{bmatrix}.$$

   (c) Give the dual problem of the max-flow problem from (b).

(d) Using CVX, solve the dual problem of the max-flow problem. Do you find $p^\star = d^\star$?

[**This part is a bonus question**] Another very famous problem in network optimization is the *min-cut* problem. We define a *s-t cut* as a set $\mathcal{C} = (\mathcal{S}, \mathcal{T})$ where $\mathcal{S}$ and $\mathcal{T}$ are two sets where $\mathcal{S}$ contains a number of vertices and $\mathcal{T}$ all other vertices. The *cut-set* of $\mathcal{C}$ is the set of all edges which both in $\mathcal{S}$ and $\mathcal{T}$. The capacity of a s-t cut is the sum of the capacity of all edges in the cut-set. The min-cut problem searches for the s-t cut with the smallest capacity. For example, in Figure 6.3, the min-cut is the capacity of the set composed of the two edges with capacity 1 and the edge with capacity 2. The most famous theorem of network optimization is that for a given network, the optimal value of the max-flow problem and the min-cut problem are the same. In fact these two problems are dual.

(e) Explain why the dual problem found in question (b) solves the min-cut problem.

2. ***Transportation problem*** [**bonus problem**]

We first consider the network illustrated in Figure 6.4. The scenario illustrated

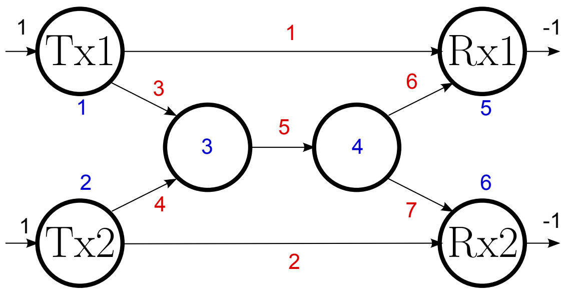


Figure 6.4: Example network

is the following. We have two warehouses and two factories linked by a train network. The first warehouse needs to ship products to the the first factory and the second warehouse to the second factory. For that purpose, each factory has one train. Each section (edge) of the train network has a cost. When a train occupies a section we have to pay the cost of the section. Each section is either occupied or not. Of course, a section cannot be occupied by two trains! The network illustrated in Figure 6.4 is represented by a matrix $\tilde{\mathbf{A}} \in \mathbf{R}^{m\times n}$, the costs for the different sections are represented by a vector $\tilde{\mathbf{c}} \in \mathbf{R}^n$. We define the vectors $\mathbf{x}_1 \in \{0,1\}^n$ and $\mathbf{x}_2 \in \{0,1\}^n$ as the paths followed by the first and second train.

We define

$$\mathbf{b}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} \qquad \mathbf{b}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix},$$

as the demand vectors for the link warehouse 1 to factory 1 and warehouse 2 to factory 2. Our objective is to deliver the two trains while

1) minimizing the cost of the overall delivery and

2) minimizing the difference between the cost of the first and the second train.

We can write our optimization problem as follows

$$\begin{array}{ll} \underset{\mathbf{x}_1,\mathbf{x}_2}{\text{minimize}} & \tilde{\mathbf{c}}^{\mathrm{T}}\mathbf{x}_1 + \tilde{\mathbf{c}}^{\mathrm{T}}\mathbf{x}_2 + t|\tilde{\mathbf{c}}^{\mathrm{T}}(\mathbf{x}_1 - \mathbf{x}_2)| \\ \text{subject to} & \tilde{\mathbf{A}}\mathbf{x}_1 = \mathbf{b}_1 \\ & \tilde{\mathbf{A}}\mathbf{x}_2 = \mathbf{b}_2 \\ & \mathbf{x}_{1i} \in \{0,1\}, \quad \mathbf{x}_{2i} \in \{0,1\}, \quad \mathbf{x}_{1i} + \mathbf{x}_{2i} \in \{0,1\}, \quad i = 1,\dots,n, \end{array}$$

where t is a tradeoff parameter representing the importance that the cost for the two trains is the same. For now we will take $t = 1$. Of course this problem is not convex. Our goal is to approximate the solution of this problem using duality theory.

(a) Consider the unconstrained optimization problem

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{x}\|_1.$$

Rewrite this problem as an LP by introducing a new variable $\mathbf{r}$.

(b) We define the following new system parameters

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ r \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \tilde{\mathbf{A}} & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times 1} \\ \mathbf{0}_{m\times n} & \tilde{\mathbf{A}} & \mathbf{0}_{m\times 1} \\ \mathbf{0}_{1\times n} & \mathbf{0}_{1\times n} & 0 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{I}_n & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times 1} \\ \mathbf{0}_{n\times n} & \mathbf{I}_n & \mathbf{0}_{n\times 1} \\ \mathbf{I}_n & \mathbf{I}_n & \mathbf{0}_{n\times 1} \end{bmatrix}$$

and

$$\mathbf{c} = \begin{bmatrix} \tilde{\mathbf{c}} \\ \tilde{\mathbf{c}} \\ t \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ 0 \end{bmatrix}, \quad \mathbf{d}_1 = \begin{bmatrix} \tilde{\mathbf{c}} \\ -\tilde{\mathbf{c}} \\ -1 \end{bmatrix}, \quad \mathbf{d}_2 = \begin{bmatrix} \tilde{\mathbf{c}} \\ -\tilde{\mathbf{c}} \\ 1 \end{bmatrix}.$$

With the help of these new parameters, rewrite the optimization problem with only $\mathbf{x}$ as optimization variable.

(c) One way to find a lower bound for our optimization problem is called *Boolean relaxation*. We relax the Boolean constraints as follow

$$0 \le \mathbf{x}_{1i} \le 1, \quad 0 \le \mathbf{x}_{2i} \le 1, \quad 0 \le \mathbf{x}_{1i} + \mathbf{x}_{2i} \le 1, \quad i = 1,\dots,n.$$

Using CVX, solve the relaxed problem and give a lower bound on the solution of the original problem. We consider the following cost vector

$$\tilde{\mathbf{c}} = [8\ 8\ 1\ 1\ 1\ 1\ 1]^{\mathrm{T}}.$$

(d) Another way to find such a lower bound is of course duality. Give the Lagrangian L of the optimization problem. For that you will need to introduce dual variables $\lambda_1 \in \mathbb{R}$ and $\lambda_2 \in \mathbb{R}$ associated with inequality constraints and dual variables $\boldsymbol{\nu}_1 \in \mathbb{R}^{2m+1}$ and $\boldsymbol{\nu}_2 \in \mathbb{R}^{3n}$ associated with equality constraints. You should also use the fact that

$$\sum_{i=1}^{3n} \nu_{2i}((\mathbf{h}_i\mathbf{x})^2 - \mathbf{h}_i\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathrm{diag}(\boldsymbol{\nu}_2)\mathbf{H}\mathbf{x} - \boldsymbol{\nu}_2^{\mathrm{T}}\mathbf{H}\mathbf{x},$$

where $\mathbf{h}_i$ is the i-th row of $\mathbf{H}$.

(e) We now need to find $g(\lambda_1, \lambda_2, \boldsymbol{\nu}_1, \boldsymbol{\nu}_2)$, which is the infimum of $L(\mathbf{x}, \lambda_1, \lambda_2, \boldsymbol{\nu}_1, \boldsymbol{\nu}_2)$ over $\mathbf{x}$. Give a condition on the matrix $\mathbf{H}^{\mathrm{T}}\mathrm{diag}(\boldsymbol{\nu}_2)\mathbf{H}$ so that $g(\lambda_1, \lambda_2, \boldsymbol{\nu}_1, \boldsymbol{\nu}_2)$ is not equal to $-\infty$.

(f) Assuming that the condition in question (e) holds, find $\mathbf{x}_{\min}$ that minimizes L. For that solve the equation

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \lambda_1, \lambda_2, \boldsymbol{\nu}_1, \boldsymbol{\nu}_2) = \mathbf{0}_{2n+1\times 1}.$$

So that your solution holds, you will need a very important fact that we saw in the first session. There exists a solution $\mathbf{y}$ to a system of the form

$$\mathbf{P}\mathbf{y} = \mathbf{q},$$

where $\mathbf{P}$ is not necessarily invertible, if and only if $\mathbf{q} \in \mathcal{R}(\mathbf{P})$. **This condition should be part of your dual problem**. If this condition holds, we have

$$\mathbf{y} = \mathbf{P}^{\dagger}\mathbf{q},$$

where $\mathbf{P}^{\dagger}$ is the pseudoinverse of $\mathbf{P}$ and $\mathbf{P}\mathbf{P}^{\dagger}\mathbf{q} = \mathbf{q}$ if $\mathbf{q} \in \mathcal{R}(\mathbf{P})$.

(g) Give the dual problem to the problem of question (b).

(h) In this form we cannot solve the dual problem. First rewrite your problem as follows

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{maximize}} & f_0(\mathbf{x}) \\ \text{subject to} & f_i(\mathbf{x}) \le 0 \\ & h_i(\mathbf{x}) = 0 \end{array} \quad \Rightarrow \quad \begin{array}{ll} \underset{\mathbf{x},\delta}{\text{maximize}} & \delta \\ \text{subject to} & \delta \le f_0(\mathbf{x}) \\ & f_i(\mathbf{x}) \le 0 \\ & h_i(\mathbf{x}) = 0 \end{array}$$

Now consider the following proposition. Given a symmetric singular matrix $\mathbf{M}$, a vector $\mathbf{v}$ and a scalar s, it holds

$$M \succcurlyeq 0, \quad \mathbf{v} \in \mathcal{R}(\mathbf{M}), \quad s - \mathbf{v}^{\mathrm{T}}\mathbf{M}^{\dagger}\mathbf{v} \geq 0 \quad \Leftrightarrow \quad \begin{bmatrix} \mathbf{M} & \mathbf{v} \\ \mathbf{v}^{\mathrm{T}} & s \end{bmatrix} \succcurlyeq 0.$$

This is a simplified version of the Schur complement for singular matrices. Using this proposition, rewrite your dual problem in a form that we can solve.

