CONVEX OPTIMIZATION

by

MARTIN LOTZ
School of Mathematics
The University of Manchester

MATH36061
Semester 1, 2016/17
December 2016

Contents

Part I

Introduction

Lecture 1

"[N]othing at all takes place in the universe in which some rule of maximum or minimum does not appear."

— Leonhard Euler

Mathematical optimization, traditionally also known as mathematical programming, is the theory of optimal decision making. Optimization problems arise in a large variety of contexts, including scheduling and logistics problems, finance, optimal control, signal processing, and machine learning. The underlying mathematical problem always amounts to finding parameters that minimize[1] (cost) or maximize (utility) an objective function in the presence or absence of a set of constraints. An important special case is the class of *convex optimization* problems. Such problems will be the main focus of this course.

1.1 What is an optimization problem?

A general mathematical optimization problem is a problem of the form

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{x} \in \Omega \end{aligned} \tag{1.1}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ are real-valued **objective function** and $\Omega \subseteq \mathbb{R}^n$ is a set defining the **constraints**. Among all $\boldsymbol{x} \in \Omega$, we seek one with smallest f-value. Typically, the constraint set Ω will consist of such $\boldsymbol{x} \in \mathbb{R}^n$ that satisfy certain equations and inequalities,

$$f_1(\boldsymbol{x}) \leq 0, \ldots, f_m(\boldsymbol{x}) \leq 0, g_1(\boldsymbol{x}) = 0, \ldots, g_p(\boldsymbol{x}) = 0.$$

A vector $\boldsymbol{x}^*$ satisfying the constraints is called an *optimum*, a *solution*, or a *minimizer* of the problem (1.1), if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all other $\boldsymbol{x}$ that satisfy the constraints. Note that replacing f by $-f$, we could equivalently state the problem as a maximization problem. In this course we are mostly concerned with functions and constraint sets that are **convex**.

- A set $C \subseteq \mathbb{R}^n$ is **convex**, if for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda \in [0, 1]$, $\lambda\boldsymbol{x} + (1 - \lambda)\boldsymbol{y} \in C$. That is, for any two points in C, the line segment connecting them is also in C.

[1] For the sake of consistency with most of the literature, throughout these notes we use the American spelling of minimizer and maximizer with "z" instead of "s".

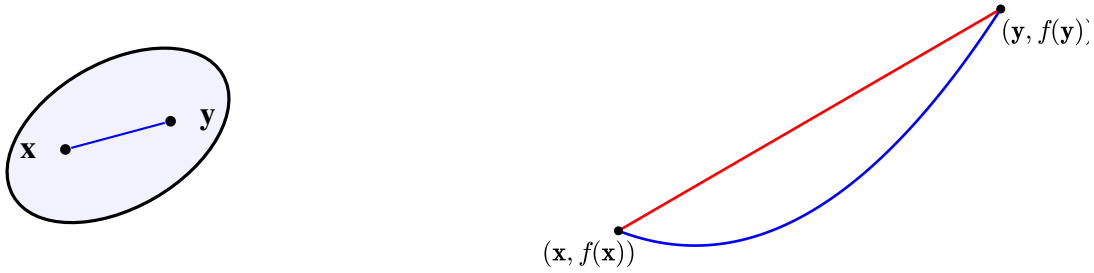


Figure 1.1: A convex set and a convex function

- A function $f\colon C \to \mathbb{R}$ is convex, if C is convex and for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda \in [0,1]$, $f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$.

A **convex optimization** problem is one where the set of constraints Ω and the function f are convex. While most general optimization problems are practically intractable, convex optimization problems can be solved efficiently, and still cover a surprisingly large range of applications!

1.2 Examples of optimization problem

Countless problems from science and engineering can be cast as optimization problems. We present a few first examples, many more will follow in the course of this lecture. The examples below come with associated Python code. At this moment it is not expected that you understand them in detail, they are merely intended to illustrate some of the problems that convex optimization deals with, and how they can be solved.

Example 1.1. Suppose we want to understand the relationship of a quantity Y (for example, sales data) to a series of *predictors* $X_1, \ldots, X_p$ (for example, advertising budget in different media). We can often assume the relationship to be *approximately linear*,

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon, \tag{1.2}$$

where ε is some error or noise term. The goal is to determine the *model parameters* $\beta_0, \ldots, \beta_p$. To determine these, we can collect $n \geq p$ sample realizations (from observations or experiments),

$$Y = y_i, \quad X_1 = x_{i1}, \ldots, X_p = x_{ip}, \quad 1 \leq i \leq n,$$

and assume that the data is related according to (1.2),

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i, \quad 1 \leq i \leq n.$$

Collecting the data in matrices and vectors,

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad \boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix},$$

we can write the relationship concisely as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

We would then like to find $\boldsymbol{\beta}$ in such a way that the difference $\boldsymbol{\varepsilon} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}$ is as *small* as possible. One way of measuring the size of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is the square of its 2-norm, or Euclidean norm,

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

The best $\boldsymbol{\beta}$ is then the vector that solves the unconstrained optimization problem

$$\text{minimize} \quad \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2.$$

This is an example of an optimization problem, with variables $\boldsymbol{\beta}$, no constraints (*all* β are valid candidates and the constraint set is $\Omega = \mathbb{R}^{p+1}$), and a *quadratic* objective function

$$f(\boldsymbol{\beta}) = \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2 = (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})^\top (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) = \boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} - 2\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{y}^\top \boldsymbol{y},$$

where $\boldsymbol{X}^\top$ is the matrix transpose. As we will see later, quadratic functions are convex, so this is a convex optimization problem. This simple optimization problem has a **unique closed form solution**,

$$\boldsymbol{\beta}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{1.3}$$

In practice one wouldn't compute $\boldsymbol{\beta}^*$ by evaluating [1], as there are more efficient methods available (see Lecture 2).

To illustrate the least squares setting using a concrete example, assume that we have data relating the basal metabolic rate (energy expenditure per time unit) in mammals to their mass.[2] The model we use is $Y = \beta_0 + \beta_1 X$, with Y the basal metabolic rate and X the mass. Using

[2]This example is from the episode "Size Matters" of the BBC series Wonders of Life.

data for 573 mammals from the PanTHERIA database[3], we can assemble the vector $\boldsymbol{y}$ and the matrix $\boldsymbol{X} \in \mathbb{R}^{n\times(p+1)}$ in order to compute the $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$. Here, $p = 1$ and $n = 573$.

We next illustrate how to solve this problem in Python. As usual, we first have to import some relevant libraries: **numpy** for numerical computation, **pandas** for loading and transforming datasets, **cvxpy** for convex optimization, and **matplotlib** for plotting.

In [1]:
```
# Import some important Python modules
import numpy as np
import pandas as pd
from cvxpy import *
import matplotlib.pyplot as plt
```

We next have to load the data. The data is saved in a table with 573 rows and 2 columns, where the first column list the mass and the second the basal metabolic rate.

In [2]:
```
# Load data into numpy array
bmr = pd.read_csv('../../data/bmr.csv',header=None).as_matrix()
# We can find out the dimension of the data
bmr.shape
```

Out [2]: (573, 2)

To see the first three and the last three rows of the dataset, we can use the "print" command.

In [3]:
```
print(bmr[0:3,:])
```

```
[[ 13.108   10.604 ]
 [  9.3918   8.2158]
 [ 10.366    9.3285]]
```

To visualise the whole dataset, we can make a scatterplot by interpreting each row as a coordinate on the plane, and marking it with a dot.

In [4]:
```
# Display scatterplot of data (plot all the rows as points)
% matplotlib inline
bmr1 = plt.plot(bmr[:,0],bmr[:,1],'o')
plt.xlabel("Mass")
plt.ylabel("Basal metabolic rate")
plt.show()
```

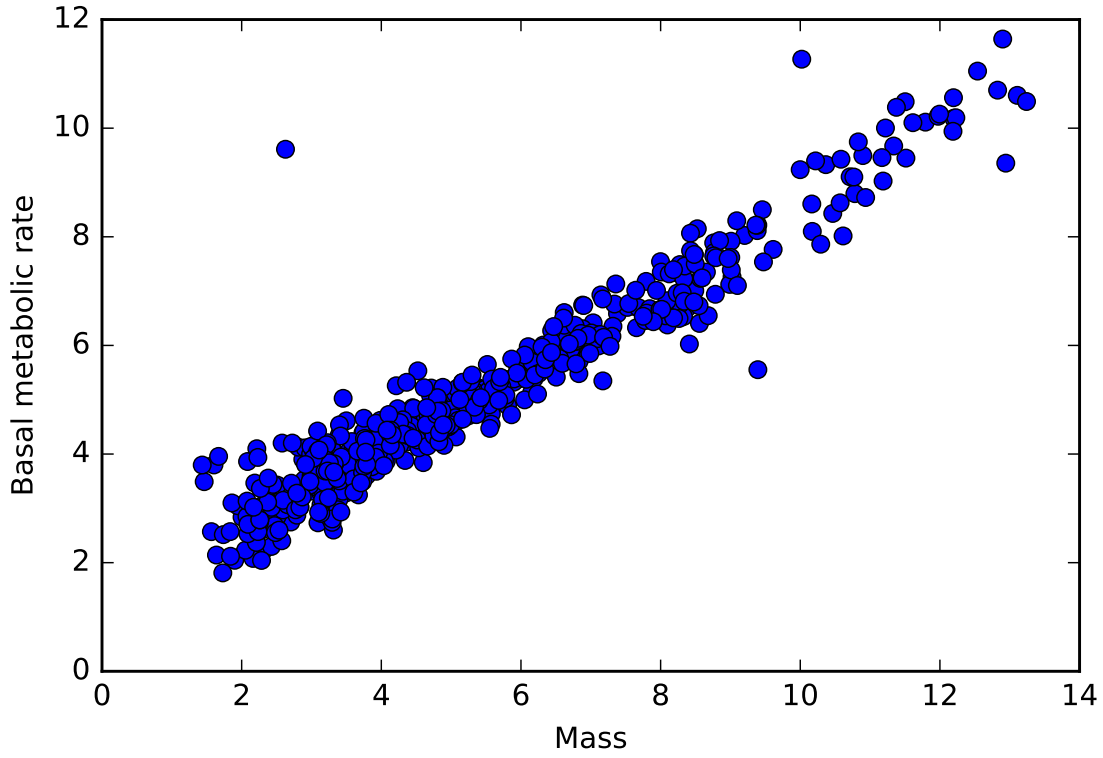
The plot above suggests that the relation of the basal metabolic rate to the mass is linear, i.e., of the form

$$Y = \beta_0 + \beta_1 X,$$

where X is the mass and Y the BMR. We can find β_0 and β_1 by solving an optimization problem as described above. We first have to assemble the matrix $\boldsymbol{X}$ and the vector $\boldsymbol{y}$.

In [5]:
```
n = bmr.shape[0]
p = 1
X = np.concatenate((np.ones((n,1)),bmr[:,0:p]),axis=1)
y = bmr[:,-1]
```

[3] http://esapubs.org/archive/ecol/E090/184/#data



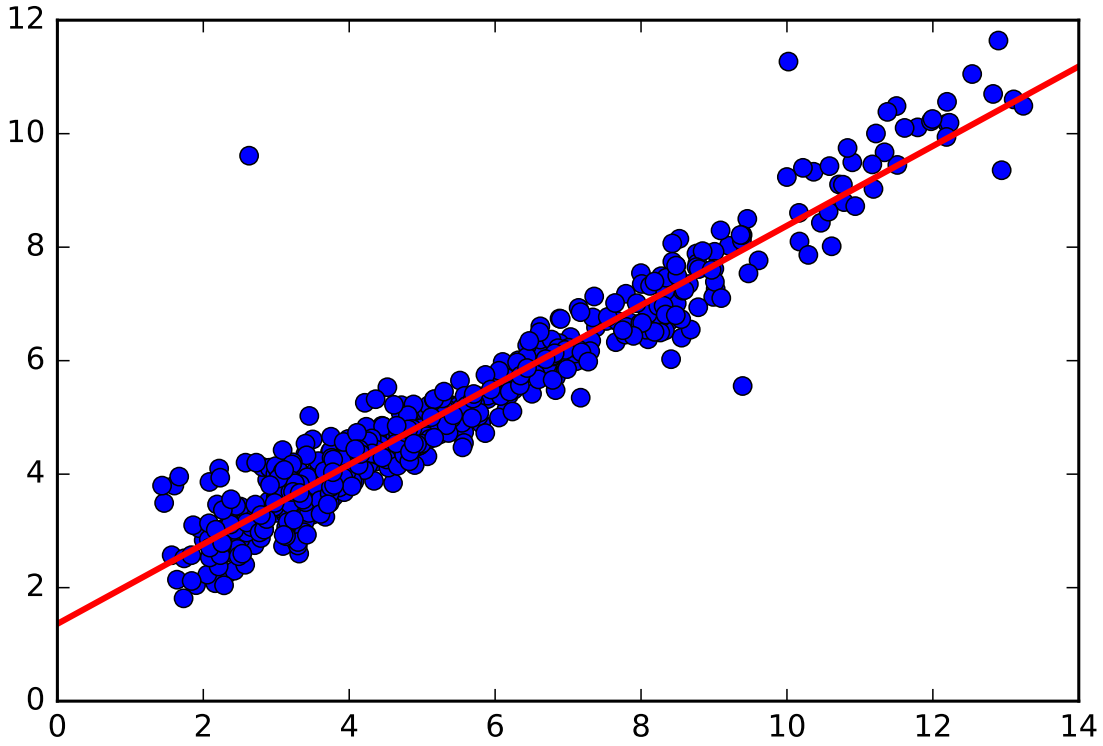
In [6]:
```
# Create a (p+1) vector of variables
Beta = Variable(p+1)

# Create sum-of-squares objective function
objective = Minimize(sum_entries(square(X*Beta - y)))

# Create problem and solve it
prob = Problem(objective)
prob.solve()

print("status: ", prob.status)
print("optimal value: ", prob.value)
print("optimal variables: ", Beta[0].value, Beta[1].value)
```

```
status:  optimal
optimal value:  152.736200529558
optimal variables:  1.3620698558275837 0.7016170245505547
```

Now that we solved the problem and have the values $\beta_0 = 1.362$ and $\beta_1 = 0.702$, we can plot the line and see how it fits the data.

In [6]:
```
plt.plot(bmr[:,0],bmr[:,1],'o')

xx = np.linspace(0,14,100)
bmr = plt.plot(xx, Beta[0].value+Beta[1].value*xx, color='red',\
 linewidth=2)
plt.show()
```

Even though for illustration purposes we used the CVXPY package, this particular problem can be solved directly using the least squares solver in numpy.



In [7]:
```
import numpy.linalg as la
beta = la.lstsq(X,y)
print(beta[0])
```

```
[ 1.36206997  0.70161692]
```

The above example is an example of a **machine learning** problem. In machine learning, one seeks to *learn* a function F mapping some inputs X to outputs Y, $Y = F(X)$. A few examples:

- X: economic data, Y: value of a stock;
- X: physiological data, Y: medical diagnosis;
- X: email, Y: 1 if email is span, 0 otherwise;
- X: scanned image, Y: a letter represented by that image.

In *supervised learning* we have a set of sample input pairs, (y_i, x_i), $1 \leq i \leq m$, and we typically try to find a function F that minimizes the **least squared error**,

$$\text{minimize} \quad \sum_{i=1}^{m} (\boldsymbol{y}_i - F(\boldsymbol{x}_i))^2,$$

where one minimizes over all functions F from some class. In the above example, we assumed our functions to be linear, in which case the can by parametrized by the coefficients $\beta_0, \ldots, \beta_p$. As the course progresses, we will see examples of more sophisticated machine

learning problems, often with nonlinear objective function and other *loss functions* instead of the least square error.

Example 1.2. (Linear programming) Suppose a plane has two cargo compartments with weight capacities $C_1 = 35$ and $C_2 = 40$ tonnes, and volumes (space capacities) $V_1 = 250$ and $V_2 = 400$ cubic metres. Assume we have three types of cargo to transport, specified as follows.

	Volume (m^3 per tonne)	Weight (tonnes)	Profit (£/ tonne)
Cargo 1	8	25	£300
Cargo 2	10	32	£350
Cargo 3	7	28	£270

The problem is now to decide how much of each cargo to take on board, and how to distribute it in an optimal way among the two compartments.

1. The *decision variables* x_{ij} specify the amount, in tonnes, of cargo i to go into compartment j. We collect them in a vector $\boldsymbol{x}$.

2. The *objective function* is the total profit,

$$f(\boldsymbol{x}) = 300 \cdot (x_{11} + x_{12}) + 350 \cdot (x_{21} + x_{22}) + 270 \cdot (x_{31} + x_{32}).$$

3. The *constraints* are given by the space and weight limitations of the compartments, and the amount of cargo available.

$$\begin{aligned}
x_{11} + x_{12} &\leq 25 && \text{(total amount of cargo 1)}\\
x_{21} + x_{22} &\leq 32 && \text{(total amount of cargo 2)}\\
x_{31} + x_{32} &\leq 28 && \text{(total amount of cargo 3)}\\
x_{11} + x_{21} + x_{31} &\leq 35 && \text{(weight constraint on compartment 1)}\\
x_{12} + x_{22} + x_{32} &\leq 40 && \text{(weight constraint on compartment 2)}\\
8x_{11} + 10x_{21} + 7x_{31} &\leq 250 && \text{(volume constraint on compartment 1)}\\
8x_{12} + 10x_{22} + 7x_{32} &\leq 400 && \text{(volume constraint on compartment 2)}\\
(x_{11} + x_{21} + x_{31})/35 - (x_{12} + x_{22} + x_{32})/40 &= 0 && \text{(maintain balance of weight ratio)}\\
x_{ij} &\geq 0 && \text{(cargo can't have negative weight)}
\end{aligned}$$

It is customary to write the objective function as a scalar product, $f(\boldsymbol{x}) = \langle \boldsymbol{c}, \boldsymbol{x} \rangle := \boldsymbol{c}^\top \boldsymbol{x}$, and to express the constraints as systems of linear equations and inequalities using matrix-vector products,

$$\begin{aligned}
\text{maximize} \quad & \langle \boldsymbol{c}, \boldsymbol{x} \rangle\\
\text{subject to} \quad & A\boldsymbol{x} \leq \boldsymbol{b}\\
& B\boldsymbol{x} = \boldsymbol{d}\\
& \boldsymbol{x} \geq 0
\end{aligned}$$

where the inequalities $\geq$ and $\leq$ are to be understood componentwise. This problem has a unique solution that can be found using CVXPY in Python,

In [8]:

```
\\# Define all the matrices and vectors involved
c = np.array([300,300,350,350,270,270])
A = np.array([[1, 1, 0, 0, 0, 0],
              [0, 0, 1, 1, 0, 0],
              [0, 0, 0, 0, 1, 1],
              [1, 0, 1, 0, 1, 0],
              [0, 1, 0, 1, 0, 1],
              [8, 0, 10, 0, 7, 0],
              [0, 8, 0, 10, 0, 7]])
b = np.array([25,32,28,35,40,250,400]);
B = np.array([1/35, -1/40, 1/35, -1/40, 1/35, -1/40]);
d = np.zeros(1)

# Create variables, objective and constraints
x = Variable(6)
constraints = [A*x <= b, B*x == d, x >= 0]
objective = Maximize(c*x)

# Create a problem using the objective and constraints and solve it
prob = Problem(objective, constraints)
prob.solve()

print("Solution found: \n", np.round(np.abs(x.value), \
  decimals=2).transpose())
```

```
Solution found:
 [[  6.75   7.71   0.    32.    28.     0.  ]]
```

The solution found is

$$x_{11} = 6.7500, x_{12} = 7.7143, x_{21} = 0, x_{22} = 32, x_{31} = 28, x_{32} = 0.$$

We made some simplifying assumptions, for example that the cargo can be split up into arbitrary fractions. Additional work is required to resolve these issues. Problems of this kind are known as **linear programming**, because the objective function and the constraints are given by linear functions. Such problems can be solved efficiently using the simplex algorithm or interior point methods. The highly developed theory of linear programming acts as a template for more general convex optimization that is developed in this course.

Example 1.3. (Image inpainting) Optimization methods play an increasingly important role in image and signal processing. An image can be viewed as an $m \times n$ matrix $\boldsymbol{U}$, with each entry u_{ij} corresponding to a light intensity (for greyscale images), or a colour vector, represented by a triple of red, green and blue intensities (usually with values between 0 and 255 each). For simplicity the following discussion assumes a greyscale image. For compututational pursposes, the matrix of an image is often viewed as an mn-dimensional vector $\boldsymbol{u}$, with the columns of the matrix stacked on top of each other.

In the *image inpainting* problem, one aims to *guess* the true value of missing or corrupted entries of an image. There are different approaches to this problem. A conceptually simple approach is to replace the image with the *closest* image among a set of images satisfying typical properties. But what are typical properties of a typical image? Some properties that come to mind are:

- Images tend to have large homogeneous areas in which the colour doesn't change much;

- Images have approximately low rank, when interpreted as matrices.

Total variation image analysis takes advantage of the first property. The **total variation** or TV-norm is the sum of the norm of the horizontal and vertical differences,

$$\|\boldsymbol{U}\|_{\mathrm{TV}} = \sum_{i=1}^{m}\sum_{j=1}^{n}\sqrt{(u_{i+1,j}-u_{i,j})^2 + (u_{i,j+1}-u_{i,j})^2},$$

where we set entries with out-of-bounds indices to 0. The TV-norm naturally increases with increased variation or sharp edges in an image. Consider for example the two following matrices (imagine that they represent a 3×3 pixel block taken from an image).

$$\boldsymbol{U}_1 = \begin{pmatrix} 0 & 17 & 3 \\ 7 & 32 & 0 \\ 2 & 9 & 27 \end{pmatrix}, \quad \boldsymbol{U}_2 \begin{pmatrix} 1 & 1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

The left matrix has TV-norm $\|\boldsymbol{U}_1\|_{\mathrm{TV}} = 200.637$, while the right one has TV-norm $\|\boldsymbol{U}_2\|_{\mathrm{TV}} = 14.721$ (verify this!) Intuitively, we would expect a natural image with artifacts added to it to have a higher TV norm.

Now let $\boldsymbol{U}$ be an image with entries u_{ij}, and let $\Omega \subset [m] \times [n] = \{(i,j) : 1 \leq i \leq m, 1 \leq j \leq n\}$ be the set of indices where the original image and the corrupted image coincide (all the other entries are missing). One could attempt to find the image with the *smallest* TV-norm that coincides with the knwon pixels u_{ij} for $(i,j) \in \Omega$. This is an optimization problem of the form

$$\text{minimize} \quad \|\boldsymbol{X}\|_{\mathrm{TV}} \quad \text{subject to} \quad x_{ij} = u_{ij} \text{ for } (i,j) \in \Omega.$$

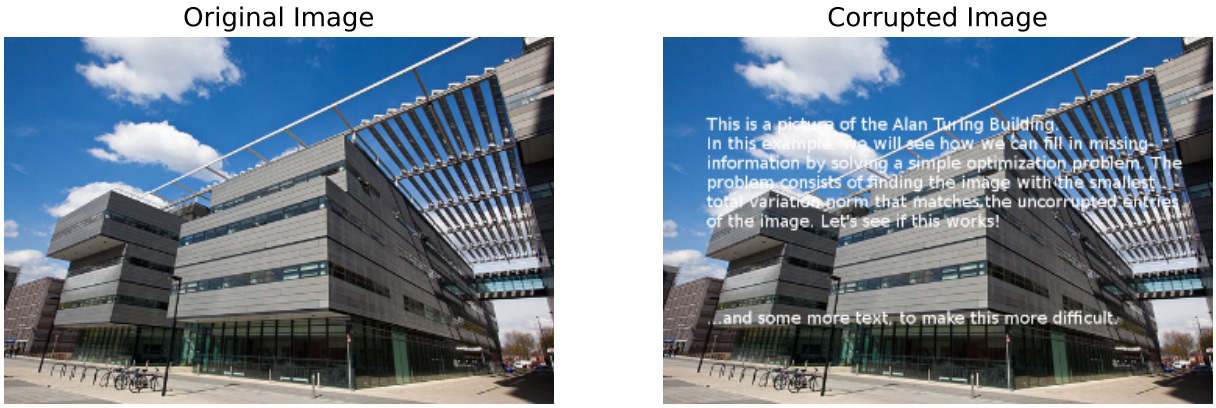
The TV-norm is an example of a convex function and the constraints are linear conditions which define a convex set. This is again an example of a **convex optimization problem** and can be solved efficiently by a range of algorithms. For the time being we will not go into the algorithms but solve it using CVXPY. The example below is based on an example from the CVXPY Tutorial[4], and it is recommended to look at this tutorial for other interesting examples!

Warning: the example below uses some more advanced Python programming, it is not necessary to understand the details at this point.

In our first piece of code below, we load the image and a version of the image with text written on it, and display the images. The **Python Image Library (PIL)** is used for this purpose.

[4] http://www.cvxpy.org/en/latest/tutorial/index.html

In [9]:
```
from PIL import Image

# Load the images and convert to numpy arrays for processing.
U = np.array(Image.open("../images/alanturing.png"))
Ucorr = np.array(Image.open("../images/alanturing-corr.png"))

# Display the images
%matplotlib inline
fig, ax = plt.subplots(1, 2,figsize=(10, 5))

ax[0].imshow(U);
ax[0].set_title("Original Image")
ax[0].axis('off')

ax[1].imshow(Ucorr);
ax[1].set_title("Corrupted Image")
ax[1].axis('off');
```



After having the images at our disposal, we determine which entries of the corrupted image are known. We store these in a *mask* M, with entries $m_{ijk} = 1$ if the colour k of the (i, j)-th pixel is known, and 0 otherwise.

In [10]:
```
# Each image is now an m x n x 3 array, with each pixel
# represented by three numbers between 0 and 255,
# corresponding to red, green and blue
rows, cols, colours = U.shape

# Create a mask: this is a matrix with a 1 if the corresponding
# pixel is known, and zero else
M = np.zeros((rows, cols, colours))
for i in range(rows):
    for j in range(cols):
        for k in range(colours):
            if U[i, j, k] == Ucorr[i, j, k]:
                M[i, j, k] = 1
```

We are now ready to solve the optimization problem using CVXPY. As the problem is rather big ($400 \times 600 \times 3 = 720000$ variables), it is important to choose a good solver that will solve the problem to sufficient accuracy in an acceptable amount of time. For the example at hand, we choose the SCS solver, which can be specified when calling the **solve** function.

In [11]:
```
# Determine the variables and constraints
variables = []
constraints = []
for k in range(colours):
    X = Variable(rows, cols)
    # Add variables
    variables.append(X)
    # Add constraints by multiplying the relevant variable matrix
    # elementwise with the mask
    constraints.append(mul_elemwise(M[:, :, k], X) ==
      \ (M[:, :, k], Ucorr[:, :, k]))

# Create a problem instance with
objective = Minimize(tv(variables[0],variables[1],variables[2]))

# Create a problem instance and solve it using the SCS solver
prob = Problem(objective, constraints)
prob.solve(verbose=True, solver=SCS)
```

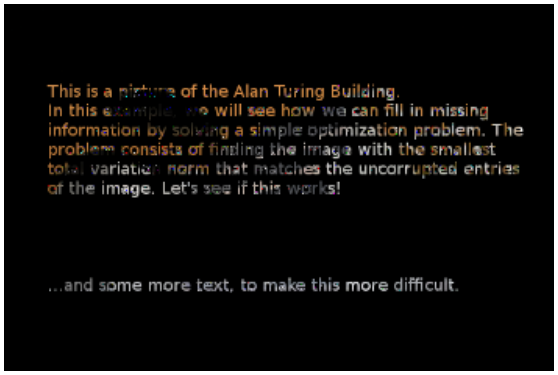
Out [11]:
```
8263910.812250629
```

Now that we solved the optimization problem, we have a solution stored in 'variables'. We have to transform this back into an image and display the result.

In [12]:
```
%matplotlib inline

# Load variable values into a single array.
Urec = np.zeros((rows, cols, colours), dtype=np.uint8)
for i in range(colours):
    Urec[:, :, i] = variables[i].value

fig, ax = plt.subplots(1, 2,figsize=(10, 5))

# Display the inpainted image.
ax[0].imshow(Urec);
ax[0].set_title("Inpainted Image")
ax[0].axis('off')

ax[1].imshow(np.abs(Ucorr[:,:,0:3] - Urec));
ax[1].set_title("Difference Image")
ax[1].axis('off');
```

Inpainted Image

Difference Image



Another typical structure of images is that the **singular values** of the image, considered as matrix, decay quickly. The **singular value decomposition** (SVD) of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is the

matrix product

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T,$$

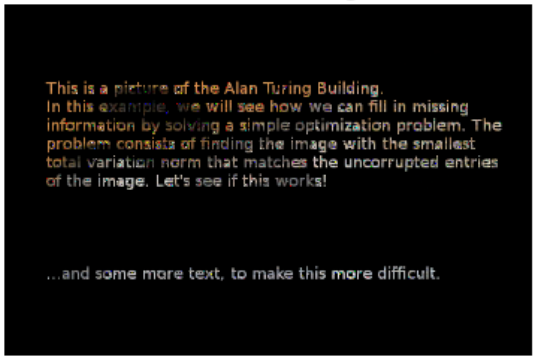
where $\boldsymbol{U} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ are orthogonal matrices, and $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$ is a diagonal matrix with entries $\sigma_1, \ldots, \sigma_{\min\{m,n\}}$ on the diagonal. Instead of minimizing the TV-norm of an image $\boldsymbol{X}$, one may instead try to minimize the **Schatten 1-norm**, defined as the sum of the singular values, $\|\boldsymbol{U}\|_{S_1} = \sigma_1 + \cdots + \sigma_{\min\{m,n\}}$. The problem is then

$$\text{minimize} \quad \|\boldsymbol{X}\|_{S_1} \quad \text{subject to} \quad x_{ij} = u_{ij} \text{ for } (i,j) \in \Omega.$$

As we will see towards the end of the course, this is an instance of a type of convex optimization problem known as **semidefinite programming**. Luckily, CVXPY includes the Schatten 1-norm (also known as nuclear norm) as valid objective function, so we don't have to deal with the details of this problem. As the problem is computationally intensive, we just reproduce the result.

Inpainted Image

Difference Image



In this example, the result appears worse as in the problem involving the TV-norm. Alternatively, one may also use the 1-norm of the image applied to a discrete cosine transfrom (DCT) or a discrete wavelet transform (DWT).

Of course, one could run the above examples for fun with different types of images in an attempt to get rid of certain parts. The image below show the result of applying the total variation inpainting procedure to set a parrot free.

Caged parrot

Free parrot

Part II

Unconstrained Optimization

Lecture 2

In this lecture we will study the unconstrained problem

$$\text{minimize} \quad f(\boldsymbol{x}), \tag{2.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$. Optimality conditions aim to identify properties that potential minimizers need to satisfy in relation to $f(\boldsymbol{x})$. We will review the well known local optimality conditions for differentiable functions from calculus. We then introduce convex functions and discuss some of their properties.

2.1 Unconstrained optimization

Solutions to (2.1) come in different flavours, as in the following definition.

Definition 2.1. A point $\boldsymbol{x}^* \in \mathbb{R}^n$ is a

- *global minimizer* of (2.1) if for all $\boldsymbol{x} \in \mathbb{R}^n$, $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$;
- a *local minimizer*, if there is an open neighbourhood U of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in U$;
- a *strict local minimizer*, if there is an open neighbourhood U of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) < f(\boldsymbol{x})$ for all $\boldsymbol{x} \in U$;
- an *isolated minimizer* if there is an open neighbourhood U of $\boldsymbol{x}^*$ such that $\boldsymbol{x}^*$ is the only local minimizer in U.