(i) Using CVX, solve the dual problem and give a lower bound on the solution of the original problem. How does this bound compares to the the one of Boolean relaxation?

(j) Now that we have a lower bound on the solution of the primal problem, we need to find a heuristic to solve the primal problem. The bound we have found will help us to evaluate the efficiency of our heuristic. We first propose the following heuristic.

   1) Find the cheapest path for one train only, consider that this train drives alone on the network. This gives you a cost c_1.

   2) Delete all edges driven by the first train from the network.

   3) Find the cheapest path for the second train only, considering the network with removed edges. This gives you a cost c_2.

   4) The heuristic achieves an overall cost of $c_1 + c_2 + t|c_1 - c_2|$.

   Implement this heuristic. How good is this heuristic for $t = 1$? And for $t = 2$?

(k) Can you find a better heuristic for the case that $t > 1$? Implement a new heuristic in the file `train.m` and compare it to the dual bound.

7 Approximation, vector optimization and regularization

7.1 Norm and penalty function approximation problems

In Section 1, we discussed several implications of the relation $\mathbf{y} = \mathbf{A}\mathbf{x}$, where $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{y} \in \mathbb{R}^m$ are problem specific data, and $\mathbf{x} \in \mathbb{R}^n$ is the variable. For example, we want to find an $\mathbf{x}$ that is plausible under the observation $\mathbf{y}$ (see subsection 1.1.3). When we have $m = n$ and $\mathcal{R}(\mathbf{A}) = \mathbb{R}^n$, then we can directly conclude that the only solution is simply given by

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}. \tag{7.1}$$

Needless to say, in theory, this is not an interesting case, let alone what happens in practice is usually a completely different story. Neither will our observations be consistent, i.e., in the range of the measurement matrix $\mathbf{A}$, nor can we always assume the measurement matrix to be square[11]. In fact, for an approximation problem we will assume that $m \geq n$. We tried to fix the above problems by using the singular value decomposition (SVD) of $\mathbf{A}$. The goal of this section is to study these problems in more detail.

The basic and simplest *norm approximation problem* is given by the unconstrained convex optimization problem of the form

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_p, \tag{7.2}$$

where $\|\cdot\|_p$ is an ℓ_p-norm, $1 \leq p \leq \infty$. A solution of this problem is called *approximate solution* of $\mathbf{A}\mathbf{x} \approx \mathbf{y}$, with respect to the corresponding norm. The problem in (7.2) minimizes the so-called *residual*, that is,

$$\|\mathbf{r}\|_p = \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_p = \left(|r_1|^p + \cdots + |r_m|^p\right)^{1/p}. \tag{7.3}$$

The residual can be interpreted as the part of uncertainty or inconsistency, which we would like to keep small. The interesting point is that depending on the choice of p (and later on penalty functions), we can make different kinds of residuals small. It is here where a thoughtfully chosen norm (or penalty function) can really make a difference.

7.1.1 Example: Least-squares approximation

Consider the case where $p = 2$. We can transform the objective of (7.2) into an equivalent objective by taking the square. The resulting optimization problem is called *least-squares approximation* and comes up in a myriad of different fields. The least-squares approximation problem can be solved analytically. We first notice that

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 = \mathbf{x}^\mathrm{T}\mathbf{A}^\mathrm{T}\mathbf{A}\mathbf{x} - 2\mathbf{y}^\mathrm{T}\mathbf{A}\mathbf{x} + \mathbf{y}^\mathrm{T}\mathbf{y}. \tag{7.4}$$

This objective is minimized if and only if a point $\mathbf{x}$ satisfies the KKT conditions (see (6.16)). In this case, only the gradient condition has to be evaluated

$$2\mathbf{A}^\mathrm{T}\mathbf{A}\mathbf{x} - 2\mathbf{A}^\mathrm{T}\mathbf{y} = \mathbf{0} \quad \Leftrightarrow \quad \mathbf{x}^\star = (\mathbf{A}^\mathrm{T}\mathbf{A})^{-1}\mathbf{A}^\mathrm{T}\mathbf{y}, \tag{7.5}$$

[11]See also problem 1.2: Sensor failure detection with noise.

where we assumed that $\mathbf{A}$ is full column rank. Interestingly, the solution of a least-squares approximation problem can be easily represented in terms of the SVD by noting that

$$\begin{aligned}
\mathbf{x}^\star &= (\mathbf{A}^\mathrm{T}\mathbf{A})^{-1}\mathbf{A}^\mathrm{T}\mathbf{y} && (7.6)\\
&= \left((\mathbf{U\Sigma V}^\mathrm{T})^\mathrm{T}(\mathbf{U\Sigma V}^\mathrm{T})\right)^{-1}(\mathbf{U\Sigma V}^\mathrm{T})^\mathrm{T}\mathbf{y} && (7.7)\\
&= \mathbf{V\Sigma}^{-1}\mathbf{U}^\mathrm{T}\mathbf{y}, && (7.8)
\end{aligned}$$

where the full column rank of $\mathbf{A}$ implies $\mathbf{\Sigma} \succ 0$. Two important comments are in order. First, although the SVD gives, at least, conceptual insights, the actual way how least-squares approximation problems are solved efficiently depends on many parameters, e.g., matrix structure, size, or time constraints. An excellent starting point for numerical linear algebra is the Appendix C in [6]. A deeper treatment can be found in the (almost) classic book [8]. Second, despite its simplicity, it is important to point out that the direct application of the pseudo-inverse formula (7.5) might be very limited. Introducing constraints and other norm functions will most probably result in optimization problems that don't offer explicit analytical solutions.

7.1.2 Penalty function approximation

We have already seen that a norm approximation problem will try to keep the residuals

$$(|r_1|^p + |r_2|^p + \cdots + |r_m|^p)^{1/p} \tag{7.9}$$

small, which is equivalent to keeping

$$|r_1|^p + |r_2|^p + \cdots + |r_m|^p \tag{7.10}$$

small. We will now extend the notion of norm approximation problems based on *penalty functions.* A penalty function $\phi : \mathbb{R} \to \mathbb{R}$ gives different weights to a residual, depending on the size of the residual. The *penalty function approximation problem* can be stated as

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \phi(r_1) + \phi(r_2) + \cdots + \phi(r_m) \\ \text{subject to} & \mathbf{r} = \mathbf{Ax} - \mathbf{y}. \end{array} \tag{7.11}$$

When ϕ is a convex function, the whole problem is a convex optimization problem. Adding convex constraints to the penalty function approximation problem will not change the convexity. For instance, we could require that $\mathbf{x}$ is a stochastic vector and hence simply use a polyhedral description of the feasible set, i.e., $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{1}^\mathrm{T}\mathbf{x} = 1$. Evidently, large values of $\phi(r_i)$ will incur a high cost. As one example, consider the so-called *deadzone-linear* penalty function that is defined as

$$\phi(u) = \begin{cases} 0 & |u| \leq a \\ |u| - a & |u| > a. \end{cases} \tag{7.12}$$

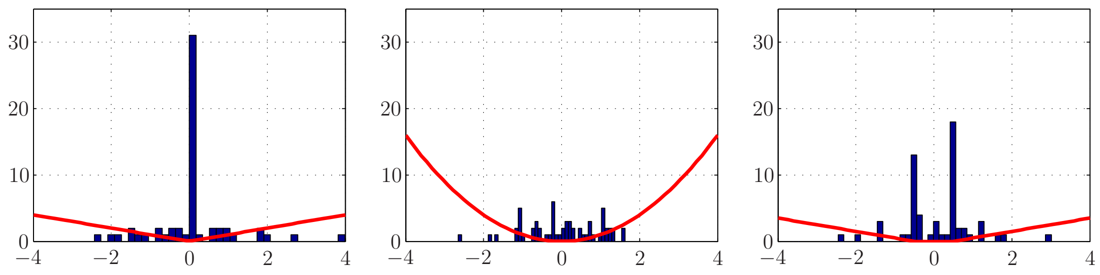
This function definitely likes to have values inside the interval $|u| \leq a$, as they come without any cost. For values $|u| > a$, the cost grows linearly only. This means, it is still acceptable to have some very high residuals in the objective, since we only have to pay a linearly growing bill. Note that ℓ_p-norms are penalty functions as well.

Let's investigate the influence of some frequently used penalty functions on the distribution of the residuals. We set up the basic penalty function approximation problem as formulated in (7.11). The problem data $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ is generated randomly, where $m = 60$, $n = 30$. We take the ℓ_1- and ℓ_2-norm, and the deadzone-linear function (with $a = 0.5$) as penalty functions. The resulting (optimal) residuals are depicted in histograms (see Figure 7.1). We can make the following observations:

- In case of small residuals, i.e. $r_i \approx 0$, the ℓ_1-norm puts the most weight on the objective. Minimizing the ℓ_1-norm results in many residuals that are equal to zero[12]. Higher residuals are also acceptable, since the cost in the objective grows only linearly. We could also say that the ℓ_1-minimization is not heavily influenced by large outliers.

- When compared to the ℓ_1-norm, the values of the ℓ_2-norm are not very large for small arguments. This means we can afford to have many small residuals, without making the objective too large. However, for growing arguments, the quadratic part of the ℓ_2-norm starts getting more costly. Large outliers would increase the objective dramatically.

- The deadzone-linear puts no weight on residuals that are inside $|u| \leq a$. We freely accept residual errors of order a. Outside this interval, we only have a ℓ_1 cost.



(a) ℓ_1-norm (b) ℓ_2-norm (c) deadzone-linear

Figure 7.1: Histogram counts of residual amplitudes. The red lines denote the corresponding penalty function.