Without any further assumptions on f, finding a minimizer is a hopeless task: we simply can't examine the function at *all* points in $\mathbb{R}^n$. The situation becomes more tractable if we assume some *smoothness* conditions. Recall that $C^k(U)$ denotes the set of functions that are k times continuously differentiable on some set U. The following *first-order* necessary condition for optimality is well known. We write $\nabla f(\boldsymbol{x})$ for the gradient of f at $\boldsymbol{x}$, i.e., the vector

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}(\boldsymbol{x}), \ldots, \frac{\partial f}{\partial x_n}(\boldsymbol{x}) \right)^\top$$

Theorem 2.2. *Let $\boldsymbol{x}^*$ be a local minimizer of f and assume that $f \in C^1(U)$ for a neighbourhood of U of $\boldsymbol{x}^*$. Then $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$.*

There are simple examples that show that this is not a sufficient condition: maxima and saddle points will also have a vanishing gradient. If we have access to *second-order information*, in form of the second derivative, or Hessian, of f, then we can say more. Recall that the Hessian of f at $\boldsymbol{x}$, $\nabla^2 f(\boldsymbol{x})$, is the $d \times d$ symmetric matrix given by the second derivatives,

$$\nabla^2 f(\boldsymbol{x}) = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{1 \le i,j \le n}.$$

In the one-variable case we have learned that if x^* is a local minimizer of $f \in C^2([a,b])$, then $f'(x^*) = 0$ *and* $f''(x^*) \ge 0$. Moreover, the conditions $f'(x^*) = 0$ *and* $f''(x^*) > 0$ guarantee that we have a local minimizer. These conditions generalise to higher dimension, but first we need to know what $f''(x) > 0$ when we have more than one variable.

Recall also that a matrix $\boldsymbol{A}$ is **positive semidefinite**, written $\boldsymbol{A} \succeq \boldsymbol{0}$, if for every $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \ge 0$, and positive definite, written $\boldsymbol{A} \succ \boldsymbol{0}$, if $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} > 0$. The property that the Hessian matrix is positive semidefinite is a multivariate generalization of the property that the second derivative is nonnegative. The known conditions for a minimizer involving the second derivative generalize accordingly.

Theorem 2.3. *Let $f \in C^2(U)$ for some open set U and $\boldsymbol{x}^* \in U$. If $\boldsymbol{x}^*$ is a local minimizer, then $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite. Conversely, if $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite, then $\boldsymbol{x}^*$ is a strict local minimizer.*

Unfortunately, the above criteria are not able to identify global minimizers, as differentiability is a local property.

2.2 Convex functions

We now come to the central notion of this course.

Definition 2.4. A set $C \subseteq \mathbb{R}^n$ is **convex** if for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda \in [0,1]$, the line $\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in C$. A **convex body** is a convex set that is closed and bounded.

Definition 2.5. Let $S \subseteq \mathbb{R}^n$. A function $f\colon S \to \mathbb{R}$ is called *convex* if S is convex and for all $\boldsymbol{x}, \boldsymbol{y} \in S$ and $\lambda \in [0,1]$,

$$f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \le \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}).$$

The function f is called *strictly convex* if

$$f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) < \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}).$$

A function f is called *concave*, if $-f$ is convex.

Figure 2.2 illustrates how a convex function of one variable looks like. The graph of the function lies below any line connecting two points on it.

Convex function have pleasant properties, while at the same time covering many of the functions that arise in applications. Perhaps the most important property is that local minima are global minima.

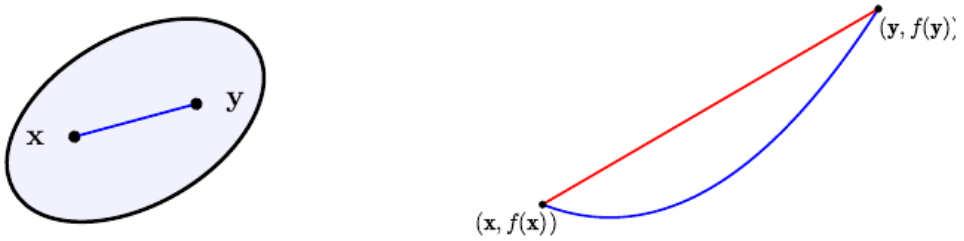


Figure 2.2: A convex set and a convex function

Theorem 2.6. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then any local minimizer of f is a global minimizer.*

Proof. Let $\boldsymbol{x}^*$ be a local minimizer and assume that it is not a global minimizer. Then there exists a vector $\boldsymbol{y} \in \mathbb{R}^d$ such that $f(\boldsymbol{y}) < f(\boldsymbol{x}^*)$. Since f is convex, for any $\lambda \in [0,1]$ and $\boldsymbol{x} = \lambda \boldsymbol{y} + (1-\lambda)\boldsymbol{x}^*$ we have

$$f(\boldsymbol{x}) \leq \lambda f(\boldsymbol{y}) + (1-\lambda) f(\boldsymbol{x}^*) < \lambda f(\boldsymbol{x}^*) + (1-\lambda) f(\boldsymbol{x}^*) = f(\boldsymbol{x}^*).$$

This holds for all $\boldsymbol{x}$ on the line segment connecting $\boldsymbol{y}$ and $\boldsymbol{x}^*$. Since every open neighbourhood U of $\boldsymbol{x}^*$ contains a bit of this line segment, this means that every open neighbourhood U of $\boldsymbol{x}^*$ contains an $\boldsymbol{x} \neq \boldsymbol{x}^*$ such that $f(\boldsymbol{x}) \leq f(\boldsymbol{x}^*)$, in contradiction to the assumption that $\boldsymbol{x}^*$ is a local minimizer. It follows that $\boldsymbol{x}^*$ has to be a global minimizer. □

Remark 2.7. Note that in the above theorem we made no assumptions about the differentiability of the function f! In fact, while a convex function is always *continuous*, it need not be differentiable. The function $f(x) = |x|$ is a typical example: it is convex, but not differentiable at $x = 0$.

Example 2.8. Affine functions $f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{a} \rangle + b$ and the exponential function e^x are examples of convex functions.

Example 2.9. In optimization we will often work with functions of matrices, where an $m \times n$ matrix is considered as a vector in $\mathbb{R}^{m \times n} \cong \mathbb{R}^{mn}$. If the matrix is symmetric, that is, if $\boldsymbol{A}^\top = \boldsymbol{A}$, then we only care about the upper diagonal entries, and we consider the space $\mathcal{S}^n$ of symmetric matrices as a vector space of dimension $n(n+1)/2$ (the number of entries on and above the main diagonal). Important functions on symmetric matrices that are convex are the operator norm $\|\boldsymbol{A}\|_2$, defined as

$$\|\boldsymbol{A}\|_2 := \max_{\boldsymbol{x}\colon \|\boldsymbol{x}\| \leq 1} \frac{\|\boldsymbol{A}\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2},$$

or the function $\log\det(\boldsymbol{X})$, defined on the set of *positive semidefinite* symmetric matrices $\mathcal{S}^n_+$.

There are useful ways of characterising convexity using differentiability.

Theorem 2.10. *1. Let $f \in C^1(\mathbb{R}^n)$. Then f is convex if and only if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$,*

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}).$$

2. *Let $f \in C^2(\mathbb{R}^n)$. Then f is convex if and only if $\nabla^2 f(\boldsymbol{x})$ is positive semidefinite. If $\nabla^2 f(\boldsymbol{x})$ is positive definite, then f is strictly convex.*

Example 2.11. Consider a quadratic function of the form

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x} + c,$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is symmetric. Writing out the product, we get

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} &= \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ &= \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix} \begin{pmatrix} a_{11}x_1 + \cdots + a_{1n}x_n \\ \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n \end{pmatrix} = \sum_{i=1}^n \sum_{j=1}^n a_{ij}x_i x_j.\end{aligned}$$

Because $\boldsymbol{A}$ is symmetric, we have $a_{ij} = a_{ji}$, and the above product simplifies to

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \sum_{i=1}^n a_{ii}x_i^2 + 2\sum_{1\le i<j\le n} a_{ij}x_i x_j.$$

This is a quadratic function, because it involves products of the x_i. The gradient and the Hessian of $f(\boldsymbol{x})$ are found by computing the partial derivatives of f:

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^n a_{ij}x_j + b_i, \quad \frac{\partial^2 f}{\partial x_i \partial x_j} = a_{ij}.$$

In summary, we have

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, \quad \nabla^2 f(\boldsymbol{x}) = \boldsymbol{A}.$$

Using the previous theorem, we see that f is convex **if and only if** $\boldsymbol{A}$ is positive semidefinite. A typical example for such a function is

$$f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{b}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{b}.$$

The matrix $\boldsymbol{A}^T\boldsymbol{A}$ is always symmetric and positive semidefinite (why?) so that the function f is convex.

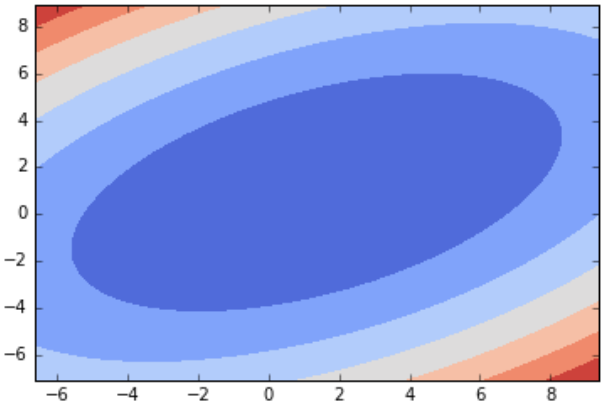
A convenient way to visualise a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ is through **contour plots**. A **level set** of the function f is a set of the form

$$\{\boldsymbol{x} : f(\boldsymbol{x}) = c\},$$

where c is the **level**. Each such level set is a curve in $\mathbb{R}^2$, and a contour plot is a plot of a collection of such curves for various c. If one colours the areas between adjacent curves, one

gets a plot as in the following figure. A *convex function* is has the property that there is only one *sink* in the contour plot.

In [16]:
```
# import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
% matplotlib inline

# Create random data: we use the randn function
X = np.random.randn(3,2)
y = np.random.randn(3)

# Solve least squares problem minimize \|X\beta-y\|^2
# the index 0 says that we get the first component of the solution
# (the function lstsq give more output than just the beta vector)
beta = la.lstsq(X,y)[0]

# Create function and plot the contours
def f(a,b):
    return sum((a*X[:,0]+b*X[:,1]-y)**2)

# Find the "right" boundaries around the minimum
xx = np.linspace(beta[0]-8,beta[0]+8,100)
yy = np.linspace(beta[1]-8,beta[1]+8,100)
XX, YY = np.meshgrid(xx,yy)

Z = np.zeros(XX.shape)
for i in range(Z.shape[0]):
    for j in range(Z.shape[1]):
        Z[i,j] = f(XX[i,j],YY[i,j])

cmap = plt.cm.get_cmap("coolwarm")
plt.contourf(XX,YY,Z, cmap = cmap)
plt.show()
```



Lecture 3

Most modern optimization methods are iterative: they generate a sequence of points $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ in $\mathbb{R}^n$ in the hope that this sequences will converge to a local or global minimizer $\boldsymbol{x}^*$ of a function $f(\boldsymbol{x})$. A typical rule for generating such a sequence would be to start with a vector $\boldsymbol{x}_0$, chosen by an educated guess, and then for $k \geq 0$, move from step k to $k+1$ by

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

in a way that ensures that $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$. The parameter α_k is called the **step length**, while $\boldsymbol{p}_k$ is the **search direction**. In this lecture we discuss one such method, the method of gradient descent, or steepest descent, and discuss how to select the right step length.

3.1 Gradient descent

In the method of gradient descent, the search direction is chosen as

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k). \tag{3.1}$$

To see why this makes sense, let $\boldsymbol{p}$ be a direction and consider the Taylor expansion

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{p}) = f(\boldsymbol{x}_k) + \alpha \langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k) \rangle + O(\alpha^2).$$

Considering this as a function of α, the rate of change in direction $\boldsymbol{p}$ at $\boldsymbol{x}_k$ is the derivative of this function at $\alpha = 0$,

$$\frac{\mathrm{d} f(\boldsymbol{x}_k + \alpha \boldsymbol{p})}{\mathrm{d}\alpha}|_{\alpha=0} = \langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k) \rangle,$$

also known as the **directional derivative** of f at $\boldsymbol{x}_k$ in the direction $\boldsymbol{p}$. This formula indicates that the rate of change is *negative*, and we have a **descent direction**, if $\langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k) \rangle < 0$.

The Cauchy-Schwarz inequality (see Preliminaries, Page 9) gives the bounds

$$-\|\boldsymbol{p}\|_2 \|\nabla f(\boldsymbol{x}_k)\|_2 \leq \langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k) \rangle \leq \|\boldsymbol{p}\|_2 \|\nabla f(\boldsymbol{x}_k)\|_2.$$

We see that the rate of change is the smallest when the first inequality is an equality, which happens if

$$\boldsymbol{p} = -\alpha \nabla f(\boldsymbol{x}_k)$$

for some $\alpha > 0$.

For a visual interpretation of what it means to be a descent direction, note that the **angle** θ between a vector $\boldsymbol{p}$ and the gradient $\nabla f(\boldsymbol{x})$ at a point $\boldsymbol{x}$ is given by (see Preliminaries, Page 9)

$$\langle \boldsymbol{x}, \nabla f(\boldsymbol{x}) \rangle = \|\boldsymbol{p}\|_2 \|\nabla f(\boldsymbol{x})\|_2 \cos(\theta).$$

This is negative if the vector $\boldsymbol{p}$ forms and angle greater than $\pi/2$ with the gradient. Recall that the gradient points in the direction of steepest ascent, and is orthogonal to the *level sets*. If you are standing on the slope of a mountain, walking along the level set lines will not change your elevation, the gradient points to the steepest upward direction, and the negative gradient to the steepest descent.

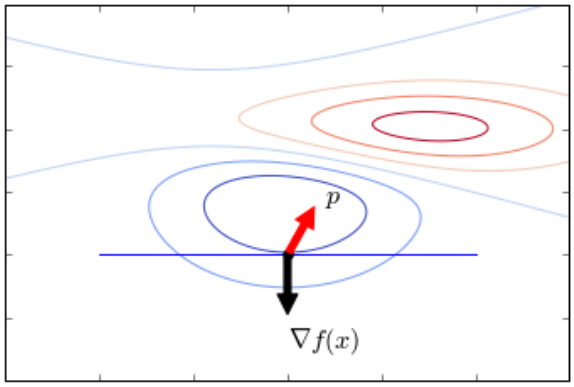


Figure 3.3: A descent direction

Any multiple $\alpha \nabla f(\boldsymbol{x}_k)$ points in the direction of steepest descent, but we have to choose a sensible parameter α to ensure that we make sufficient progress, but at the same time don't overshoot. Ideally, we would choose the value α_k that minimizes $f(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k))$. While finding such a minimizer is in general not easy (see Section Lecture 4 for alternatives), for quadratic functions in can be given in closed form.

Linear least squares

Consider a function of the form

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2.$$

In Problem Sheet 1 you will show that that the Hessian is symmetric and positive semidefinite, with the gradient given by

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}).$$

The method of gradient descent proceeds as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}).$$

To find the best α_k, we compute the minimum of the function

$$\alpha \mapsto f(\boldsymbol{x}_k - \alpha \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b})). \tag{3.2}$$

If we set $\boldsymbol{r}_k := \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) = -\nabla f(\boldsymbol{x}_k)$ and compute the minimum of (3.2) by differentiating, we get the step length

$$\alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{r}_k^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k} = \frac{\|\boldsymbol{r}_k\|_2^2}{\|\boldsymbol{A}\boldsymbol{r}_k\|_2^2}.$$

(Verify this!) Note also that when we have $\boldsymbol{r}_k$ and α_k, we can compute the next $\boldsymbol{r}_k$ as

$$\begin{aligned} r_{k+1} &= \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{k+1}) \\ &= \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}(\boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k)) \\ &= \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k - \alpha_k \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k) = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k. \end{aligned}$$

The gradient descent algorithm for the linear least squares problem proceeds by first computing $\boldsymbol{r}_0 = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0)$, and then at each step

$$\begin{aligned} \alpha_k &= \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{r}_k^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k. \end{aligned}$$

Does this work? How do we know when to stop? It is worth noting that the residual satisfies $\boldsymbol{r} = 0$ if and only if $\boldsymbol{x}$ is a stationary point, in our case, a minimizer. One criteria for stopping could then be to check whether $\|\boldsymbol{r}_k\|_2 \leq \varepsilon$ for some given tolerance $\varepsilon > 0$. One potential problem with this criterion is that the function can become *flat* long before reaching a minimum, so an alternative stopping method would be to stop when the difference between two successive points, $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2$, becomes smaller than some $\varepsilon > 0$.

Example 3.1. We plot the trajectory of gradient descent with the data

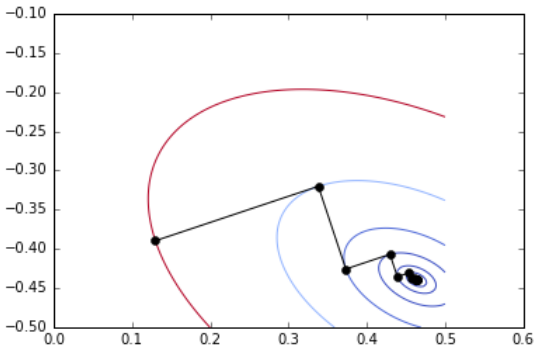


Figure 3.4: Trajectory of gradient descent

$$\boldsymbol{A} = \begin{pmatrix} 2 & 0 \\ 1 & 3 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}.$$

As can be seen from the plot, we always move in the direction orthogonal to a level set, and stop at a point where we are tangent to a level set.

Lecture 4

Iterative algorithms for solving a problem of the form

$$\text{minimize} \quad f(\boldsymbol{x}) \tag{4.1}$$

on $\mathbb{R}^n$ generate a sequence of vectors $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ in the hope that this sequence converges to a (local or global) minimizer $\boldsymbol{x}^*$ of (4.1). In this lecture we first study step length selection procedures and then study what it means for a sequence to converge, and how to quantify the speed of convergence.

4.1 Step length selection

When moving in a descent direction $\boldsymbol{p}_k$ (not necessarily the steepest descent direction),

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

we can only guarantee that the function value will decrease if the step length α_k is small enough. If we move too far in a descent direction, we might even land at a point where f is larger than where we started! It is therefore important to choose a step length that

- is not too small (so that the algorithm does not take too long);
- is not too large (so that we don't end up at a point with larger function value);
- is easy to compute.

An optimal step α for a descent method would be the minimizer of the function

$$\alpha \mapsto \varphi(\alpha) := f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k).$$

In practice, minimizing this function is not always the most efficient thing to do (or even possible). One would rather choose a step length that satisfies some criteria that ensure that the sequence $\boldsymbol{x}_k$ converges to a minimizer $\boldsymbol{x}^*$ under suitable conditions on a function f. One such condition is a **sufficient decrease condition**,

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + c\alpha \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{p}_k \rangle =: \ell(\alpha),$$

with $c \in (0, 1)$. Note that $\varphi'(0) = \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{p}_k \rangle < 0$, because $\boldsymbol{p}_k$ is a descent direction. The function $\ell(\alpha)$ is therefore a line through $f(\boldsymbol{x}_k)$ with a slope $c\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{p}_k \rangle = c\varphi'(0) > \varphi'(0)$.

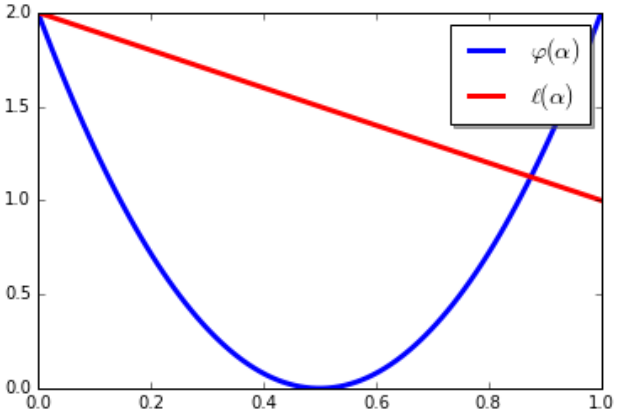


Figure 4.5: The sufficient decrease condition

To see why this condition is necessary, consider the function $f(x) = x^2 - 1$ and the sequence $x_k = \sqrt{1 + 1/k}$ for $k \geq 1$. Clearly, the sequence $f(\boldsymbol{x}_k) = 1/k$ decreases, but fails to converge to the minimizer $f(0) = -1$.

The sufficient decrease condition (also called Armijo condition) can always be satisfied if α is chosen small enough, but the algorithm may become very slow. It is therefore common to supplement the sufficient descent condition with other criteria that guarantee that sufficient progress is made. Two of the commonly used criteria are:

- the Wolfe conditions, which add a *curvature condition*

  $$\varphi'(\alpha) \geq \tilde{c}\varphi'(0)$$

  for some $\tilde{c} \in (c, 1)$, which gives a lower bound on the slope of the new point;

- the Armijo-Goldstein conditions, which state that a step length α_k should additionally satisfy bound

  $$f(\boldsymbol{x}_k) + (1 - c)\alpha_k \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{p}_k \rangle \leq f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k), \tag{4.1}$$

  which gives a lower bound on the step size.

Another common approach is **backtracking**: in this method one uses a high initial value of α (for example, $\alpha = 1$), and then decreases it until the sufficient descent condition is satisfied.

Example 4.1. Consider the function $f\colon \mathbb{R}^2 \to \mathbb{R}$ defined by $f(\boldsymbol{x}) = x_1^2 + x_2^2$. The gradient is $\nabla f(\boldsymbol{x}) = 2\boldsymbol{x}$, and the φ function at $\boldsymbol{x}_k = (1, 1)^\top$

$$\varphi(\alpha) = f(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k)) = 2(1 - 2\alpha)^2, \quad \varphi'(\alpha) = -8(1 - 2\alpha).$$

The Armijo-Goldstein conditions (4.1) then state that we can choose α such that

$$2(1 - 4(1 - c)\alpha) \leq 2(1 - 2\alpha)^2 \leq 2(1 - 4c\alpha).$$

The optimal step length in this case would be $\alpha = 0.5$.

4.2 Convergence of iterative methods

A sequence of vectors $\{\boldsymbol{x}_k\}$ in $\mathbb{R}^n$, $k \geq 0$, *converges* to a vector $\boldsymbol{x}^*$ with respect to a norm $\|\cdot\|$ as $k \to \infty$, written $\boldsymbol{x}_k \to \boldsymbol{x}$, if the sequence of numbers $\|\boldsymbol{x}_k - \boldsymbol{x}^*\|$ converges to zero. More formally, if for every $\varepsilon > 0$ there exists an index N such that for all $n \geq N$,

$$\|\boldsymbol{x}_n - \boldsymbol{x}^*\| < \varepsilon.$$

Iterative algorithms will rarely find the exact solution to a problem like (4.1), so we will usually be happy to find a solution that differs from the true one by at most some specified accuracy.

Definition 4.2. A sequence of vectors $\{\boldsymbol{x}_k\}$, $k \geq 0$, is said to converge to $\boldsymbol{x}^*$

(a) linearly (or Q-linear, Q for quotient), if there exist an $r \in (0, 1)$ such that for sufficiently large k,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq r\|\boldsymbol{x}_k - \boldsymbol{x}^*\|.$$

(b) superlinearly, if

$$\lim_{k\to\infty} \frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|} = 0,$$

(c) with order p, if there exists a constant $M > 0$, such that for sufficiently large k,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq M\|\boldsymbol{x}_k - \boldsymbol{x}^*\|^p.$$

The case $p = 2$ is called *quadratic convergence*.

Of course, as mentioned earlier, these definitions depend on the choice of a norm. It can be shown that quadratic convergence implies superlinear convergence, and superlinear convergence implies linear convergence.

Example 4.3. Consider the sequence of numbers $x_k = 1/2^{r^k}$ for some $r > 1$. Clearly, $x_k \to x^* = 0$ as $k \to \infty$. Moreover,

$$x_{k+1} = \frac{1}{2^{r^{k+1}}} = \frac{1}{2^{r^k r}} = \left(\frac{1}{2^{r^k}}\right)^r = x_k^r,$$

which shows that the sequence has rate of convergence r.

4.3 Convergence of gradient descent

In this section, a norm $\|\cdot\|$ will refere to the 2-norm, unless otherwise stated. We now study the convergence of gradient descent for the least squares problem

$$\text{minimize} \quad f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2, \tag{4.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ of full rank. As we have seen in Lecture 3, the gradient descent method is the procedure

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

where the step length and the *residual* are given by

$$\alpha_k = \frac{\|\boldsymbol{r}_k\|^2}{\|\boldsymbol{A}\boldsymbol{r}_k\|^2}, \quad \boldsymbol{r}_k = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) = -\nabla f(\boldsymbol{x}_k).$$

At the minimizer, the residual is

$$\boldsymbol{r} = -\nabla f(\boldsymbol{x}^*) = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*) = 0, \tag{4.2}$$

and as the sequence $\boldsymbol{x}_k$ converges to $\boldsymbol{x}^*$, the norms of the residuals converge to 0. Conversely, the residual is related to the difference $\boldsymbol{x}_k - \boldsymbol{x}^*$ by

$$\boldsymbol{r}_k = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k - (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*)) = \boldsymbol{A}^\top\boldsymbol{A}(\boldsymbol{x_k} - \boldsymbol{x}^*), \tag{4.3}$$

where we used the "intelligent zero" (4.2). Therefore

$$\|\boldsymbol{x}_k - \boldsymbol{x}^*\| = \|(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{r}\| \leq \|(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\|\|\boldsymbol{r}_k\|,$$

where $\|\boldsymbol{B}\| = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \|\boldsymbol{B}\boldsymbol{x}\|/\|\boldsymbol{x}\|$ is the operator norm of a matrix $\boldsymbol{B}$ with respect to the 2-norm. Consequently, if the sequence $\|\boldsymbol{r}_k\|$ converges to zero, so does the sequence $\|\boldsymbol{x}_k - \boldsymbol{x}^*\|$. A reasonable criterium is to stop the algorithm is therefore when the residual norm $\|\boldsymbol{r}_k\|$ is below a predefined tolerance ε.

The following theorem (whose proof we omit) shows that the gradient descent method for linear least squares converges linearly with respect to the $\boldsymbol{A}$ norm. The statement involves the *condition number* of $\boldsymbol{A}$[5]. This quantity is defined as

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^\dagger\|.$$

Theorem 4.4. *The error in the $k+1$-th iterate is bounded by*

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

In particular, the gradient descent algorithm converges linearly.

[5]The concept of condition number, introduced by Alan Turing while in Manchester, is one of the most important ideas in numerical analysis, as it is indispensable in studying the performance of numerical algorithms.

Lecture 5

In Lecture 4 we found out that gradient descent works, and has linear convergence. In this lecture we introduce Newton's method, an algorithm that takes advantage of the second derivative and has quadratic convergence under certain circumstances. Throughout this lecture, $\|\cdot\|$ will refer to the 2-norm $\|\cdot\|_2$.

5.1 Newton's Method

Let $f \in C^2(\mathbb{R}^n)$ and let's look again at the unconstrained problem

$$\text{minimize} \quad f(\boldsymbol{x}).$$

Newton's method starts with a guess $\boldsymbol{x}_0$ and then proceeds to compute a sequence of points $\{\boldsymbol{x}_k\}_{k\geq 0}$ in $\mathbb{R}^n$ by the rule

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \nabla^2 f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k), \quad k \geq 0. \tag{5.1}$$

The algorithm stops when $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \varepsilon$ for some predefined tolerance $\varepsilon > 0$. In the context of the general scheme $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$, the step length is $\alpha_k = 1$, and the search direction is the inverse of the Hessian multiplied with the negative gradient.

Recall that the inner product $\langle \boldsymbol{p}, \nabla f(\boldsymbol{x}) \rangle$ is the directional derivative of f, and that a *descent direction* is a direction in which the rate of change (slope) is negative. The following gives a criterion for the search direction in Newton's method to be a descent direction.

Lemma 5.1. *Let $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ be a positive definite symmetric matrix and $f \in C^1(\boldsymbol{R}^n)$. Then $\boldsymbol{p} = -\boldsymbol{B}^{-1}\nabla f(\boldsymbol{x})$ is a descent direction of f at $\boldsymbol{x}$.*

Proof. If $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ is symmetric and positive definite, then $\boldsymbol{B}^{-1}$ is also positive definite, since for all $\boldsymbol{v} \in \mathbb{R}^n$,

$$\boldsymbol{v}^\top \boldsymbol{B}^{-1}\boldsymbol{v} = (\boldsymbol{B}\boldsymbol{B}^{-1}\boldsymbol{v})^\top \boldsymbol{B}^{-1}\boldsymbol{v} = (\boldsymbol{B}^{-1}\boldsymbol{v})^\top \boldsymbol{B}^\top (\boldsymbol{B}^{-1}\boldsymbol{v}) = (\boldsymbol{B}^{-1}\boldsymbol{v})^\top \boldsymbol{B}(\boldsymbol{B}^{-1}\boldsymbol{v}) > 0.$$

(This can also be seen by noting that the eigenvalues of $\boldsymbol{B}^{-1}$ are the inverses of the eigenvalues of $\boldsymbol{B}$.) For $\boldsymbol{p} = -\boldsymbol{B}^{-1}\nabla f(\boldsymbol{x})$ we then get

$$\langle \boldsymbol{p}, \nabla f(\boldsymbol{x}) \rangle = -\langle \boldsymbol{B}^{-1}\nabla f(\boldsymbol{x}), \nabla f(\boldsymbol{x}) \rangle = -\nabla f(\boldsymbol{x})^\top \boldsymbol{B}^{-1}\nabla f(\boldsymbol{x}) < 0,$$

which shows that $\boldsymbol{p}$ is a descent direction. □

To better understand Newton's method, we first look at the one dimensional case.

Example 5.2. Let $f \in C^2(\mathbb{R})$. In this case Newton's method is described as

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}, \quad k \geq 0. \tag{5.2}$$

Newton's method looks for a local minimizer in the form of a point x^* such that $f'(x^*) = 0$ and $f''(x^*) > 0$. Setting $g(x) := f'(x)$, we are looking for a *root* x^*,

$$g(x^*) = 0.$$

One approach to find such a root is to approximate the function $g(x)$ at a point x_k by its tangent line,

$$g(x) \approx g(x_k) + g'(x_k)(x - x_k),$$

and then identify the next iterate x_{k+1} as the root of this linear approximation:

$$g(x_k) + g'(x_k)(x_{k+1} - x_k) = 0 \iff x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$$

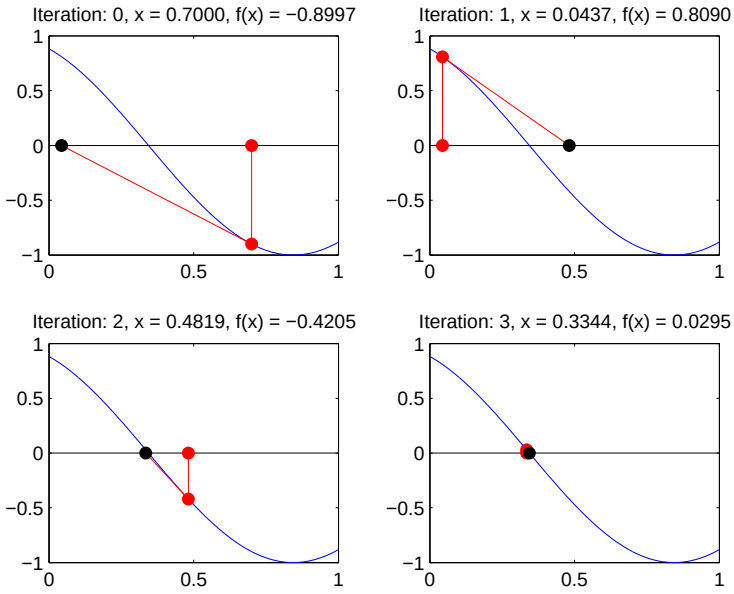


Figure 5.6: Newton's method

Geometrically this corresponds to taking the tangent to g at x_k and setting x_{k+1} to be the intersection of this tangent line with the x-axis, as shown in Figure 5.6. Replacing $g(x) = f'(x)$ gives precisely Newton's method (5.2).