7.2 Least-norm and least-penalty problems

Consider now the linear mapping $\mathbf{y} = \mathbf{A}\mathbf{x}$, with $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $m < n$. The difference to norm approximation problems is that we are now dealing with underdetermined systems of equalities. Assuming that the rows of $\mathbf{A}$ are independent (note the slang in this

[12]This particular behavior is an essential feature of a whole research area called compressed sensing. However, it still remains some kind of ℓ_1-trick.

statement), we could try to find any solution of the above equation. This comes up in control or design problems, where we would like to find all possible control inputs $\mathbf{x}$ such that a desired control output $\mathbf{y} = \mathbf{y}_{\text{des}}$ is achieved. However, as mentioned in Subsection 1.1.3, it might be also interesting to find solutions that are somehow small in their size. We can measure the size of a solution by its ℓ_p-norm. Minimizing this norm subject to the desired output leads to the basic *least-norm problem*, which has the form

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \|\mathbf{x}\|_p \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{y}. \end{array} \tag{7.13}$$

Obviously, this problem is a convex optimization problem.

7.2.1 Example: Least-squares solution of linear equations

We consider the minimization of the ℓ_2-norm. Squaring the objective yields the equivalent problem

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{x}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{y}. \end{array} \tag{7.14}$$

Evaluating the KKT conditions gives

$$2\mathbf{x}^{\star} + \mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}^{\star} = \mathbf{0}, \qquad \mathbf{A}\mathbf{x}^{\star} = \mathbf{y}, \tag{7.15}$$

where $\boldsymbol{\nu} \in \mathbb{R}^m$ is the dual variable. From the first equation we know that $\mathbf{x}^{\star} = -(1/2)\mathbf{A}^{\mathrm{T}}\boldsymbol{\nu}^{\star}$. This can be used to solve the second equation. Since $\mathbf{A}$ is full row rank, it follows that $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ is nonsingular and hence invertible. The resulting primal dual optimal point can then be expressed as

$$\boldsymbol{\nu}^{\star} = -2(\mathbf{A}\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{y}, \qquad \mathbf{x}^{\star} = \mathbf{A}^{\mathrm{T}}(\mathbf{A}\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{y} \tag{7.16}$$

In terms of the SVD, we can also see that $\mathbf{x}^{\star} = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{y}$.

7.2.2 Relation to norm approximation problems

One might ask if there is a relationship between least-norm and norm approximation problems. Consider the basic least-norm problem (7.13). Assume that $\mathbf{x}_0$ is any solution of $\mathbf{A}\mathbf{x} = \mathbf{y}$. We can pick a matrix $\mathbf{Z} \in \mathbb{R}^{n \times k}$ such that the columns of $\mathbf{Z}$ form a basis for the nullspace of $\mathbf{A}$, that is, $\mathbf{A}\mathbf{Z} = \mathbf{0}_{m \times k}$. Clearly, any solution of $\mathbf{A}\mathbf{x} = \mathbf{y}$ can be expressed as

$$\mathbf{A}(\mathbf{x}_0 + \mathbf{Z}\mathbf{u}) = \mathbf{A}\mathbf{x}_0 + \mathbf{A}\mathbf{Z}\mathbf{u} = \mathbf{A}\mathbf{x}_0, \tag{7.17}$$

where $\mathbf{u} \in \mathbb{R}^k$. Using this, then the basic least-norm problem can be rewritten in terms of the following norm approximation problem

$$\underset{\mathbf{u}}{\text{minimize}} \quad \|\mathbf{x}_0 + \mathbf{Z}\mathbf{u}\|_p. \tag{7.18}$$

We have mentioned this relation for the sake of completeness. In order to study convex optimization problems, both formulations are valid. However, the interpretation of any solution has to be treated in terms of the above transformation.

7.2.3 Least-penalty problems

Obviously, the concept of penalty functions can be easily extended to least-norm problems. The *least-penalty problem* is given by

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \phi(x_1) + \phi(x_2) + \cdots + \phi(x_n) \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{y}. \end{array} \tag{7.19}$$

When the penalty function ϕ is convex, the least-penalty problem is a convex optimization problem. We can also put in additional convex constraints, as well as reformulate this problem in terms of a penalty approximation problem.

7.3 Vector optimization problems

7.3.1 Definition

Consider the general *vector optimization* problem

$$\begin{array}{lll} \underset{\mathbf{x}}{\text{minimize}} \,(\text{with respect to } K) & f_0(\mathbf{x}) & \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, & i = 1, \ldots, m \\ & h_j(\mathbf{x}) = 0, & j = 1, \ldots, p. \end{array} \tag{7.20}$$

As expected, $\mathbf{x} \in \mathbb{R}^n$ is the optimization variable, $f_i : \mathbb{R}^n \to \mathbb{R}$ and $h_j : \mathbb{R}^n \to \mathbb{R}$ are the inequality and equality constraint functions, and K is a proper cone. If the constraint set is convex and f_0 is K-convex, we also say that the optimization problem (7.20) is a *convex vector optimization problem.* Compare this to the standard optimization problem as given in (2.1),(2.6). The main differences are as follows:

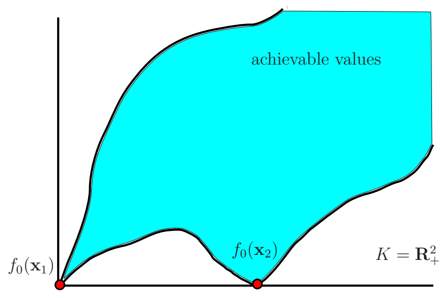
1. The objective function f_0 can now take values in a q-dimensional space, that is, we have $f_0 : \mathbb{R}^n \to \mathbb{R}^q$.

2. To compare different feasible (if not optimal) points, we have to define a proper cone $K \in \mathbb{R}^q$ in order to decide if $f_0(\mathbf{x}) \succeq_K f_0(\mathbf{y})$ or $f_0(\mathbf{x}) \preceq_K f_0(\mathbf{y})$.

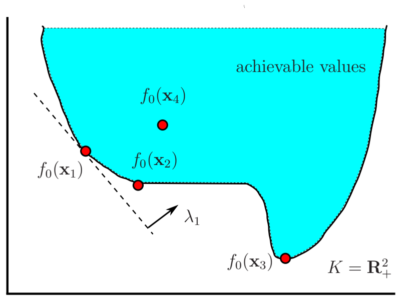
Here comes the tricky part: It is possible that two objective values are not comparable, i.e., there is no feasible point that is smaller (with respect to $\preceq_K$) than all other feasible points, see also Subsection 2.2.4.

7.3.2 Pareto optimality and scalarization

In order to analyze possible solutions of this problem, we will introduce the concept of minimum and minimal elements of a set S. A point $x \in S$ is called *minimum element*, if for all $y \in S$ we have $x \preceq_K y$. A point x is called *minimal element*, if for any $y \in S$, $y \preceq_K x$ only if $x = y$. Note that as opposed to minimum elements, from this definition it does not follow that $x \preceq_K y$. Roughly speaking, minimum element translates into "every other point is worse, you are the best". On the other hand, minimal means something like "no other point is better, but you are also not the best".



(a) $f_0(\mathbf{x}_1)$ is the minimum element



(b) $f_0(\mathbf{x}_i), i = 1, 2, 3$ are minimal elements

Figure 7.2: Comparison between minimum and minimal elements. When comparing these properties, we always have to refer to a proper cone K. Often this will be the nonnegative orthant.

We will now extend this concept to the objective of a vector optimization problem. Let us have a look at Figure 7.2. On the left hand side, we can see that out of all achievable points the minimum element is given by $f_0(\mathbf{x}_1)$. On the right hand side, there is no minimum element. Clearly, $f_0(\mathbf{x}_4)$ seems to be a bad choice in any case. We could go down and left and would do better. However, several intervals on the boundary contain minimal elements, including the three minimal elements $f_0(\mathbf{x}_1), f_0(\mathbf{x}_2)$ and $f_0(\mathbf{x}_3)$. The mathematical formalization of this principle is called Pareto optimality. A feasible point $\mathbf{x}$ is *Pareto optimal* if $f_0(\mathbf{x})$ is a minimal element of the set of achievable values.

So far we have seen that vector optimization problems may or may not attain a minimum value for feasible $\mathbf{x}$. In some way, the best we can do is to find minimal values, that is, Pareto optimal points. *Scalarization* is one technique for finding Pareto optimal points. Consider the scalarized vector optimization problem

$$\begin{array}{lll} \underset{\mathbf{x}}{\text{minimize}} & \boldsymbol{\lambda}^{\mathrm{T}} f_0(\mathbf{x}) & \\ \text{subject to} & f_i(\mathbf{x}) \le 0, & i = 1, \dots, m \\ & h_j(\mathbf{x}) = 0, & j = 1, \dots, p, \end{array} \tag{7.21}$$

where $\boldsymbol{\lambda} \succ_{K^*} \mathbf{0}$, i.e., any positive vector in the so-called dual cone K^*. From now on, we will assume that $K = \mathbb{R}^n_+$. The dual cone is then given by $K^* = K = \mathbb{R}^n_+$, which means the nonnegative orthant is self-dual. To make a long story short, we can simply take any positive vector $\boldsymbol{\lambda}$ and scalarize the above problem. Why does scalarization help us find Pareto optimal points? Suppose $\mathbf{x}$ is an optimal solution of (7.21). Furthermore, assume $\mathbf{x}$ is not a Pareto optimal point for the vector optimization problem (7.20). Let us take a feasible $\mathbf{y}$, such that

$$f_0(\mathbf{y}) \preceq_K f_0(\mathbf{x}) \quad \Leftrightarrow \quad f_0(\mathbf{x}) - f_0(\mathbf{y}) \succeq_K \mathbf{0}, \qquad f_0(\mathbf{x}) \neq f_0(\mathbf{y}). \tag{7.22}$$

From the last inequality, it follows with $\boldsymbol{\lambda} \succ_{K^*} \mathbf{0}$ that

$$\boldsymbol{\lambda}^{\mathrm{T}}(f_0(\mathbf{x}) - f_0(\mathbf{y})) > 0 \quad \Leftrightarrow \quad \boldsymbol{\lambda}^{\mathrm{T}} f_0(\mathbf{x}) > \boldsymbol{\lambda}^{\mathrm{T}} f_0(\mathbf{y}). \tag{7.23}$$

But this can not hold because we assumed that $\mathbf{x}$ is an optimal solution of (7.21). Therefore, $\mathbf{x}$ must be a Pareto optimal point. In order to find different Pareto optimal points, we have to modify $\boldsymbol{\lambda}$. Be careful, there exist Pareto optimal points that can not be found through scalarization, for example, the point $f_0(\mathbf{x}_2)$ in Figure 2(b).

How about convex vector optimization problems? For this case and $\boldsymbol{\lambda} \succ_{K^*} \mathbf{0}$, it can be shown that every Pareto optimal point $\mathbf{x}$ is a solution of the scalarized problem[13]. In other words, through the method of scalarization we will find all Pareto optimal points, again, thanks to convexity.

7.3.3 Bi-criterion problems and regularization

We define a *bi-criterion problem* as

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & f_0(\mathbf{x}) = (F_1(\mathbf{x}), F_2(\mathbf{x})) \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \\ & h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, p, \end{array} \tag{7.24}$$

where the corresponding cone is simply $K = \mathbb{R}^2_+$. For example, the risk-sensitive diet problem (5.4) can be stated as a (convex) bi-criterion optimization problem, i.e.,

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & (F_1(\mathbf{x}), F_2(\mathbf{x})) = (\bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x}, \mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{x}) \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}. \end{array} \tag{7.25}$$

In order to solve this problem, we used the simple argumentation that we can control the trade-off between expected cost $\bar{\mathbf{c}}^{\mathrm{T}}\mathbf{x}$ and cost variance $\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{x}$ through a risk-aversion parameter γ. What we really did was, in fact, a scalarization technique using $\boldsymbol{\lambda}^{\mathrm{T}} = \begin{pmatrix} 1 & \gamma \end{pmatrix} \succeq \mathbf{0}$. (Remember what happened for γ?)

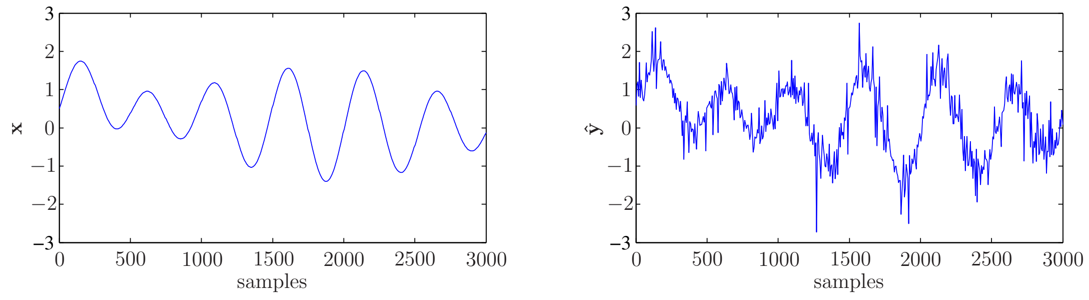
Now consider a bi-criterion problem that is related to approximation problems. On the one hand, we want to achieve a small residual. On the other hand, we might be interested to find, if possible, a small solution for the prescribed approximation problem. This means we would like to solve the bi-criterion problem

$$\underset{\mathbf{x}}{\text{minimize}} \quad (F_1(\mathbf{x}), F_2(\mathbf{x})) = (\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_p, \|\mathbf{x}\|_p). \tag{7.26}$$

In this approximation context, another term for this scalarization method is *regularization*. The corresponding unconstrained convex optimization problem is then given by

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_p + \gamma\|\mathbf{x}\|_p. \tag{7.27}$$

It should be pointed out that we can add further constraints, and include different norms or penalty functions.



(a) Original signal (b) Noisy observation

Figure 7.3: Original and noisy observation for the quadratic smoothing problem.

7.3.4 Example: Quadratic smoothing

An interesting special case of regularized approximation problems is the reconstruction of a signal corrupted by rapidly varying noise. The signal model is given by

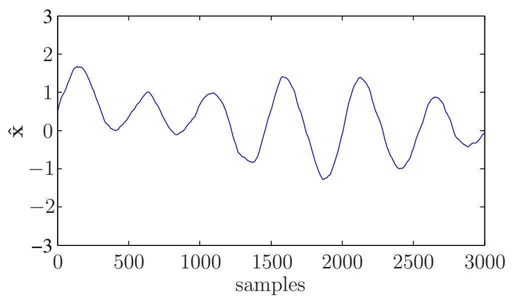
$$\mathbf{y} = \mathbf{x} + \mathbf{n}, \tag{7.28}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the signal vector, and $\mathbf{n} \in \mathbb{R}^n$ is additive noise. The noise consists of a weighted multiplicative mixture of a Gaussian, uniform, and exponential random vector. Our goal is to reconstruct the original signal $\mathbf{x}$ from the noisy observation $\mathbf{y}$ (see Figure 7.3). In estimation, e.g., maximum likelihood or maximum a posteriori estimation, people are traditionally trained to estimate the parameters of the noise in order to denoise the signal. We will not have the tedium of a long analytical derivation in order to find the exact probability density function of the noise. Another approach would consist in a carefully tuned low-pass filter. Instead, we let ourselves be guided by intuition and try the following regularized objective

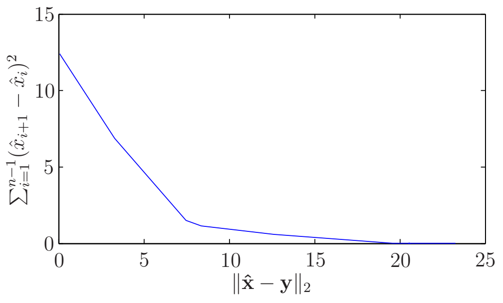
$$\underset{\hat{\mathbf{x}}}{\text{minimize}} \quad \|\hat{\mathbf{x}} - \mathbf{y}\|_2 + \gamma \sum_{i=1}^{n-1} (\hat{x}_{i+1} - \hat{x}_i)^2, \tag{7.29}$$

where $\gamma \geq 0$. This means we would like to find an $\hat{\mathbf{x}}$, which is consistent with the measurement $\mathbf{y}$ and should not vary too much from sample to sample. This sample-by-sample variation is smoothed by a quadratic function and tuned through the trade-off parameter γ. The results of this simple and intuitive method can be seen in Figure 7.4.

[13]There is a weird subtlety hidden in this statement. For $\boldsymbol{\lambda} \succeq_{K^*} \mathbf{0}$ (not $\boldsymbol{\lambda} \succ_{K^*} \mathbf{0}$!), it is true that not every solution of the scalarized problem is Pareto optimal.



(a) Denoised signal



(b) Optimal trade-off

Figure 7.4: Quadratic smoothing using $\gamma = 4.2$. The Pareto optimal trade-off curve was calculated using different values of γ.

7.4 Problem section

1. ***Outliers may not harm the majority, at least, when penalized correctly.*** Suppose we have a bunch of data points $(t_i, y(t_i))$. The data points are depicted in Fig. 7.5. Despite the presence of a few outliers, we should have the strong feeling that the majority obeys the linear relationship

$$y(t_i) = x_1 + x_2 \cdot t_i, \quad \text{for} \quad i = 1, \ldots, m, \tag{7.30}$$

where x_1 and x_2 are unknown variables, and m is the number of measurements.

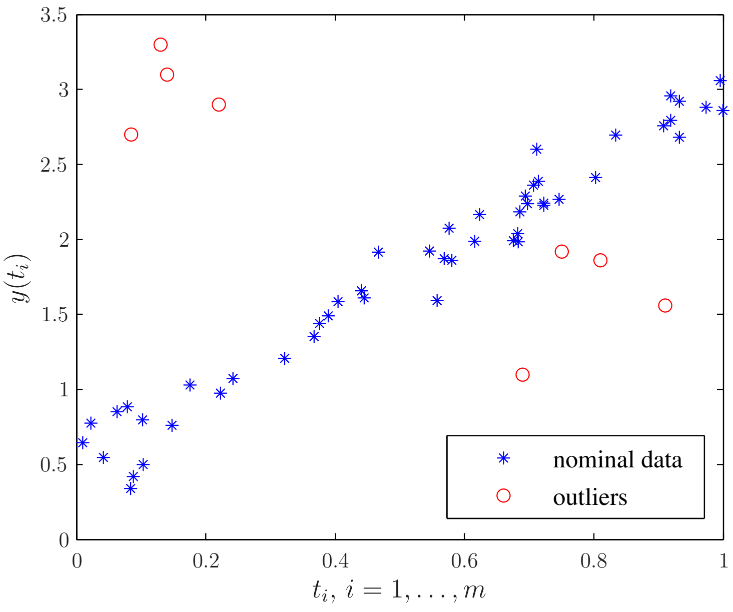


Figure 7.5: Data points with outliers.

(a) Our job is to find the unknown variables in (7.30). Formulate this task as a least-squares approximation problem.

(b) Load the file `data_robust_regression.mat`. The matrices `Y` and `Yo` contain the nominal and outlier data, respectively. In both, the first row contains the values of t_i, while the second row contains $y(t_i)$. We will start with the nominal data. Find the optimal values $x_1^\star$ and $x_2^\star$. In addition, save the norm of the optimal residual. Plot the linear relationship $y(t) = x_1^\star + x_2^\star \cdot t$ together with the data in a single figure.

(c) Repeat the steps from (b) for the complete data set, that is, nominal and outlier data. How does the presence of outliers affect the optimal linear relationship and the residual?