Another way to understand Newton's method is to view it in contrast with gradient descent. While gradient descent corresponds to working with a *linear approximation*

$$f(\boldsymbol{x}_{k+1}) \approx f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \rangle,$$

Newton's method is based on the *quadratic approximation*,

$$f(\boldsymbol{x}_{k+1}) \approx f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \rangle + \frac{1}{2} \langle \nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k), \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \rangle.$$

Example 5.3. Consider the function f on $\mathbb{R}^2$,

$$f(\boldsymbol{x}) = \frac{1}{2}(x_1^2 + 10x_2^2).$$

Starting with $\boldsymbol{x}_0 = (10, 1)^\top$, gradient descent takes 84 iterations to reach accuracy 10^{-6}, while Newton's method, unsurpringly, takes only one.

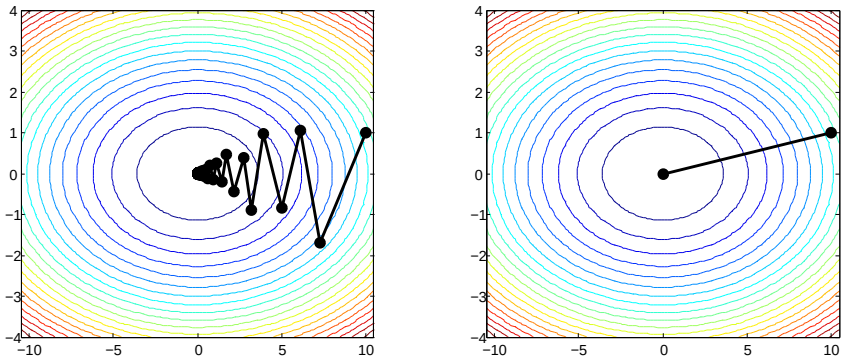


Figure 5.7: Gradient descent vs. Newton's method on a quadratic function.

In practice, when implementing Newton's method one does not explicitly compute the inverse of the Hessian. The reason is that one does not need the inverse itself, but only the product $\nabla^2 f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k)$, which is the solution of a system of equations $\nabla^2 f(\boldsymbol{x}_k)\boldsymbol{y} = \nabla f(\boldsymbol{x}_k)$ that can be solved much more efficiently. One therefore replaces the update step (5.1) with the following two steps:

$$\begin{aligned} \text{Find } \boldsymbol{y} \text{ such that} \quad & \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{y} = -\nabla f(\boldsymbol{x}_k), \\ & \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{y}. \end{aligned}$$

There is a lot that can go wrong with Newton's method. In particular, the matrix $\nabla^2 f(\boldsymbol{x})$ has to be non-singular, or invertible, at every step. If, however, we start at a point $\boldsymbol{x}_0$ that is not too far from a local minimizer $\boldsymbol{x}^*$ with $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$ and $\nabla^2 f(\boldsymbol{x}^*)$ positive definite, then we are on the safe side.

Lemma 5.4. *Let $\boldsymbol{x}^* \in \mathbb{R}^n$ be such that $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite. Then there exists an open neighbourhood U of $\boldsymbol{x}^*$ such that for all $\boldsymbol{x} \in U$, $\nabla^2 f(\boldsymbol{x})$ is positive definite.*

For the main result of this lecture, namely the quadratic convergence of Newton's method, we make the additional assumption that the Hessian $\nabla^2 f(\boldsymbol{x})$ is Lipschitz continuous as a function of $\boldsymbol{x}$.

Definition 5.5. A function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz continuous on a domain $\Omega \subseteq \mathbb{R}^n$ with respect to a pair of norms on $\mathbb{R}^n$ and $\mathbb{R}^m$ if there is a constant $L > 0$ such that for all $\boldsymbol{x}, \boldsymbol{y} \in \Omega$,

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

The constant L is called the *Lipschitz constant* of the map.

In particular, the Hessian of a function $f \in C^2(\mathbb{R}^n)$, considered as a map from $\mathbb{R}^n$ to $\mathbb{R}^{n\times n}$, is Lipschitz continuous with respect to a norm on $\mathbb{R}^n$ and the corresponding operator norm on $\mathbb{R}^{n\times n}$, if for any $\boldsymbol{x}, \boldsymbol{y}$ we have

$$\|\nabla^2 f(\boldsymbol{x}) - \nabla^2 f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|.$$

Theorem 5.6. *Let $f \in C^2(\mathbb{R}^n)$ and $\boldsymbol{x}^* \in \mathbb{R}^n$ a local minimizer with $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}^*) > 0$. Then for $\boldsymbol{x}_0$ sufficiently close to $\boldsymbol{x}^*$, Newton's method has quadratic convergence.*

Proof. (Optional) Assume that $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite. Consider the difference

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| &= \|\boldsymbol{x}_k - \boldsymbol{x}^* - \nabla^2 f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)\| \\ &\overset{(1)}{=} \|\boldsymbol{x}_k - \boldsymbol{x}^* - \nabla^2 f(\boldsymbol{x}_k)^{-1}(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}^*))\| \\ &= \|\nabla^2 f(\boldsymbol{x}_k)^{-1}(\nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}^*) - (\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}^*))\|\end{aligned} \tag{5.3}$$

where (1) follows from $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$. The Fundamental Theorem of Calculus tells us

$$\begin{aligned}\nabla f(\boldsymbol{x}^*) - \nabla f(\boldsymbol{x}_k) &= \int_0^1 \frac{\mathrm{d}}{\mathrm{d}t}\nabla f(\boldsymbol{x}_k + t(\boldsymbol{x}^* - \boldsymbol{x}_k))\,\mathrm{d}t \\ &= \int_0^1 \nabla^2 f(\boldsymbol{x}_k + t(\boldsymbol{x}^* - \boldsymbol{x}_k))(\boldsymbol{x}^* - \boldsymbol{x}_k)\,\mathrm{d}t.\end{aligned}$$

Continuing from (5.3), by inserting this identity,

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| &= \left\|\nabla^2 f(\boldsymbol{x}_k)^{-1}\int_0^1 \left[\nabla^2 f(\boldsymbol{x}_k) - \nabla^2 f(\boldsymbol{x}_k + t(\boldsymbol{x}^* - \boldsymbol{x}_k))\right](\boldsymbol{x}_k - \boldsymbol{x}^*)\,\mathrm{d}t\right\| \\ &\leq \|\nabla^2 f(\boldsymbol{x}_k)^{-1}\|\cdot \\ &\quad \int_0^1 \|\nabla^2 f(\boldsymbol{x}_k) - \nabla^2 f(\boldsymbol{x}_k + t(\boldsymbol{x}^* - \boldsymbol{x}_k))\|\,\mathrm{d}t \cdot \|(\boldsymbol{x}_k - \boldsymbol{x}^*)\|.\end{aligned}$$

Applying the Lipschitz bound to the term inside the integral gives

$$\|\nabla^2 f(\boldsymbol{x}_k) - \nabla^2 f(\boldsymbol{x}_k + t(\boldsymbol{x}^* - \boldsymbol{x}_k))\| \leq Lt\|\boldsymbol{x}_k - \boldsymbol{x}^*\|.$$

Integrating this out, we end up with the bound

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq \frac{L}{2}\|\nabla^2 f(\boldsymbol{x}_k)^{-1}\| \cdot \|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2.$$

The only remaining issue is that the "constant" on the right-hand side is not a constant. However, the Lipschitz continuity implies that $\nabla^2 f(\boldsymbol{x}_k)$ converges to $\nabla^2 f(\boldsymbol{x}^*)$ if $\boldsymbol{x}_k$ converges to $\boldsymbol{x}^*$. Since

the inversion of a matrix is a continuous operation, also $\nabla^2 f(\boldsymbol{x}_k)^{-1}$ converges to $\nabla^2 f(\boldsymbol{x}^*)^{-1}$. In particular, if $\boldsymbol{x}_k$ is sufficiently close to $\boldsymbol{x}^*$, we have $\|\nabla^2 f(\boldsymbol{x}_k)^{-1}\| \leq 2\|\nabla^2 f(\boldsymbol{x}^*)\|$. Setting $M := L\|\nabla^2 f(\boldsymbol{x}^*)^{-1}\|/2$, we end up with the bound

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq M \cdot \|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2.$$

By Lemma 5.4 there exists an open neighbourhood around $\boldsymbol{x}^*$ in which $\nabla^2 f(\boldsymbol{x})$ is positive definite, and within this neighbourhood there is an $\boldsymbol{x}_0$ such that $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 < 1/M$, which ensures that all following iterates remain in U. This shows quadratic convergence. □

Note that the conditions for quadratic convergence in an open neighbourhood U of $\boldsymbol{x}^*$ are precisely that f is convex on U.

5.2 Quasinewton methods

One drawback of Newton's method is that it requires the computation of the Hessian matrix, which can be expensive. Quasinewton methods use an approximation of the Hessian, $\boldsymbol{B}_k \approx \nabla f(\boldsymbol{x}_k)$, at each step of the algorithm. These are construction in a way that $\boldsymbol{B}_{k+1}$ can easily be computed from $\boldsymbol{B}_k$. A popular method, that is used often in practical applications because of its efficiency, is the **Broyden-Fletcher-Shanno-Goldfarb** (**BFGS**) method. The BFGS method may be described as follows.

- Start with $\boldsymbol{x}_0$, $\boldsymbol{B}_0$.
- For $k \geq 0$, compute

$$\begin{aligned}
\boldsymbol{p}_k &= \boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{x}_k) \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \text{ for a suitable step length } \alpha_k \\
\boldsymbol{s}_k &= \alpha_k \boldsymbol{p}_k \\
\boldsymbol{y}_k &= \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) \\
\boldsymbol{B}_{k+1} &= \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^\top}{\boldsymbol{y}_k^\top \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^\top \boldsymbol{B}_k}{\boldsymbol{s}_k^\top \boldsymbol{B}_k \boldsymbol{s}_k}.
\end{aligned}$$

- Stop if $\|\nabla f(\boldsymbol{x}_k)\| < \varepsilon$ for some tolerance ε, or if $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \varepsilon$.

Lecture 6

In this lecture we will embark on one of the most important modern applications of convex optimization: **machine learning**. The goal of machine learning is to develop methods to automatically *learn* a function

$$h\colon \mathcal{X} \to \mathcal{Y},$$

where $\mathcal{X}$ is a space of *inputs* or *features*, and $\mathcal{Y}$ consists of *outputs* or *responses*. The input space $\mathcal{X}$ is usually an $\mathbb{R}^n$, though the inputs could represent anything such as images, texts, emails, genome sequences, or networks, financial time series, or demographic data. The output could be either **quantitative** values, such as a temperature or the amount of some substance in the body, or **qualitative** or **categorical**, such as YES, NO or $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (for recognising spam or handwritten digits, respectively). The first type of problem is usually called **regression**, while the latter is called **classification**. Machine learning techniques underlie much of modern technology, from car electronics to online product recommendation systems and search engines.

6.1 Supervised Learning

In supervised learning, we have at our disposal a collection of input-output pairs

$$(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y},\ 1 \leq i \leq m,$$

and the goal is to *learn* a function $h\colon \mathcal{X} \to \mathcal{Y}$ from this data. The given pairs $\{(\boldsymbol{x}_i, y_i)\}_{1 \leq i \leq m}$ are called the **training set**.

Example 6.1. (Handwriting recognition) Given a dataset of pixel matrices, each representing a grey-scale image, with associated **labels** telling us for each image the number it represents, the task is to use this to train a computer program to recognise new numbers. Such classification tasks are often carried out using **deep neural networks**.

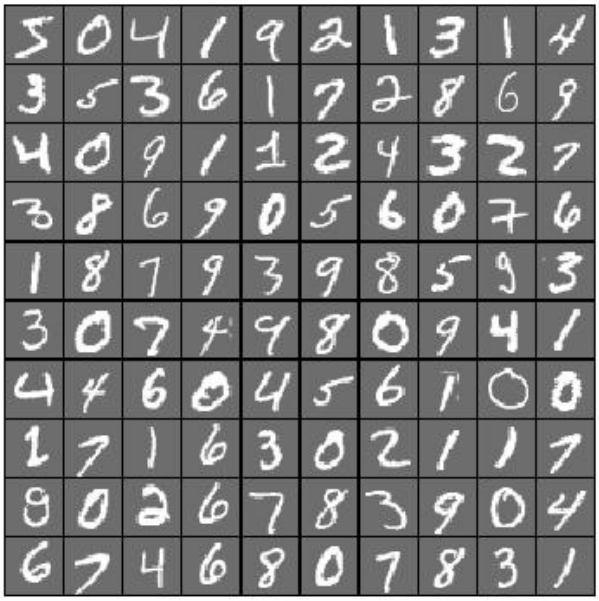
Example 6.2. (Linear regression) Recall from Lecture 1 the problem of finding a function of the form

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

Given a set of input-output pairs $(\boldsymbol{x}_i, y_i)$, arranged in a matrix $\boldsymbol{X}$ and a vector $\boldsymbol{y}$, we saw that we could *guess* the correct $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^\top$ by solving the *least-squares* optimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2.$$

By now, we know how to solve this problem using gradient descent.



Example 6.3. In text classification, the task is to decide to which of a given set of categories a given text belongs. The training data consists of a *bag of words*: this is a large sparse matrix, whose columns represent words and the rows represent articles, with the (i, j)-th entry containing the number of times word j is contained in text i.

	Rooney	Boris
Article 1	5	0
Article 2	1	7

For example, in the above set we would classify the first article as "Sports" and the second one as "Politics". One such training dataset is the Reuters Corpus Volume I (RCV1), an archive of over 800,000 categorised newswire stories.

A typical binary classifier for such a problem would be a **linear classifier** of the form

$$h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} - \tau,$$

with $\boldsymbol{w} \in \mathbb{R}^n$ and $\tau \in \mathbb{R}$. Given a text, represented as a row of the dataset $\boldsymbol{x}$, it is classified into one of two classes $\{+1, -1\}$, depending on whether $h(\boldsymbol{x}) > 0$ or $h(\boldsymbol{x}) < 0$.

Suppose we have the training data $\{\boldsymbol{x}_i, y_i\}_{1 \le i \le m}$ and we found a candidate function $h\colon \mathcal{X} \to \mathcal{Y}$. How do we assess if this is a good fit? One usually assigns to the problem a **loss function** $\ell\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ that measures the mismatch of a prediction. One would then aim to find a function h among a set of candidates that minimizes the loss when applying the function to the training data:

$$\underset{h}{\text{minimize}} \quad \frac{1}{m} \sum_{i=1}^{m} \ell(h(\boldsymbol{x}_i, y_i).$$

The form of the loss function depends on the problem at hand, but two typical candidates are the **square error** for regression problems,

$$\ell(h(\boldsymbol{x}), y) = (h(\boldsymbol{x}) - y)^2,$$

and the indicator loss function

$$\mathbf{1}\{h(\boldsymbol{x}) \neq y\},$$

where $\mathbf{1}\{A\}$ is the indication function, which takes the value 1 if A is true, and 0 else. As this function is not continuous, in practice one often encounters the *log loss* function,

$$\ell(h(\boldsymbol{x}), y) = \log\left(1 + e^{-h(\boldsymbol{x})y}\right).$$

Note that if $h(\boldsymbol{x})$ and $\boldsymbol{y}$ have the same sign (corresponding to a match), then the value of this function will be close to zero.

Suppose we have a binary classification task at hand, with $\mathcal{Y} = \{-1, 1\}$. We could *learn* the following function from our data:

$$h(\boldsymbol{x}) = \begin{cases} y_i & \text{if } \boldsymbol{x} = \boldsymbol{x}_i, \\ 1 & \text{otherwise.} \end{cases}$$

The **empirical misclassification risk** in for this problem is then 0,

$$R(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{h(\boldsymbol{x}_i \neq y_i\} = 0.$$

Nevertheless, this is not a good classifier: it will not perform very well outside of the training set. This is an example of **overfitting**: when the function is adapted too closely to the seen data. To remedy this, one often (randomly) splits the available data into a **training set** and a **test set**. The training set is used to find the function h, while the test set is for testing how good it is (in practise, one often adds a validation set, used to tune some parameters of the problem).