Obviously, eliminating the outliers from the data will result in a better norm approximation performance. However, having a small approximation error should not always be the guiding principle. Why? Because first of all, to decide if a given data point is an outlier or not may depend on the application, personal judgement etc. Second, even if we have a clear technical definition of outliers, in a higher dimension the identification of those can result in a computationally demanding task. Finally, removing outliers also changes the data, and this is not always justified in a scientific way.

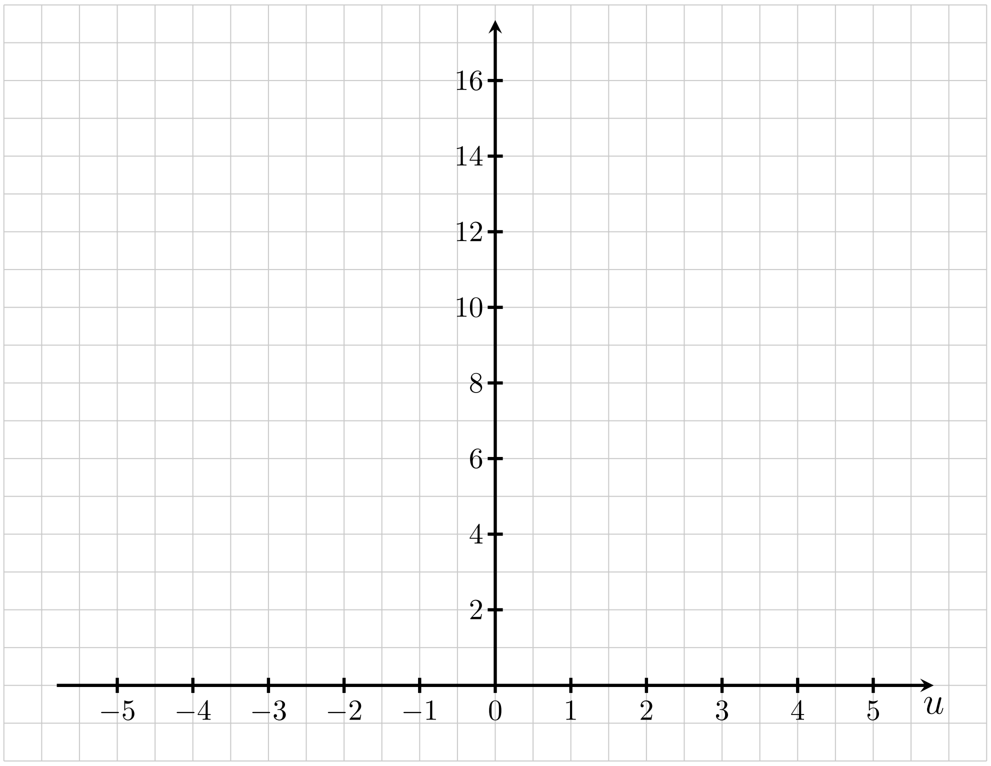
(d) We are now trying to find a nicely fitting linear relationship for the majority, while keeping the outliers as well. Consider the so-called *Huber penalty function* that can be stated as

$$\phi_{\text{hub}}(u) = \begin{cases} u^2 & |u| \leq M \\ M(2|u| - M) & |u| > M, \end{cases} \tag{7.31}$$

where $M > 0$. What are the basic properties and why should it work for our problem? What happens for $M \to 0$ and $M \to \infty$? Can we improve the Huber penalty function by taking the following function?

$$\phi_{\text{nch}}(u) = \begin{cases} u^2 & |u| \leq M \\ M^2 & |u| > M \end{cases} \tag{7.32}$$

Make a sketch of both functions for $M = 2$ for easier comparison.



(e) Repeat the steps from (b) for the complete data set by using the Huber penalty function. Take M based on your personal judgment and compare the results with (b) and (c). Make use of the MATLAB function `huber` contained in CVX (type `help huber` for learn about the function's interface).

2. ***ℓ_2 smoothing, ℓ_1 trend filtering, and being fooled by (non)-randomness.***[14] We are given a specific time series y_t consisting of deterministic components x_t hidden in a fairly erratic noise z_t, i.e., we have $y_t = x_t + z_t$. Unfortunately, we are not able to make assumptions about the noise statistics. The time series itself reflects some price behavior of a certain stock. Our goal is to analyze the stock price characteristics in order to draw conclusions to, say, economic trends. That is, we would like to estimate the price based on the given time series, without building a statistical framework around it.

   (a) An expert trader gives us a hint about the 'true' underlying deterministic price structure of this stock. The trader asserts that behind the random noise lies a portion consisting of smooth periodic components.

   Load the data file `data_l1_trend_filtering.mat`. Plot the initial time series `yti`. Can you, at least conceptually verify the traders hint?

   (b) We will now use ℓ_2 smoothing in order to reveal the deterministic price behavior. The objective to be minimized is defined as

$$\sqrt{\sum_{t=1}^{n}(y_t - \hat{x}_t)^2} + \gamma\sqrt{\sum_{t=1}^{n-1}(\hat{x}_{t+1} - \hat{x}_t)^2}. \qquad (7.33)$$

   Take 20 different values of the trade-off parameter γ, linearly spaced between 1 and 40. You might want to use the MATLAB function `linspace`. Plot the optimal trade-off curve for all values of γ. Use the same axes as given in Fig. 7.4. Why are the points on this curve Pareto-optimal?

   (c) Have a closer look at the optimal trade-off curve. Each point corresponds to a specific value of γ. Pick a reasonable point based on your personal judgement. Denoise the time series with the corresponding γ and plot the results. If there are still many sharp transitions present, try to pick a higher value of γ and repeat.

   (d) We now have access to a longer time series. The additional part is saved in the variable `ytf`. Stack the variables `yti` and `ytf` together in one vector `yt`. Repeat the ℓ_2 smoothing for the longer time series using γ from c). Does the resulting denoised time series look plausible?

   (e) Now let's say we have a strange feeling triggered by the recently added time series. The expert trader has tried to convince us (very eloquently!) that there is a smooth deterministic portion. But he is, alas, an expert trader because he might know more than we do, and simply exploits this situation of asymmetric information.

[14]Graciously borrowed from [15].

Let's make our objection to the smooth deterministic component more precise. We believe that the true component is not smooth, but piecewise affine. This could mean that a group of mischievous people tries to control the price trends over certain periods. Instead of ℓ_2 smoothing, we will now use a technique called ℓ_1 *trend filtering* [11], where the objective is given by

$$(1/2)\sum_{t=1}^{n}(y_t - \hat{x}_t)^2 + \gamma_{\ell_1}\sum_{t=2}^{n-1}|\hat{x}_{t-1} - 2\hat{x}_t + \hat{x}_{t+1}|. \tag{7.34}$$

The additional γ_{ℓ_1}-weighted term is called second-order difference. The factor (1/2) is for notational convenience.

Represent the second-order difference in a compact matrix-vector formulation. What is the precise meaning of the second-order difference, in particular, when will it be equal to zero? Take $\gamma_{l_1} = 5000$ and solve the ℓ_1 filtering problem for the stacked complete signal. Again, plot the results and make a comment on the plausibility.
Hint: Use the `cvx` function `sum_square` to implement the squared ℓ_2 norm.

(f) Load the vector `x_true` from the data file `data_l1_discussion.mat`, which, of course, contains the true deterministic component. Plot this true component together with the results of the ℓ_2 and ℓ_1 filtering. Comment on what you see. Have you been fooled by randomness?

3. ***Sparse signal recovery – compressed sensing.*** **[bonus exercise]**

Let's say we want to recover an unknown signal $\mathbf{x} \in \mathbb{R}^n$ from our (meanwhile) well known linear measurement model

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{7.35}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a measurement matrix, and $\mathbf{y} \in \mathbb{R}^m$ is the observation. Traditionally, in order to have a unique solution we need more measurements than unknown variables, i.e., $m > n$ [7]. Now suppose we know that the unknown signal contains at most $k < m$ nonzero elements. We then say that $\mathbf{x}$ is k-sparse. For a perfect sparse signal recovery, is it still necessary to have more measurements than unknown variables?

We start with one version of a *convex cardinality problem* that is given by [5]

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \operatorname{card}(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathcal{C}, \end{array} \tag{7.36}$$

where $\mathcal{C}$ is convex and

$$\operatorname{card}(\mathbf{x}) = \#\{x_i \neq 0\} \tag{7.37}$$

is the so-called cardinality function. This nonconvex function gives the number of nonzero components of $\mathbf{x}$. A convex cardinality problem would be convex, except for appearance of card in the objective.

(a) In the case of sparse signal recovery, the convex cardinality problem has the form

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \operatorname{card}(\mathbf{x}) \\ \text{subject to} & \mathbf{y} = \mathbf{A}\mathbf{x}. \end{array} \tag{7.38}$$

We are trying to find the simplest (sparsest) explanation fitting our observation. Note that we only know that $\mathbf{x}$ is k-sparse. We don't know the exact sparsity pattern. How could we solve the problem (7.38) globally?

(b) Consider the scalar cardinality function that is given by

$$\operatorname{card}(x) = \begin{cases} 0 & x = 0 \\ 1 & x \neq 0. \end{cases} \tag{7.39}$$

We will approximate this nonconvex function by using ℓ_p-norms. Plot the scalar cardinality function together with the scalar ℓ_1- and ℓ_2-norm on the interval $[-1, 1]$. By visual inspection, can you find a p such that the ℓ_p norm of x provides a better convex approximation of card than the ℓ_1-norm?

(c) Replacing the cardinality function by the ℓ_1-norm gives a heuristic for the nonconvex problem (7.38), that is,

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \|\mathbf{x}\|_1 \\ \text{subject to} & \mathbf{y} = \mathbf{A}\mathbf{x}. \end{array} \tag{7.40}$$

Load the data file `data_sparse_signal_recovery.mat`. This file contains the sparse signal `xtrue`, which we want to recover. Give the values of k and n. Generate $m_1 = 190$ and $m_2 = 290$ measurements by using a random matrix `A=randn(m,n)` (really!). Solve the problem (7.40) for m_1 and m_2. Plot the sparse and the recovered signals together in a scatter plot. For which number of measurements is the reconstruction perfect?

(d) Let's try to reduce the number of measurements while still attaining perfect reconstruction. By taking the ℓ_1-norm, the resulting cost or penalty for each element depends on the size of the magnitude. Suppose we have found a solution $\mathbf{x}^\star$ for the problem (7.40). We might draw the following conclusions:

1. $|x_i^\star|$ is 'large': the corresponding element is likely to be nonzero
2. $|x_i^\star|$ is 'small': the corresponding element is likely to be zero

The basic assumption is that after solving (7.40) once, we somehow have a first impression about the sparsity pattern. We will now use this information for an iterative algorithm, called *Reweighted ℓ_1 Minimization* [7]:

1. set initial weight $\mathbf{w}^{(0)} := \mathbf{1}$, number of iterations N, stability parameter ϵ

2. solve for $l = 0, \ldots, N-1$

$$\mathbf{x}^{(l)} = \arg\min_{\mathbf{x}} \|\operatorname{diag}(\mathbf{w}^{(l)})\mathbf{x}\|_1 \quad \text{subject to} \quad \mathbf{y} = \mathbf{A}\mathbf{x}$$

and set after each iteration $w_i^{(l+1)} := 1/(\epsilon + |x_i^{(l)}|)$

3. if $l = N - 1$, take the solution $\mathbf{x}^\star = \mathbf{x}^{(l)}$

Using the Reweighted ℓ_1 minimization algorithm, try to recover the sparse signal with only $m = 190$ measurements. Take $\epsilon = 0.1$ and $N = 5$. What is the meaning of $\mathbf{w}^{(l)}$? Can you find perfect reconstruction?

8 Semidefinite programming

8.1 Definition

A *semidefinite program* (SDP) is an optimization problem of the form

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & x_1\mathbf{F}_1 + \cdots + x_n\mathbf{F}_n + \mathbf{G} \preccurlyeq \mathbf{0} \\ & \mathbf{A}\mathbf{x} = \mathbf{b}, \end{array} \tag{8.1}$$

where $\mathbf{G}$, $\mathbf{F}_1$, ..., $\mathbf{F}_n \in \mathbb{S}^k$ and $\mathbf{A} \in \mathbb{R}^{p\times n}$. We can see that an SDP resembles an LP a lot. The difference is that for the inequality constraint, we take a linear combination of symmetric matrices, and the inequality is with respect to the positive semidefinite cone. The representation form of SDPs (8.1) might seem a bit complicated. In practice we will recognize an SDP, simply as a problem which has inequalities with respect to the positive semidefinite cone. But let us try to understand why the above formulation is general. Assume we want to solve the following problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & \|\mathbf{X}\|_{\mathrm{F}}^2 \\ \text{subject to} & \mathbf{X} \succcurlyeq \mathbf{C} \end{array} \tag{8.2}$$

with $\mathbf{X}$, $\mathbf{C} \in \mathbb{S}^n$ and $\|\mathbf{X}\|_{\mathrm{F}}$ denotes the Frobenius norm of $\mathbf{X}$, i.e.,

$$\|\mathbf{X}\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} X_{ij}^2} = \sqrt{\operatorname{tr}(\mathbf{X}^{\mathrm{T}}\mathbf{X})} = \sqrt{\sum_{i=1}^{n} \sigma_i^2(\mathbf{X})}.$$

Is the problem (8.2) an SDP? At first it is not evident that this problem can be recast in the form of (8.1). To see why this problem actually is a valid SDP, let us decompose $\mathbf{X}$ as follows for the case $n = 2$.

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \end{bmatrix} = x_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

If we define $\mathbf{F}_1 \triangleq \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}$, $\mathbf{F}_2 \triangleq \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$ and $\mathbf{F}_3 \triangleq \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}$, we can rewrite the problem (8.2) as

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & \operatorname{tr}(\mathbf{X}^{\mathrm{T}}\mathbf{X}) \\ \text{subject to} & x_1\mathbf{F}_1 + x_2\mathbf{F}_2 + x_3\mathbf{F}_3 + \mathbf{C} \preccurlyeq \mathbf{0}. \end{array} \tag{8.3}$$

Then we rewrite the objective function as

$$\operatorname{tr}(\mathbf{X}^{\mathrm{T}}\mathbf{X}) = \mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1 + \mathbf{x}_2^{\mathrm{T}}\mathbf{x}_2,$$

where $\mathbf{x}_i$ is the i-th column of $\mathbf{X}$. The problem (8.2) is then equivalent to

$$\begin{array}{ll} \underset{\mathbf{X},t_1,t_2}{\text{minimize}} & t_1 + t_2 \\ \text{subject to} & \mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1 \leq t_1 \\ & \mathbf{x}_2^{\mathrm{T}}\mathbf{x}_2 \leq t_2 \\ & t_1 \geq 0, \quad t_2 \geq 0 \\ & x_1\mathbf{F}_1 + x_2\mathbf{F}_2 + x_3\mathbf{F}_3 + \mathbf{C} \preccurlyeq \mathbf{0}. \end{array} \tag{8.4}$$

Further we use the following important Lemma called the *Schur complement.* Given a matrix $\mathbf{X} \in \mathbb{S}^n$ such that

$$\mathbf{X} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{C} \end{bmatrix},$$

where $\mathbf{A} \in \mathbb{S}^k$ is invertible, it holds that

$$\mathbf{X} \succcurlyeq \mathbf{0} \Leftrightarrow \mathbf{A} \succcurlyeq \mathbf{0},\ \mathbf{C} - \mathbf{B}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{B} \succcurlyeq \mathbf{0}. \tag{8.5}$$

Using the Schur complement we can replace the inequalities $\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i \leq t_i,\ t_i \geq 0$ by the following matrix inequality

$$\begin{bmatrix} \mathbf{I}_2 & \mathbf{x}_i \\ \mathbf{x}_i^{\mathrm{T}} & t_i \end{bmatrix} \succcurlyeq \mathbf{0}.$$

Finally, by defining the matrices

$$\mathbf{F_4} \triangleq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \mathbf{F_5} \triangleq \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \mathbf{F_6} \triangleq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \text{ and } \mathbf{G} \triangleq \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

we can reformulate the problem (8.2) as

$$\begin{array}{ll} \underset{\mathbf{x},t_1,t_2}{\text{minimize}} & t_1 + t_2 \\ \text{subject to} & t_1\mathbf{F_4} + x_1\mathbf{F_5} + x_2\mathbf{F_6} + \mathbf{G} \preccurlyeq \mathbf{0} \\ & t_2\mathbf{F_4} + x_2\mathbf{F_5} + x_3\mathbf{F_6} + \mathbf{G} \preccurlyeq \mathbf{0} \\ & x_1\mathbf{F}_1 + x_2\mathbf{F}_2 + x_3\mathbf{F}_3 + \mathbf{C} \preccurlyeq \mathbf{0}, \end{array} \tag{8.6}$$

which is in the desired form of (8.1). Fortunately we do not have to do this conversion all the time, actually we can formulate the problem (8.2) directly in CVX and we do not have to give the form (8.1). So to wrap-up, for us, a SDP will be a convex optimization problem with inequalities with respect to the positive semidefinite cone.

8.2 Duality

To find the dual problem of the problem (8.1) we first write its Lagrangian

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\Lambda}, \boldsymbol{\nu}) &= \mathbf{c}^{\mathrm{T}}\mathbf{x} + \operatorname{tr}((x_1\mathbf{F}_1 + \cdots + x_n\mathbf{F}_n + \mathbf{G})\boldsymbol{\Lambda}) + \boldsymbol{\nu}^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) \\ &= \left(\mathbf{c}^{\mathrm{T}} + \boldsymbol{\nu}^{\mathrm{T}}\mathbf{A} + \begin{bmatrix} \operatorname{tr}(\mathbf{F}_1\boldsymbol{\Lambda}) \\ \vdots \\ \operatorname{tr}(\mathbf{F}_n\boldsymbol{\Lambda}) \end{bmatrix}^{\mathrm{T}} \right) \mathbf{x} - \boldsymbol{\nu}^{\mathrm{T}}\mathbf{b} + \operatorname{tr}(\mathbf{G}\boldsymbol{\Lambda}). \end{aligned} \tag{8.7}$$

We can see that the Lagrangian is a function affine in $\mathbf{x}$. Therefore the dual problem can be written as

$$\begin{array}{ll}\underset{\boldsymbol{\Lambda},\boldsymbol{\nu}}{\text{maximize}} & -\boldsymbol{\nu}^{\mathrm{T}}\mathbf{b} + \operatorname{tr}(\mathbf{G}\boldsymbol{\Lambda}) \\ \text{subject to} & \mathbf{c} + \mathbf{A}^{\mathrm{T}}\boldsymbol{\nu} + \begin{bmatrix}\operatorname{tr}(\mathbf{F}_1\boldsymbol{\Lambda}) \\ \vdots \\ \operatorname{tr}(\mathbf{F}_n\boldsymbol{\Lambda})\end{bmatrix} = \mathbf{0} \\ & \boldsymbol{\Lambda} \succcurlyeq \mathbf{0}.\end{array} \tag{8.8}$$

Note that the dual of an SDP is very similar to the one of an LP as seen in Section 6.2.