Example 6.4. In the linear regression problem with the quadratic loss function, we end up with the minimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i^\top \boldsymbol{\beta} - y_i)^2 = \frac{1}{n}\|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2.$$

This is precisely the problem encountered in Lecture 1.

Example 6.5. In the example of classifying a text into two classes using a linear classifier, we can look at the problem

$$\underset{\boldsymbol{w},\tau}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{m} \mathbf{1}\{\boldsymbol{w}^\top \boldsymbol{x}_i - \tau \neq y_i\}.$$

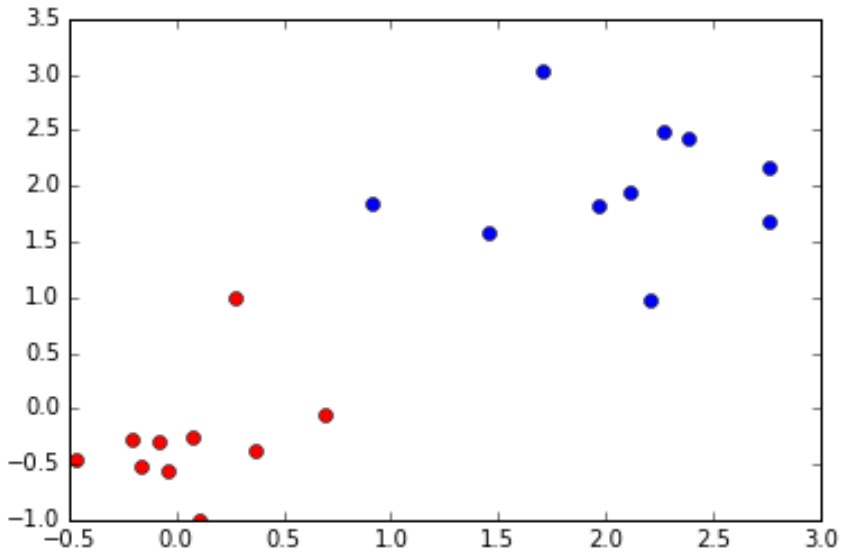
Since this function is not smooth or even continuous, we can work with the log-loss function described above,

$$\underset{\boldsymbol{w},\tau}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{m} \ell(\boldsymbol{w}^\top \boldsymbol{x}_i - \tau, y_i).$$

This function can, in principle, be minimized using gradient descent or Newton's method.

In the following example, we use this classifier to separate an artificial data set of 20 points. 10 points are generated as normal (Gaussian) distributed vectors in $\mathbb{R}^2$, with mean $(2,2)^\top$ and variance 0.25, while the other 10 points are generated with mean $(0,0)^\top$ and the same variance.

In [17]:
```
X = np.zeros((2,20))
for i in range(10):
    X[:,i] = np.array([2,2]) + 0.5*rnd.randn(2)
for i in range(10):
    X[:,10+i] = np.array([0,0]) + 0.5*rnd.randn(2)

% matplotlib inline
plt.plot(X[0,0:10], X[1,0:10], 'o')
plt.plot(X[0,10:], X[1,10:], 'o', color='red')
plt.show()
```



We next minimize the *log-loss function*,

$$F(\boldsymbol{w}) = \sum_{i=1}^{20} \log(1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - \tau)}),$$

where the $\boldsymbol{x}_i \in \mathbb{R}^2$ are the points, and the $y_i \in \{-1, 1\}$ classify these as either red or blue. As minimization algorithm we choose gradient descent with backtracking, though any other reasonable algorithm would do.

In [18]:
```
def graddesc_bt(f, df, x0, tol, maxiter=100, rho=0.5, c=0.1):
    """
    Gradient descent with backtracking
    """
    x = np.vstack((x0+2*tol*np.ones(x0.shape),x0)).transpose()
    i = 1
    while ( la.norm(x[:,i]-x[:,i-1]) > tol ) and ( i < maxiter ):
        p = -df(x[:,i])
        # Start backtracking
        alpha = 1
        xnew = x[:,i] + alpha*p
        while (f(xnew) >= f(x[:,i]) + alpha*c*np.dot(p, df(x[:,i]))):
            alpha = alpha*rho
            xnew = x[:,i] + alpha*p
        x = np.concatenate((x,xnew.reshape((len(x0),1))), axis=1)
        i += 1
    return x[:,1:]
```

In [19]:
```
y = np.concatenate((-np.ones(10),np.ones(10)))
X = np.concatenate((X, np.ones((1,20))), axis=0)

def f(w):
    return np.sum(np.log(1+np.exp(-y*(np.dot(w,X)))))

def df(w):
    return np.array([-np.sum(y*X[0,:]*np.exp(-y*np.dot(w,X))/
                                    (1+np.exp(-y*np.dot(w,X)))),
                     -np.sum(y*X[1,:]*np.exp(-y*np.dot(w,X))/
                     (1+np.exp(-y*np.dot(w,X)))),
                     -np.sum(y*X[2,:]*np.exp(-y*np.dot(w,X))/
                     (1+np.exp(-y*np.dot(w,X))))])

W = graddesc_bt(f, df, np.array([1.,1.,1.]), 1.e-8)
```

The algorithm gives as output a vector $\boldsymbol{w} = (w_1, w_2, w_3) \in \mathbb{R}^3$, so that the line separating the two sets of points is given by

$$w_1 x_1 + w_2 x_2 + w_3 = 0.$$

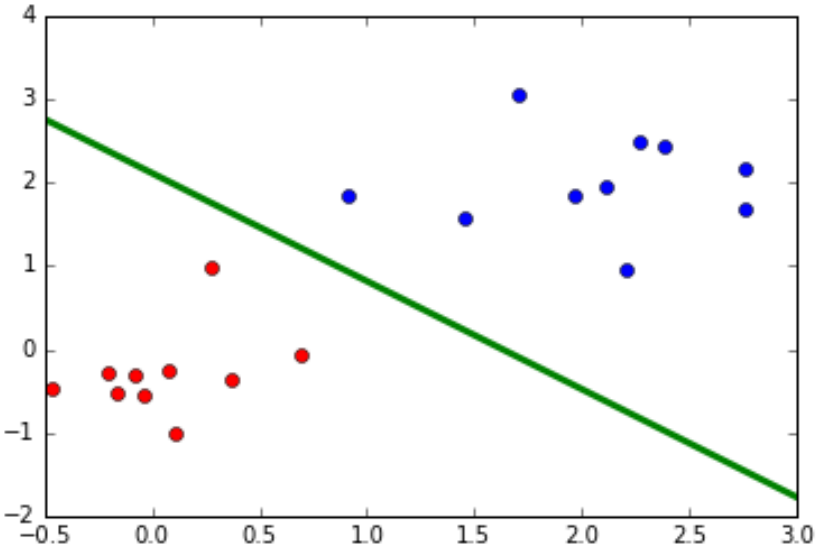
Rearranging this as a function $\ell(x_1) = x_2 = -(w_1 x_1 + w_3)/w_2$ and plotting the line into the pointcloud, we get the following graph.

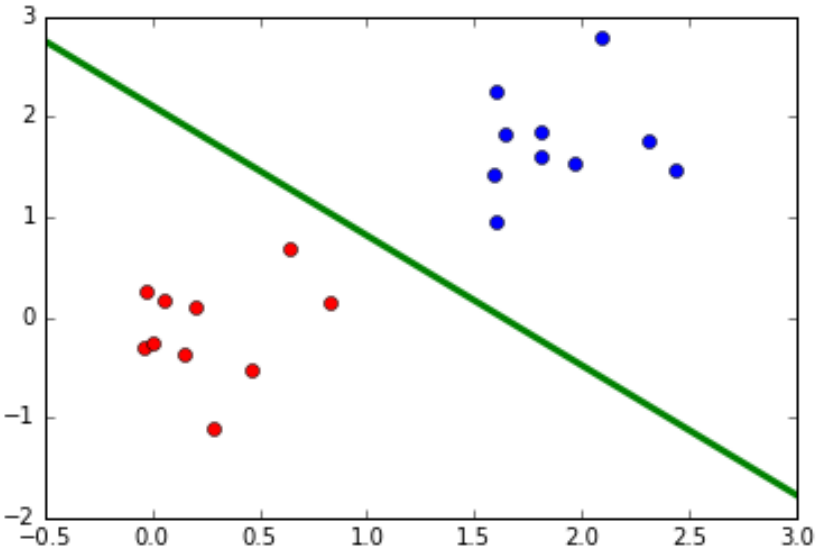
In [20]:
```
def l(x):
    return (-W[0,-1]*x-W[2,-1])/W[1,-1]

xx = np.linspace(-0.5,3,100)
yy = l(xx)
plt.plot(X[0,0:10], X[1,0:10], 'o')
plt.plot(X[0,10:], X[1,10:], 'o', color='red')
plt.plot(xx,yy,linewidth=3)
plt.show()
```

Of course, we would like to know how our classifier does with a *test set* generated according to the same distribution. The result is seen the following plot.

The given example is unproblematic, since the two sets of points are generated in a way that makes it unlikely that they can't be separated using a linear classifier. With real data, the situation may not be as favourable.

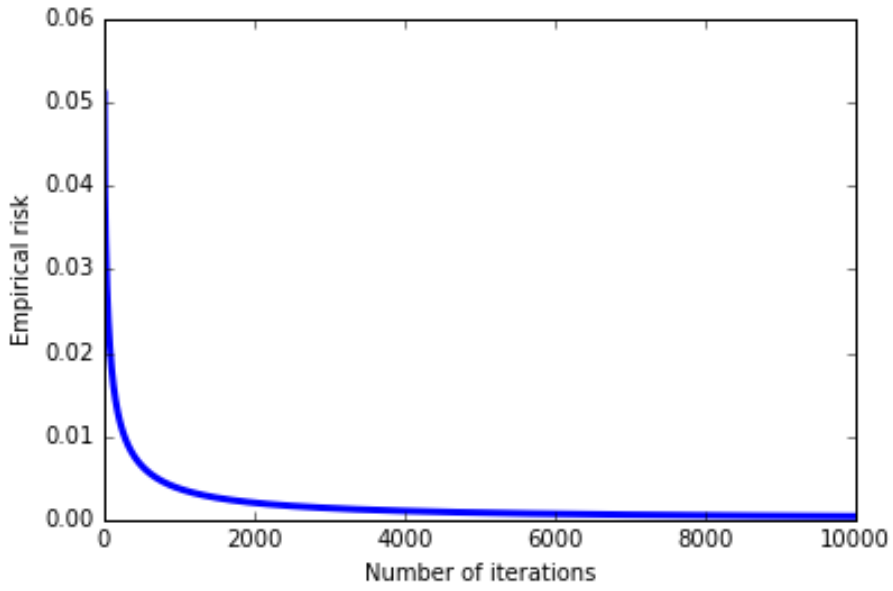
It is also interesting to see how fast the objective function (the *empirical risk*) approaches the minimum. While the value of the objective function decreases very quickly at the beginning, the





decrease slows down dramatically and the algorithm does not reach the desired accuracy within the limit we set for the number of iterations. Now, however, that for the classification task we are interested, we don't need the best possible precision! In fact, the noise in the data makes too much precision meaningless, and we can safely use the result obtained after 100 iterations. Other algorithms and approaches may perform better for this problem.

In [21]:
```
risk = np.zeros(W.shape[1])
for i in range(W.shape[1]):
    risk[i] = f(W[:,i])

plt.plot(range(W.shape[1]), risk, linewidth=3)
plt.xlabel('Number of iterations')
plt.ylabel('Empirical risk')
plt.show()
```



We have seen how a machine learning problem can be turned into an optimization problem. Unfortunately, when the amount of data is large, computing gradients and Hessians of the resulting functions can be very expensive. Suppose the function h depends on parameters $\boldsymbol{w}$ which we want to optimize over, $h = h(\boldsymbol{x}; \boldsymbol{w})$, and denote the composition of the loss function ℓ with h on the data $\boldsymbol{x}_i$ by

$$f_i(\boldsymbol{w}) := \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), y_i).$$

We can then write our optimization problem as

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad F(\boldsymbol{w}), \quad F(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} f_i(\boldsymbol{w}).$$

The gradient is then

$$\nabla F(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{w}).$$

For large datasets (for example, $m > 10^6$ is not uncommon), computing all the gradients in each step is highly ineffective.

An old algorithm that has recently been revived in light of big data is **stochastic gradient descent**. The idea is to compute, at each step, only the gradient of *one* (or sometimes a few) of the summands of the loss function:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{w}_k),$$

where the index i_k is *chosen at random* from $\{1, \ldots, m\}$ at each step.

Each step of this algorithm is clearly faster than gradient descent, but does it even work? It turns out that this algorithm converges quickly and that it is surprisingly effective on large datasets.

Part III

Linear Programming

Lecture 7

"[A]lmost every "convex" idea can be explained by a two-dimensional picture."

— Alexander Barvinok

In this lecture we begin our study of the theory underlying constrained convex optimization. One way to define a *convex optimization problem* is

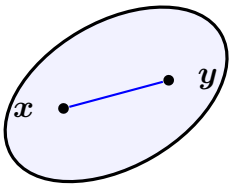
$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & f_1(\boldsymbol{x}) \leq 0 \\ & \cdots \\ & f_m(\boldsymbol{x}) \leq 0 \\ & \boldsymbol{x} \in \Omega \end{array}$$

where $f, f_1, \ldots, f_m \colon \mathbb{R}^n \to \mathbb{R}$ are *convex* functions and $\Omega \subseteq \mathbb{R}^n$ is a *convex* set. The special case where the f and the f_i are linear functions and $\Omega = \mathbb{R}^n$ is known as linear programming, and is studied first. Before embarking on the study of models and algorithms for convex optimization, we need to study convex sets in more depth.

7.1 Convex sets

We recall the definition of a convex set.

Definition 7.1. A set $C \subseteq \mathbb{R}^n$ is a *convex set*, if for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda \in [0,1]$, $\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in C$. In words, for every two points in C, the line joining them is also in C. A compact (closed and bounded) convex set is called a *convex body*.



We will denote by $\mathcal{C}(\mathbb{R}^n)$ the collection of convex sets and by $\mathcal{K}(\mathbb{R}^n)$ the collection of convex bodies. The following Lemma is left as an exercise.

Lemma 7.2. *Let $C, D \in \mathcal{C}(\mathbb{R}^n)$ be convex sets. Then the following are also convex.*

- $C \cap D$;
- $C + D = \{\boldsymbol{x} + \boldsymbol{y} : \boldsymbol{x} \in C, \boldsymbol{y} \in D\}$;
- $\boldsymbol{A}C = \{\boldsymbol{A}\boldsymbol{x} : \boldsymbol{x} \in C\}$, *where* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.

The *convex hull* $\operatorname{conv} S$ of a set S is the intersection of all convex sets containing S. Clearly, if S is convex, then $S = \operatorname{conv} S$.

Example 7.3. Let $S = \{(1,1)^\top, (1,-1)^\top, (-1,1)^\top, (-1,-1)^\top, (0,0)^\top\}$. The convex hull of this set is the square.

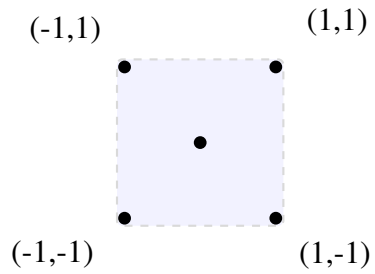


Figure 7.8: A convex hull of five points.

A *convex combination* of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ is a linear combination

$$\sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i$$

such that $\lambda_i \geq 0$ and $\sum_{i=1}^{k} \lambda_i = 1$. It can be shown inductively that convex sets are closed under convex combinations: any convex combination of points in $C \in \mathcal{C}(\mathbb{R}^n)$ is still in C. In fact, the set of all convex combinations of points in a set S is the convex hull of S.

Lemma 7.4. *Let S be a set. Then*

$$\operatorname{conv} S = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_k,\ \boldsymbol{x}_i \in S,\ \sum_{i=1}^{k} \lambda_i = 1,\ \lambda_i \geq 0\}.$$

Example 7.5. A hyperplane, defined as the solution set of one linear equation,

$$H = \{\boldsymbol{x} : \langle \boldsymbol{a}, \boldsymbol{x} \rangle = b\},$$

is a convex set. Define the halfspaces H_+ and H_- as the two sides that H divides $\mathbb{R}^n$ into:

$$H_- = \{\boldsymbol{x} : \langle \boldsymbol{a}, \boldsymbol{x} \rangle \leq b\}, \quad H_+ = \{\boldsymbol{x} : \langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq b\}$$

These are also convex sets.

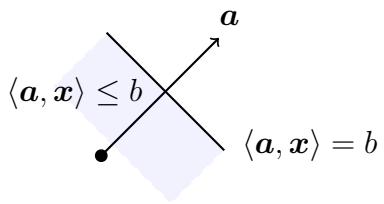


Figure 7.9: Hyperplane and halfspace

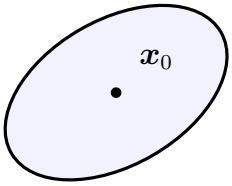


Figure 7.10: An ellipse

Example 7.6. Euclidean balls and ellipsoids are common examples of convex sets. Let $\boldsymbol{P}$ be a positive semidefinite symmetric matrix. Then an ellipsoid with center $\boldsymbol{x}_0$ is a set of the form

$$\mathcal{E} = \{\boldsymbol{x} : \langle \boldsymbol{x} - \boldsymbol{x}_0, \boldsymbol{P}^{-1}(\boldsymbol{x} - \boldsymbol{x}_0) \rangle \leq 1\}.$$

A Euclidean unit ball is the special case $\boldsymbol{P} = \boldsymbol{I}$.

Example 7.7. A *convex cone* is a set C such that for all $\boldsymbol{x}, \boldsymbol{y}$ and $\lambda \geq 0, \mu \geq 0$, $\lambda \boldsymbol{x} + \mu \boldsymbol{y} \in C$. It is easily verified that such a set is convex. Three important cones are the following:

1. The non-negative orthant $\mathbb{R}^n_+ = \{\boldsymbol{x} \in \mathbb{R}^n : x_i \geq 0, 1 \leq i \leq n\}$,
2. The second order (ice cream) cone (or Lorentz cone)

$$C_\alpha = \{\boldsymbol{x} : \sum_{i=1}^{n-1} x_i^2 \leq x_n^2\},$$

3. The cone $\mathcal{S}^n_+$ of positive semidefinite symmetric matrices.

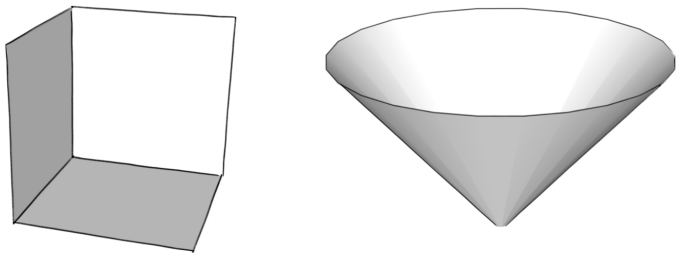


Figure 7.11: The orthant and the second order cone

Possibly the most important result in convex geometry is the *hyperplane separation theorem*. We first need the following.

Lemma 7.8. *Let C be a non-empty convex set and $\boldsymbol{x} \notin C$. Then there exists a point $\boldsymbol{y} \in C$ that minimizes the distance $\|\boldsymbol{x} - \boldsymbol{y}\|$. Moreover, for all $\boldsymbol{z} \in C$ we have*

$$\langle \boldsymbol{z} - \boldsymbol{y}, \boldsymbol{x} - \boldsymbol{y} \rangle \leq 0.$$

In words, the vectors $\boldsymbol{z} - \boldsymbol{y}$ and $\boldsymbol{x} - \boldsymbol{y}$ form an obtuse angle.

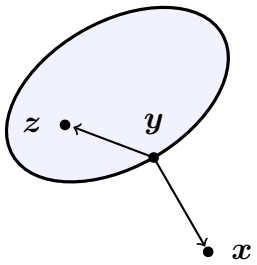


Figure 7.12: Internal and external directions

Proof. Since $C \neq \emptyset$, there exists $r > 0$ such that the ball $B(\boldsymbol{x}, r) := \{\boldsymbol{y} \in \mathbb{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\| \leq \varepsilon\}$ intersected with C is not empty. Since $K := C \cap B(\boldsymbol{x}, r)$ is compact (closed and bounded) and the function $\|\boldsymbol{y} - \boldsymbol{x}\|$ is continuous on K, it has a minimizer $\boldsymbol{y} \in K$. For the second claim, note that since C is convex, for every $\lambda \in [0, 1]$,

$$\boldsymbol{w} = \lambda \boldsymbol{z} + (1 - \lambda)\boldsymbol{y} \in C.$$

For the distance between $\boldsymbol{z}$ and $\boldsymbol{x}$ we then get

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

We now prove the claim by contradition. Assume $\langle \boldsymbol{z} - \boldsymbol{y}, \boldsymbol{x} - \boldsymbol{y} \rangle > 0$. Then we can choose λ such that

$$0 < \lambda < \min \left\{ \frac{2\langle \boldsymbol{x} - \boldsymbol{y}, \boldsymbol{z} - \boldsymbol{y} \rangle}{\|\boldsymbol{z} - \boldsymbol{y}\|^2}, 1 \right\}.$$

With such a λ we get

$$\|\boldsymbol{w} - \boldsymbol{x}\|^2 = \lambda^2 \|\boldsymbol{z} - \boldsymbol{y}\|^2 - 2\lambda \langle \boldsymbol{z} - \boldsymbol{y}, \boldsymbol{x} - \boldsymbol{y} \rangle + \|\boldsymbol{x} - \boldsymbol{y}\|^2 < \|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

This inequality, however, contradicts the assumption that $\boldsymbol{y}$ is a closest point, so that $\langle \boldsymbol{z} - \boldsymbol{y}, \boldsymbol{x} - \boldsymbol{y} \rangle \leq 0$ has to hold. □

In what follows write $\mathrm{int} S$ for the *interior* of a set S.

Theorem 7.9. *Let C be a closed convex set and $\boldsymbol{x} \notin C$. Then there exists a hyperplane H such that $C \subset \mathrm{int} H_-$ and $\boldsymbol{x} \in \mathrm{int} H_+$.*

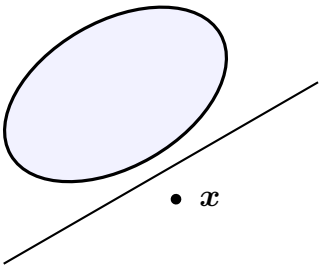


Figure 7.13: A separating hyperplane

Proof. Let $\boldsymbol{y} \in C$ be a nearest point to $\boldsymbol{x}$ in C, i.e., a point such that for all other $\boldsymbol{z} \in C$, $\|\boldsymbol{x} - \boldsymbol{y}\| \leq \|\boldsymbol{x} - \boldsymbol{z}\|$. Define

$$\boldsymbol{a} = \boldsymbol{x} - \boldsymbol{y}, \quad b = (\|\boldsymbol{x}\|^2 - \|\boldsymbol{y}\|^2)/2.$$

We aim to show that $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ defines a separating hyperplane.

For this we have to show that

1. $\langle \boldsymbol{a}, \boldsymbol{x} \rangle > b$;
2. For all $\boldsymbol{z} \in C$, $\langle \boldsymbol{a}, \boldsymbol{z} \rangle < b$.

For (1), note that

$$\langle \boldsymbol{a}, \boldsymbol{x} \rangle = \langle \boldsymbol{x} - \boldsymbol{y}, \boldsymbol{x} \rangle > \langle \boldsymbol{x} - \boldsymbol{y}, \boldsymbol{x} \rangle - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 = \frac{1}{2}(\|\boldsymbol{x}\|^2 - \|\boldsymbol{y}\|^2) = b.$$

To prove (2), assume on the contrary that there exists a $\boldsymbol{z} \in C$ such that $\langle \boldsymbol{a}, \boldsymbol{z} \rangle \geq b$. We know that the point $\boldsymbol{y} \in C$ satisfies the inequality (2), since

$$\langle \boldsymbol{a}, \boldsymbol{y} \rangle < \langle \boldsymbol{a}, \boldsymbol{y} \rangle + \frac{1}{2}\|\boldsymbol{a}\|^2 = \langle \boldsymbol{a}, \boldsymbol{y} \rangle + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 = b.$$

Therefore,

$$\langle \boldsymbol{a}, \boldsymbol{z} - \boldsymbol{y} \rangle = \langle \boldsymbol{a}, \boldsymbol{z} \rangle - \langle \boldsymbol{a}, \boldsymbol{y} \rangle > b - b = 0,$$

but this contradicts Lemma 7.8. We therefore conclude $\langle \boldsymbol{a}, \boldsymbol{z} \rangle < b$. The separating hyperplane H is thus defined by the equation $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$. □

Lecture 8

Linear programming is about problems of the form

$$\begin{aligned} &\text{maximize} && \langle \boldsymbol{c}, \boldsymbol{x} \rangle \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \end{aligned}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{c} \in \mathbb{R}^n$, and $\boldsymbol{b} \in \mathbb{R}^m$, and the inequality sign means inequality in each row. The *feasible set* is the set of all possible candidates,

$$\mathcal{F} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}.$$

This set can be empty (example: $x \leq 1,\ -x \leq -2$), unbounded (example: $x \leq 1$) or bounded (example: $x \leq 1, -x \leq 0$). In any case, it is a convex set. To understand linear programming it is of paramount importance to understand the geometry of the feasible sets of linear programming, also called *polyhedra*.

8.1 Linear Programming Duality: a first glance

Suppose we are faced with a linear programming problem and would like to know if the feasible set $\mathcal{F}$ is empty or not, i.e., if $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ has a solution. If it is not empty, we can certify that by producing a vector from $\mathcal{F}$. To verify that the feasible set is empty is more tricky: we are asked to show that *no* vector lives in $\mathcal{F}$. What we can try to do, however, is to show that the assumption of a solution to $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ would lead to a contradiction. Denote by $\boldsymbol{a}_i^\top$ the rows of $\boldsymbol{A}$. Assuming $\boldsymbol{x} \in \mathcal{F}$, then given a vector $\mathbf{0} \neq \boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^\top$ with $\lambda_i \geq 0$, the linear combination satisfies

$$\sum_{i=1}^m \lambda_i \boldsymbol{a}_i^\top \boldsymbol{x} \leq \sum_{i=1}^m \lambda_i b_i = \langle \boldsymbol{\lambda}, \boldsymbol{b} \rangle. \tag{8.1}$$

If we can find parameters $\boldsymbol{\lambda}$ such that the left-hand side of (8.1) is identically 0 and the right-hand side is strictly negative, then we have found a contradition and can conclude that no $\boldsymbol{x}$ satisfies $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. A condition that ensures this is

$$\sum_{i=1}^m \lambda_i \boldsymbol{a}_i = \mathbf{0}, \quad \langle \boldsymbol{\lambda}, \boldsymbol{b} \rangle < 0. \tag{8.2}$$

In matrix form,

$$\exists \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{A}^\top \boldsymbol{\lambda} = \mathbf{0},\ \langle \boldsymbol{\lambda}, \boldsymbol{b} \rangle < 0.$$

This condition will still be satisfied if we normalise the vector $\boldsymbol{\lambda}$ such that $\sum_{i=1}^{m} \lambda_i = 1$, so the statement says that $\mathbf{0}$ is a convex combination of the vectors defining the equations.

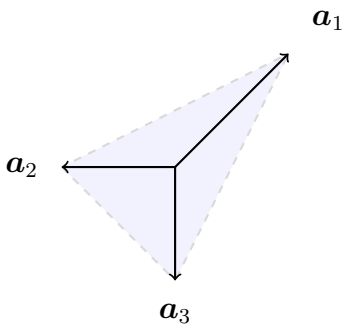
Example 8.1. Consider the system

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ -x_1 &\leq -1 \\ -x_2 &\leq -1.5. \end{aligned} \tag{8.3}$$

The transpose matrix $\boldsymbol{A}^\top$ and the vector $\boldsymbol{b}$ are

$$\boldsymbol{A}^\top = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 2 \\ -1 \\ -1.5. \end{pmatrix}$$

Drawing the columns of $\boldsymbol{A}^\top$ we get We can get the origin as a convex combination of the



vectors $\boldsymbol{a}_i$ (drawing a rope around them encloses the origin), and such a combination is given by $\boldsymbol{\lambda} = (1/3, 1/3, 1/3)$. Taking the scalar product with the vector $\boldsymbol{b}$ we get

$$\langle \boldsymbol{\lambda}, \boldsymbol{b} \rangle = \frac{1}{3}(2 - 1 - 1.5) = -\frac{1}{6} < 0.$$

This shows that the system (8.3) does not have a solution (a fact that in this simple example can also be seen by drawing a picture).

It turns out that Condition (D) is not only sufficient but also necessary, and the separating hyperplane theorem is an essential part of this. We first make a detour in order to better understand the feasible sets, the polyhedra.

8.2 Polyhedra

Definition 8.2. A polyhedron (plural: polyhedra) is a set defined as the solution of linear equalities and inequalities,

$$P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}, \tag{8.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$.

More classically, we can write out the equations.

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_d &\leq b_1, \\ &\cdots \\ a_{m1}x_1 + \cdots + a_{mn}x_d &\leq b_m. \end{aligned} \tag{8.2}$$

We now introduce some useful terminology and concepts associated to polyhedra, and illustrate them with a few examples. A supporting hyperplane H of a polyhedron P is a hyperplane such that $P \subseteq H_-$, where H_- is a halfspace associated to H. If H is a supporting hyperplane, then a set of the form $F = H \cap P$ is called a *face* of P. In particular, the polyhedron P is a face. Each of the inequalities $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle \leq b_i$ in (8.1) defines a supporting hyperplane, and therefore a face. The *dimension* of a face F, $\dim F$, is the smallest dimension of an affine space containing F. Faces of dimension $\dim F = \dim P - 1$ are called *facets*, faces of dimension 0 are vertices, and of dimension 1 edges. A vertex can equivalently be characterised as a point $\boldsymbol{x} \in P$ that can not be written as a convex combination of two other points in P.

Example 8.3. Polyhedra in one dimension are the sets $[a, b]$, $[a, \infty)$, $(-\infty, b]$, $\mathbb{R}$ or $\emptyset$, where $a \leq b$. Each of them is clearly convex.

Example 8.4. The set

$$P = \{\boldsymbol{x} \in \mathbb{R}^2 : x_1 + x_2 \leq 1,\ x_1 \geq 0,\ x_2 \geq 0\}.$$

is the polyhedron shown in Figure 8.14. We can write the defining inequalities in standard form $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ by setting

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

This polyhedron has one face of dimension 2 (itself), three facets of dimension 1 (the sides, corresponding to the three equations), and three vertices of dimension 0 (the corners, corresponding any two of the defining equations).

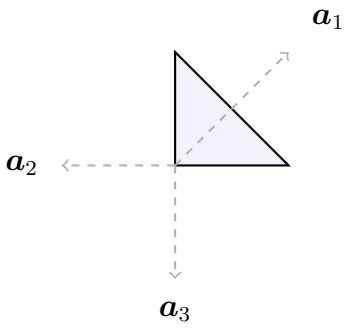


Figure 8.14: A two-dimensional polyhedron and defining equations.

Example 8.5. The polyhedron in Albrecht Dürer's famous "Melencolia I", aka the "truncated triangular trapezohedron", can be described using eight inequalities, see Figure 8.15.

$$
\begin{aligned}
0.7071x_1 - 0.4082x_2 + 0.3773x_3 &\leq 1\\
-0.7071x_1 + 0.4082x_2 - 0.3773x_3 &\leq 1\\
0.7071x_1 + 0.4082x_2 - 0.3773x_3 &\leq 1\\
-0.7071x_1 - 0.4082x_2 + 0.3773x_3 &\leq 1\\
0.8165x_2 + 0.3773x_3 &\leq 1\\
-0.8165x_2 - 0.3773x_3 &\leq 1\\
0.6313x_3 &\leq 1\\
-0.6313x_3 &\leq 1
\end{aligned}
$$

Figure 8.15: Dürer's Melencolia and equations defining the polyhedron

We now move to a different characterization of bounded polyhedra. The main result of this lecture is that bounded polytopes can be described completely from knowing their vertices. A polyhedron P is called bounded, if there exists a ball $B(\mathbf{0}, r)$ with $r > 0$ such that $P \subset B(\mathbf{0}, r)$. For example, halfspaces are not bounded, but the polytope from Examples 8.4 and 8.5 are.