8.3 Alternative formulation

Sometimes an SDP is formulated as a so called *standard form SDP* as follows

$$\begin{array}{ll}\underset{\mathbf{X}}{\text{minimize}} & \operatorname{tr}(\mathbf{C}\mathbf{X}) \\ \text{subject to} & \operatorname{tr}(\mathbf{A}_i\mathbf{X}) = b_i, \quad i = 1, \ldots, p \\ & \mathbf{X} \succcurlyeq \mathbf{0},\end{array} \tag{8.9}$$

with $\mathbf{C}, \mathbf{A}_1, \ldots, \mathbf{A}_p \in \mathbb{S}^n$. To understand why this formulation is also valid, just think of it as the dual of the problem (8.1) without equality constraints and with $\mathbf{C} = -\mathbf{G}$, $\mathbf{A}_i = \mathbf{F}_i$ and $b_i = -c_i$.

8.4 Example: portfolio optimization

Assume we have to manage a portfolio of n assets (typically shares). The value of these assets varies over time. At the beginning of a period, we buy/sell some assets, where the amount we buy is given by x_i for the ith asset (x_i is a price in Euros).

If x_i is positive we **buy** units of the ith asset for the amount of x_i Euros. We buy at the current price and at the end of the period, if the asset increased in value, we have a positive return. This a called a *long position.*

If x_i is negative, we call it a *short position.* In this case we **sell** units of the ith asset for the amount of x_i Euros. Someone gives us the assets at the beginning of the period, we sell them for the current price, and at the end of the period we have to buy them back at the new price to give it back to the person who gave us the assets. If the assets loose value we will make a positive return.

Further we define p_i as the relative price change of asset i, i.e.,

$$p_i = \frac{p_i^{\text{end}} - p_i^{\text{start}}}{p_i^{\text{start}}}.$$

The return we have at the end of the period is defined by $\mathbf{r} = \mathbf{p}^{\mathrm{T}}\mathbf{x}$. Now of course we do not know how the prices of the assets will change exactly but we might have knowledge about their statistics. For example, we might now that

$$\mathbb{E}[\mathbf{p}] = \bar{\mathbf{p}} \quad \text{and} \quad \mathbb{E}[(\mathbf{p} - \bar{\mathbf{p}})(\mathbf{p} - \bar{\mathbf{p}})^{\mathrm{T}}] = \boldsymbol{\Sigma}.$$

In that case, we write the return volatility (or the risk) as $\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{x}$. The classical *Markowitz portfolio optimization* problem is the one of minimizing the risk, while guaranteeing a minimum average return with a given budget and only allowing long positions. This problem can be formulated as follows

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{x} \\ \text{subject to} & \bar{\mathbf{p}}^{\mathrm{T}}\mathbf{x} \geq \mathbf{r}_{\min} \\ & \mathbf{1}^{\mathrm{T}}\mathbf{x} = 1, \quad \mathbf{x} \geq \mathbf{0}. \end{array} \tag{8.10}$$

This problem is a QP and it has many variants (for example allowing short positions).

Now assume that we do not know the matrix $\boldsymbol{\Sigma}$ completely but rather we have only partial information about it, for example, we have an upper and a lower bound on each of its entries, i.e.,

$$L_{ij} \leq \Sigma_{ij} \leq U_{ij}, \quad i,j = 1,\ldots,n.$$

The question is the following. Given a specific strategy $\mathbf{x}$, what is the maximum risk that we take over all possible covariance matrices that fulfill the above inequalities? In other words, what is the worst case covariance matrix for our strategy? We can answer this question by solving the following SDP

$$\begin{array}{ll} \underset{\boldsymbol{\Sigma}}{\text{maximize}} & \mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{x} \\ \text{subject to} & L_{ij} \leq \Sigma_{ij} \leq U_{ij}, \quad i,j = 1,\ldots,n \\ & \boldsymbol{\Sigma} \succcurlyeq \mathbf{0}. \end{array} \tag{8.11}$$

The optimization variable is $\boldsymbol{\Sigma}$ and the constraint $\boldsymbol{\Sigma} \succcurlyeq \mathbf{0}$ represents the fact that $\boldsymbol{\Sigma}$ is a covariance matrix.

8.5 Problem section

1. ***Sensor selection.***

   This problem is inspired by the work [10]. We want to estimate a vector $\mathbf{x} \in \mathbb{R}^n$ using $m > n$ possible sensors performing linear measurements. The measurement vector is $\mathbf{y} \in \mathbb{R}^m$ given by

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n},$$

   where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is our measurement matrix and $\mathbf{n} \in \mathbb{R}^m$ is a noise vector. The components of $\mathbf{n}$ are independently and identically-distributed at random from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$. So, each sensor i performs a measurement $y_i = \mathbf{a}_i^{\mathrm{T}}\mathbf{x} + n_i$, where $\mathbf{a}_i$ is the i-th column of $\mathbf{A}^{\mathrm{T}}$.

   Provided that $\mathbf{A}$ has full column rank, the maximum likelihood (ML) estimate of $\mathbf{x}$ is

$$\hat{\mathbf{x}} = \mathbf{A}^{\dagger}\mathbf{y} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y}.$$

   The estimation error is denoted by

$$\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}.$$

   Note that $\mathbf{e}$ is a Gaussian random variable. The mean of the error $\mathbf{e}$ is

$$\begin{aligned} \mu = \mathbb{E}[\mathbf{e}] &= \mathbb{E}[(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y} - \mathbf{x}] \\ &= \mathbb{E}[(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}(\mathbf{A}\mathbf{x} + \mathbf{n}) - \mathbf{x}] \\ &= \mathbb{E}[(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{n}] \\ &= (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbb{E}[\mathbf{n}] = \mathbf{0}. \end{aligned}$$

   and its covariance matrix is

$$\begin{aligned} \boldsymbol{\Sigma} = \mathbb{E}[\mathbf{e}\mathbf{e}^{\mathrm{T}}] &= \mathbb{E}[((\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y} - \mathbf{x})((\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y} - \mathbf{x})^{\mathrm{T}}] \\ &= \mathbb{E}[((\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{n})((\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{n})^{\mathrm{T}}] \\ &= \mathbb{E}[(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{n}\mathbf{n}^{\mathrm{T}}\mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}] \\ &= (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbb{E}[\mathbf{n}\mathbf{n}^{\mathrm{T}}]\mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1} \\ &= (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}(\sigma^2\mathbf{I}_m)\mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1} \\ &= \sigma^2(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}. \end{aligned}$$

   The *η-confidence set* $\mathcal{S}$ for a random variable $\mathbf{z}$ is the minimum volume set, such that $\Pr(\mathbf{z} \in \mathcal{S}) \geq \eta$. For example if $\eta = 0.99$, we can say that $\mathbf{z} \in \mathcal{S}$ with overwhelming probability. For a Gaussian random variable, the η-confidence set coincides with the *η-confidence ellipsoid*, i.e., the minimum volume ellipsoid that contains $\mathbf{z}$ with probability η. It is given by

$$\mathcal{E}_\alpha = \{\mathbf{z} \mid \mathbf{z}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\mathbf{z} \leq \alpha\},$$

   where $\alpha = F_{\chi_n^2}^{-1}(\eta)$ in which $F_{\chi_n^2}(\eta)$ is the cumulative distribution function of a chi-squared random variable with n degrees of freedom.

Since $\mathbf{e}$ is Gaussian, one interesting measure of the quality of our estimation is the volume of $\mathcal{E}_\alpha$ for $\mathbf{e}$. Generally, if this volume is small, then with probability η we make a small error. For convenience we will work with the log of the volume, which is given by

$$\log \text{vol}(\mathcal{E}_\alpha) = \log\left(\frac{(\alpha\pi)^{n/2}\sigma}{\Gamma(n/3)}\right) - (1/2)\log\det(\mathbf{A}^{\mathrm{T}}\mathbf{A}),$$

where Γ is the gamma function.

The *sensor selection problem* is the one of choosing, among our m potential measurements, only $k < m$ measurements and minimize the log volume of the confidence ellipsoid. In other words, we have access to m sensors but we only want to use few of them (e.g. to reduce cost or energy consumption), the question is which one should we keep active and still guarantee a good measurement quality. We can formulate this problem as the following SDP

$$\begin{array}{ll} \underset{\mathcal{S}}{\text{maximize}} & \log\det\left(\sum_{i\in\mathcal{S}} \mathbf{a}_i\mathbf{a}_i^{\mathrm{T}}\right) \\ \text{subject to} & |\mathcal{S}| = k, \end{array} \tag{8.12}$$

where $\mathcal{S} \subseteq 1, \ldots, m$ is a set of indices and $|\mathcal{S}|$ denotes the cardinality of $\mathcal{S}$. To understand the above formulation, note that

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \sum_{i=1}^{m} \mathbf{a}_i\mathbf{a}_i^{\mathrm{T}}.$$

Note also, that part of the log volume is constant and thus doesn't have to be included in the objective of the optimization problem. With this problem we want to choose a set of k indices corresponding to k measurements that minimizes the log volume of the confidence ellipsoid.

Note: Throughout this exercise, the MATLAB functions `sortrows()` and `sort()` will come in handy.

(a) Before directly solving the sensor selection problem, we want to try simple heuristics that may work very well in practice. The first is the so called *lazy approach*, which consists in simply taking the first k sensors. Load the file `sensor_selection.mat` and implement this simple method. Take as parameters $m = 50, n = 10, \sigma = 1$ and $\eta = 0.95$. Plot the log volume of the confidence ellipsoid for k varying from 10 to 30. Use `chi2inv()` to implement the mapping $\alpha = F^{-1}_{\chi^2_n}(\eta)$.

(b) A second approach, which is a bit more involved, is called the *SNR approach* and consists in selecting the k measurements that have the maximum average

SNRs. The average SNR of the ith sensor is given by

$$SNR_i = \frac{\mathbb{E}[|\mathbf{a}_i^{\mathrm{T}}\mathbf{x}|^2]}{\mathbb{E}[|n_i|^2]}.$$

Assume that $\mathbf{x}$ is i.i.d. and that its components have zero mean and variance one, then

$$SNR_i = \frac{\mathbf{a}_i^{\mathrm{T}}\mathbf{a}_i}{\sigma^2}.$$

Implement the SNR approach and plot the log volume of the confidence ellipsoid for k varying from 10 to 30.

(c) Compare these two approaches. Can you explain what you see?

(d) Now it is time to try to solve the problem (8.12) directly. We can reformulate this problem as follows

$$\begin{array}{ll} \underset{\mathbf{z}}{\text{maximize}} & \log\det\left(\sum_{i=1}^{m} z_i\mathbf{a}_i\mathbf{a}_i^{\mathrm{T}}\right) \\ \text{subject to} & \mathbf{1}^{\mathrm{T}}\mathbf{z} = k, \\ & z_i \in \{0,1\}, \quad i = 1,\dots,m, \end{array} \tag{8.13}$$

which is a Boolean problem and therefore not convex. To render this problem convex we relax the Boolean constraint to get the following convex SDP

$$\begin{array}{ll} \underset{\mathbf{z}}{\text{maximize}} & \log\det\left(\sum_{i=1}^{m} z_i\mathbf{a}_i\mathbf{a}_i^{\mathrm{T}}\right) \\ \text{subject to} & \mathbf{1}^{\mathrm{T}}\mathbf{z} = k, \\ & 0 \le z_i \le 1, \quad i = 1,\dots,m, \end{array} \tag{8.14}$$

We call $\mathbf{z}_L$ the solution of the problem (8.14) and $L(k)$ the optimal value of the problem. $L(k)$ is an upper bound for the problem (8.13) and therefore a lower bound on the log volume of the confidence ellipsoid. Of course $\mathbf{z}_L$ is not a feasible solution for (8.13) since it might have fractional components. To find a feasible solution from $\mathbf{z}_L$, we simply take the k largest components of $\mathbf{z}_L$, set them to 1 and set all other components to 0. We call this heuristic the *SDP heuristic*. We also call the new sensor selection $\mathbf{z}_U$ and define $U(k) = \log\det\left(\sum_{i=1}^{m} z_{Ui}\mathbf{a}_i\mathbf{a}_i^{\mathrm{T}}\right)$. $U(k)$ is a lower bound for the problem (8.13) and therefore an upper bound on the log volume of the confidence ellipsoid. We can measure how good the SDP heuristic is by calculating $L(k) - U(k)$.

Solve the relaxed problem and implement the SDP heuristic. Plot the log volume of the confidence ellipsoid for k varying from 10 to 30 for the relaxed problem and the SDP heuristic. How does this new heuristic compare to the two former ones?

2. ***Distributed spectrum sensing : compressed sensing vs. matrix completion.*** **[bonus exercise]**

Bandwidth scarcity has become one of the most important problems in modern communication systems. One of the most promising approach to tackle this problem is to reuse the underutilized bandwidth of a primary system. This approach is called dynamic spectrum access (DSA) and is performed by intelligent transceivers called cognitive radios (CRs). The main technology that enables DSA is spectrum sensing, the goal of which is to sense spectrum occupancy in a given bandwidth. By knowing the spectrum occupancy, CRs can reuse the spectrum which is left unoccupied by a primary user. Spectrum sensing of a potentially large bandwidth is very costly when sampling at the Nyquist rate and requires expensive hardware components. In order to counterbalance this drawback, it might be interesting to sample at a rate lower than the Nyquist rate.

Formally, we want to sense a frequency signal $\mathbf{f} \in \mathbb{R}^{n_1}$, coming from a primary user, with $m < n_1$ measurements. The underutilization of the spectrum is modeled by the fact that $\mathbf{f}$ is k-sparse, i.e., it has at most k nonzero components. In our system, we have n_2 CRs available to sense the spectrum. Each CR i receives a signal $\hat{\mathbf{f}}_i$ given by

$$\hat{\mathbf{f}}_i = \mathbf{H}_i \mathbf{f} + \mathbf{n}_i, \quad i = 1, \ldots, n_2,$$

where $\mathbf{H}_i \in \mathbb{R}^{n_1 \times n_1}$ is a diagonal channel matrix and $\mathbf{n}_i \in \mathbb{R}^{n_1}$ is a Gaussian noise vector with mean $\mathbf{0}$ and covariance matrix $\mathbf{C_n} = \sigma^2 \mathbf{I}_{n_1}$. A very important point here is that the channel matrices $\mathbf{H}_i$ are not known at the CRs. Each CR i takes m measurements of $\hat{\mathbf{f}}_i$ and sends its measurement vector to a fusion center, which then has to decide which subchannels (i.e., which component of $\mathbf{f}$) are occupied or not. Next, we present two methods to perform this task with $m < n_1$.

Compressed sensing

Each CR i takes m measurements of $\hat{\mathbf{f}}_i$ in time of the form

$$\hat{\mathbf{t}}_i = \mathbf{\Phi}_i \mathbf{\Psi} \hat{\mathbf{f}}_i, \quad i = 1, \ldots, n_2,$$

where $\mathbf{\Psi} \in \mathbb{C}^{n_1 \times n_1}$ is an IDFT matrix and $\mathbf{\Phi}_i \in \mathbb{R}^{m \times n_1}$ is a measurement matrix where each row contains $'0'$s everywhere and a single $'1'$ when a component of $\mathbf{\Psi}\hat{\mathbf{f}}_i$ is sampled. For example if a row of $\mathbf{\Phi}_i$ is $[1\ 0 \ldots 0]$ it means we sampled the first element of $\mathbf{\Psi}\hat{\mathbf{f}}_i$. Which elements of $\mathbf{\Psi}\hat{\mathbf{f}}_i$ are sampled can be chosen at random. Each CR i reconstructs $\mathbf{H}_i\mathbf{f}$ using compressed sensing as follows

$$\begin{array}{ll} \underset{\mathbf{f_r}_i}{\text{minimize}} & \|\mathbf{f_r}_i\|_1 \\ \text{subject to} & \|\mathbf{\Phi}_i \mathbf{\Psi} \mathbf{f_r}_i - \hat{\mathbf{t}}_i\|_2 \leq \epsilon, \end{array} \tag{8.15}$$

where ϵ is fixed and should be chosen carefully (possibly depending on the noise power). After that, all the $\mathbf{f_r}_i$ are collected at a fusion center. Depending on the average power detected by all CRs on a subchannel j, given by

$$(1/n_2) \sum_{i=1}^{n_2} \mathbf{f_r}_i^2(j)$$

the fusion center must decide if a subchannel is occupied or not, for example by comparing this value with a threshold P_{th}, i.e., if $(1/n_2)\sum_{i=1}^{n_2} \mathbf{f_r}_i^2(j) \geq P_{th}$ then the subchannel is busy. Of course P_{th} must also be chosen carefully. You can read more about this approach in, e.g., [16].

Matrix completion

Matrix completion can be thought as the continuation of compressed sensing for matrices. The basic idea is that we can reconstruct a matrix $\mathbf{X} \in \mathbf{R}^{n_1 \times n_2}$ with much fewer samples than $n_1 \times n_2$, if $\mathbf{X}$ has a low rank. The method goes like this. We sample M entries of $\mathbf{X}$ chosen at random. We call Ω the set of indices sampled entries and P_Ω the sampling operator, defined as

$$\begin{array}{ll} P_\Omega(\mathbf{X})_{ij} = X_{ij} & \text{if } (i,j) \in \Omega, \\ P_\Omega(\mathbf{X})_{ij} = 0 & \text{otherwise.} \end{array}$$

The result of the sampling is given by $P_\Omega(\mathbf{X})$. The important thing to understand is that $|\Omega| = M < n_1 \times n_2$. If $\mathbf{X}$ has a low rank, we can reconstruct it by solving the following SDP

$$\begin{array}{ll} \underset{\mathbf{X_r}}{\text{minimize}} & \|\mathbf{X_r}\|_* \\ \text{subject to} & P_\Omega(\mathbf{X_r}) = P_\Omega(\mathbf{X}), \end{array} \tag{8.16}$$

where $\|\mathbf{X_r}\|_*$ is the *nuclear norm* of $\mathbf{X_r}$, i.e., the sum of its singular values. The nuclear norm is actually the best convex approximation of the rank function (similar to compressed sensing, where the ℓ_1 norm is the best convex approximation of the cardinality function).

Now we want to apply this theory to spectrum sensing. Each CR i takes m measurements of $\hat{\mathbf{f}}_i$ in time of the form

$$\hat{\mathbf{t}}_i = \mathbf{\Phi}_i \mathbf{\Psi} \hat{\mathbf{f}}_i, \quad i = 1, \ldots, n_2,$$

similar to the compressed sensing methodology. In that case we have $M = n_2 m$. The CRs do not try to reconstruct the spectrum but directly send their $\hat{\mathbf{t}}_i$ to the fusion center. The fusion center forms a matrix $\hat{\mathbf{T}} = [\hat{\mathbf{t}}_1 \ldots \hat{\mathbf{t}}_{n_2}]$. It then tries to reconstruct the matrix $[\mathbf{H}_1\mathbf{f} \ldots \mathbf{H}_{n_2}\mathbf{f}]$ by solving the following SDP

$$\begin{array}{ll} \underset{\mathbf{F_r}}{\text{minimize}} & \|\mathbf{F_r}\|_* \\ \text{subject to} & \|P_\Omega(\mathbf{\Psi F_r}) - \hat{\mathbf{T}}\|_\text{F} \leq \epsilon, \end{array} \tag{8.17}$$

where P_Ω is, in that case, determined by the matrices $\mathbf{\Phi}_i$ and ϵ is fixed and should be chosen carefully. We have $\mathbf{F_r} = [\mathbf{f_r}_1 \ldots \mathbf{f_r}_{n_2}]$ and we can now, using a similar method as for compressed sensing, determine if a subchannel is busy or not. You can read more about this approach in, e.g., [14].

Problem

Use the file `cs_vs_mc.m`. Implement and compare the two approaches. You are free to decide which one is the most sensible. What is the meaning of the ROC?