We first observe the nontrivial fact that a polyhedron has only fin

Definition 8.6. A *polytope* is the convex hull of finitely many points,

$$P = \text{conv}(\{x_1, \ldots, x_k\}) = \{\sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i,\ \lambda_i \geq 0, \sum_i \lambda_i = 1\}.$$

Theorem 8.7. *A bounded polyhedron P is the convex hull of its vertices.*

Example 8.8. The triangle in Example 8.4 is the convex hull of the points $(0,0)^\top$, $(0,1)^\top$, and $(1,0)^\top$.

Example 8.9. The Dürer polytope is the convex hull of the following 12 vertices:

$$
\begin{aligned}
\boldsymbol{v}_1 &= \begin{pmatrix} -1.4142 \\ -0.8165 \\ -0.8835 \end{pmatrix}, \boldsymbol{v}_2 = \begin{pmatrix} -1.4142 \\ 0.8165 \\ 0.8835 \end{pmatrix}, \boldsymbol{v}_3 = \begin{pmatrix} -0.8536 \\ -0.4928 \\ -1.5840 \end{pmatrix}, \boldsymbol{v}_4 = \begin{pmatrix} -0.8536 \\ 0.4928 \\ 1.5840 \end{pmatrix},\\
\boldsymbol{v}_5 &= \begin{pmatrix} -0.0000 \\ -1.6330 \\ 0.8835 \end{pmatrix}, \boldsymbol{v}_6 = \begin{pmatrix} 0.0000 \\ -0.9856 \\ 1.5840 \end{pmatrix}, \boldsymbol{v}_7 = \begin{pmatrix} -0.0000 \\ 0.9856 \\ -1.5840 \end{pmatrix}, \boldsymbol{v}_8 = \begin{pmatrix} 0.0000 \\ 1.6330 \\ -0.8835 \end{pmatrix},\\
\boldsymbol{v}_9 &= \begin{pmatrix} 0.8536 \\ -0.4928 \\ -1.5840 \end{pmatrix}, \boldsymbol{v}_{10} = \begin{pmatrix} 0.8536 \\ 0.4928 \\ 1.5840 \end{pmatrix}, \boldsymbol{v}_{11} = \begin{pmatrix} 1.4142 \\ -0.8165 \\ -0.8835 \end{pmatrix}, \boldsymbol{v}_{12} = \begin{pmatrix} 1.4142 \\ 0.8165 \\ 0.8835 \end{pmatrix}.
\end{aligned}
$$

The converse of Theorem 8.7 is also true.

Theorem 8.10. *A polytope is a bounded polyhedron.*

The equivalence between polytopes and bounded polyhedra gives a first glimpse into linear programming duality theory, a topic of central importance in both modeling and algorithm design.

Lecture 9

Linear programming duality associates to a linear programming problem a **dual problem**, with the property that the optimal values of the original and of the dual problem coincide. Duality is an important tool in applications and in the design of algorithms. Linear programming duality rests upon an important family of results in convex geometry, known collectively as Farkas' Lemma.

9.1 Farkas' Lemma

Recall the definition of a convex cone. This is a set C such that for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda_1, \lambda_2 \geq 0$ we have $\lambda_1 \boldsymbol{x} + \lambda_2 \boldsymbol{y} \in C$.

Lemma 9.1. *(Hyperplane separation for cones) Let $C \neq \mathbb{R}^n$ be a closed convex cone and $\boldsymbol{z} \notin C$. Then there exists a linear hyperplane such that $C \subseteq H_-$ and $\boldsymbol{z} \in \mathrm{int} H_+$.*

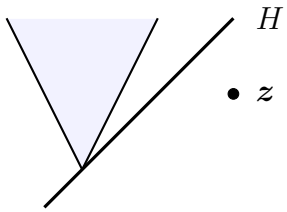


Figure 9.16: Separating hyperplane for a cone

Proof. From Lecture 7 we know that there exists an affine hyperplane H^a separating C and $\boldsymbol{z}$. Let this affine hyperplane be given by $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$. We would like to show that $H = \{\boldsymbol{x} : \langle \boldsymbol{a}, \boldsymbol{x} \rangle = 0\}$ is a linear hyperplane separating C and $\boldsymbol{z}$. From $\langle \boldsymbol{a}, \boldsymbol{z} \rangle > b$ we clearly get $\langle \boldsymbol{a}, \boldsymbol{z} \rangle > 0$. Also, $\mathbf{0} \in H_-$, since $\langle \boldsymbol{a}, \mathbf{0} \rangle = 0$. Assume now that there exists a point $\boldsymbol{x} \in C$ such that $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = c > 0$. Since C is a cone, for all $\lambda > 0$ we have that $\lambda \boldsymbol{x} \in C$. Choosing λ so that $\lambda > b/c$ we get

$$\langle \boldsymbol{a}, \lambda \boldsymbol{x} \rangle = \lambda c > b,$$

in contradiction to $C \subset H^a_-$. This shows that H is a linear separating hyperplane. □

Theorem 9.2. *(Farkas' Lemma) Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$, there exists a vector $\boldsymbol{x}$ such that*

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq 0$$

if and only if there is not $\boldsymbol{y} \in \mathbb{R}^m$ such that

$$\boldsymbol{A}^\top \boldsymbol{y} \geq \boldsymbol{0}, \quad \langle \boldsymbol{y}, \boldsymbol{b}\rangle < 0.$$

Proof. Assume $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ has a solution $\boldsymbol{x} \geq 0$. Then for any $\boldsymbol{y} \neq \boldsymbol{0}$ such that $\boldsymbol{A}^\top \boldsymbol{y} \geq \boldsymbol{0}$,

$$0 \leq \langle \boldsymbol{A}^\top \boldsymbol{y}, \boldsymbol{x}\rangle = \langle \boldsymbol{y}, \boldsymbol{A}\boldsymbol{x}\rangle = \langle \boldsymbol{y}, \boldsymbol{b}\rangle,$$

which shows that $\boldsymbol{A}^\top \boldsymbol{y} \geq \boldsymbol{0}$ and $\langle \boldsymbol{y}, \boldsymbol{b}\rangle < 0$ are not simultaneously possible.

Assume now that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ has no solution that satisfies $\boldsymbol{x} \geq 0$. Let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ be the columns of $\boldsymbol{A}$. The set of $\boldsymbol{A}\boldsymbol{x}$ for $\boldsymbol{x} \geq \boldsymbol{0}$ is the set of all nonnegative linear combinations

$$C = \{\boldsymbol{z} \in \mathbb{R}^m : \boldsymbol{z} = x_1\boldsymbol{a}_1 + \cdots + x_n\boldsymbol{a}_n,\ x_i \geq 0\},$$

and this set is a convex cone. The assumption that there is no nonnegative $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ means that $\boldsymbol{b} \notin C$. By Lemma 9.1, there exists a linear hyperplane $H = \{\boldsymbol{x} : \langle \boldsymbol{y}, \boldsymbol{x}\rangle = 0\}$ such that $C \in H_-$ and $\boldsymbol{b} \in \mathrm{int}H_+$. Formulated differently, there exists a $\boldsymbol{y} \in \mathbb{R}^m$ such that

$$\forall \boldsymbol{z} \in C\colon \langle \boldsymbol{z}, \boldsymbol{y}\rangle \geq 0, \quad \langle \boldsymbol{b}, \boldsymbol{y}\rangle < 0.$$

Since every $\boldsymbol{z} \in C$ has the form $\boldsymbol{z} = \sum_{i=1}^n x_i\boldsymbol{a}_i$ with $x_i \geq 0$, the relation

$$\langle \boldsymbol{z}, \boldsymbol{y}\rangle = \sum_{i=1}^n x_i \langle \boldsymbol{a}_i, \boldsymbol{y}\rangle \geq 0$$

for all $\boldsymbol{x} \geq 0$ is equivalent to the condition that $\langle \boldsymbol{a}_i, \boldsymbol{y}\rangle \geq 0$ for $1 \leq i \leq n$, which again is equivalent to $\boldsymbol{A}^\top \boldsymbol{y} \geq \boldsymbol{0}$. This concludes the proof. □

The following consequence is perhaps a more familiar form of Farkas' Lemma.

Corollary 9.3. *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$, there exists a vector $\boldsymbol{x} \neq \boldsymbol{0}$ such that*

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$

if and only there is no $\boldsymbol{y}$ such that

$$\boldsymbol{y} \geq \boldsymbol{0}, \quad \boldsymbol{A}^\top \boldsymbol{y} = \boldsymbol{0}, \quad \langle \boldsymbol{y}, \boldsymbol{b}\rangle < 0.$$

Proof. Consider the matrix

$$\boldsymbol{A}' := \begin{pmatrix} \boldsymbol{A} & -\boldsymbol{A} & \boldsymbol{I} \end{pmatrix},$$

where $\boldsymbol{I}$ is the $m \times m$ identity matrix. A nonnegative solution of $\boldsymbol{A}'\boldsymbol{x}' = \boldsymbol{b}$ has the form $\boldsymbol{x}' = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)^\top$, and implies $\boldsymbol{A}(\boldsymbol{x}_1 - \boldsymbol{x}_2) + \boldsymbol{x}_3 = \boldsymbol{b}$. Therefore, such a solution $\boldsymbol{x}'$ exists if and only if the system

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$

has a solution. Applying Theorem 9.2, the complementary condition is

$$\boldsymbol{A}'^{\top}\boldsymbol{y} \geq \boldsymbol{0}, \quad \langle \boldsymbol{b}, \boldsymbol{y} \rangle < 0,$$

which in terms of $\boldsymbol{A}$ translates to

$$\boldsymbol{A}^{\top}\boldsymbol{y} = \boldsymbol{0}, \quad \boldsymbol{y} \geq 0, \quad \langle \boldsymbol{b}, \boldsymbol{y} \rangle < 0.$$

This concludes the proof. □

One more important corollary will be given without proof.

Corollary 9.4. *Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. Then for $\delta > 0$ and every vector $\boldsymbol{x} \in \mathbb{R}^n$ with $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, $\langle \boldsymbol{c}, \boldsymbol{x} \rangle \leq \delta$ holds if and only if there exists $\boldsymbol{y} \in \mathbb{R}^m$ such that*

$$\boldsymbol{y} \geq \boldsymbol{0}, \quad \boldsymbol{A}^{\top}\boldsymbol{y} = \boldsymbol{c}, \quad \langle \boldsymbol{y}, \boldsymbol{b} \rangle \leq \delta.$$

9.2 Linear programming duality

After studying the feasible sets of linear programming, the polyhedra, we now return to linear programming itself, in the form

$$\text{maximize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}. \tag{9.1}$$

Geometrically this amounts to moving the hyperplane orthogonal to $\boldsymbol{c}$ to the highest level along $\boldsymbol{c}$, under the condition that it still intersects $P = \{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$.

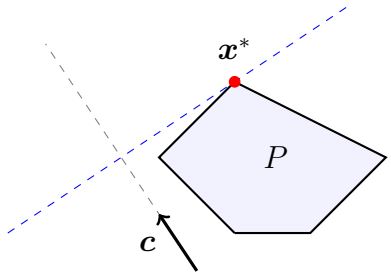


Figure 9.17: Geometry of linear programming

The famous duality theorem for linear programming states that if the maximum of (9.1) exists, then it coincides with the solution of a *dual* linear programming problem.

Theorem 9.5. *Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$. Then the optimal value of*

$$\textit{maximize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \quad \textit{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{P}$$

coincides with the optimal value of

$$\textit{minimize} \quad \langle \boldsymbol{b}, \boldsymbol{y} \rangle \quad \textit{subject to} \quad \boldsymbol{A}^{\top}\boldsymbol{y} = \boldsymbol{c},\ \boldsymbol{y} \geq \boldsymbol{0}, \tag{D}$$

provided both (P) *and* (D) *have a finite solution.*

The problem (P) is called the *primal* problem, and (D) the *dual* problem.

Example 9.6. Consider the simple problem

$$\text{maximize} \quad x_1 \quad \text{subject to} \quad x_1 + x_2 \leq 1, x_1 \geq 0, x_2 \geq 0.$$

The dual problem is

$$\text{minimize} \quad y_1 \quad \text{subject to} \quad y_1 - y_2 = 1, y_1 - y_3 = 0, y_1 \geq 0, y_2 \geq 0, y_3 \geq 0.$$

For the proof we need the following observation.

Lemma 9.7. *Let $P \subset \mathbb{R}^n$ be a polyhedron and $\boldsymbol{c} \in \mathbb{R}^n$ such that $\sup_{\boldsymbol{x}\in P}\langle \boldsymbol{c}, \boldsymbol{x}\rangle$ is finite. Then the supremum is attained, that is, it is a maximum.*

Proof of Theorem 9.5. Let $P = \{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \in \boldsymbol{b}\}$ and $D = \{\boldsymbol{y} \in \mathbb{R}^m : \boldsymbol{A}^\top \boldsymbol{y} = \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\}$. If $\boldsymbol{x} \in P$ and $\boldsymbol{y} \in Q$, then

$$\langle \boldsymbol{c}, \boldsymbol{x}\rangle = \langle \boldsymbol{A}^\top \boldsymbol{y}, \boldsymbol{x}\rangle = \langle \boldsymbol{y}, \boldsymbol{A}\boldsymbol{x}\rangle \leq \langle \boldsymbol{y}, \boldsymbol{b}\rangle,$$

so that in particular

$$\max_{\boldsymbol{x}\in P}\langle \boldsymbol{c}, \boldsymbol{x}\rangle \leq \min_{\boldsymbol{y}\in Q}\langle \boldsymbol{b}, \boldsymbol{y}\rangle,$$

which shows one inequality. To show the other inequality, set $\delta = \max_{\boldsymbol{x}\in P}\langle \boldsymbol{c}, \boldsymbol{x}\rangle$. By definition, if $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, then $\langle \boldsymbol{c}, \boldsymbol{x}\rangle \leq \delta$. By Corollary 9.4, there exists a vector $\boldsymbol{y} \in \mathbb{R}^m$ such that

$$\boldsymbol{y} \geq \boldsymbol{0}, \quad \boldsymbol{A}^\top \boldsymbol{y} = \boldsymbol{c}, \quad \langle \boldsymbol{b}, \boldsymbol{y}\rangle \leq \delta.$$

In particular,

$$\min_{\boldsymbol{y}\in Q}\langle \boldsymbol{b}, \boldsymbol{y}\rangle \leq \delta = \max_{\boldsymbol{x}\in P}\langle \boldsymbol{c}, \boldsymbol{x}\rangle.$$

This finishes the proof.

□

Lecture 10

So far we have seen aspects of the geometry of linear programming, discussed the feasible sets (polyhedra) and presented a useful duality theorem: the optimal value of

$$\text{maximize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{P}$$

coincides with the optimal value of

$$\text{minimize} \quad \langle \boldsymbol{b}, \boldsymbol{y} \rangle \quad \text{subject to} \quad \boldsymbol{A}^\top \boldsymbol{y} = \boldsymbol{c},\ \boldsymbol{y} \geq \boldsymbol{0}, \tag{D}$$

provided both (P) and (D) have a finite solution.

We now turn attention to algorithms for linear programming. We present the main idea behind the classical Simplex Algorithm and then discuss the more modern class of Interior Point Algorithms in more detail. The latter can be shown to be efficient in a very precise sense (polynomial time) and generalize to non-linear convex optimization.

10.1 A first algorithm for linear programming

For $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ let $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ be the polyhedron of feasible points for (P). Recall that a vertex of P is a zero-dimensional face of P, or equivalently, a point that cannot be written as convex combination of two distinct points of P.

If the optimization problem (P) has a solution, then it can be shown (Problem Set 4) that there exists a vertex $\boldsymbol{x}^*$ such that

$$\max_{\boldsymbol{x} \in P} \langle \boldsymbol{c}, \boldsymbol{x} \rangle = \langle \boldsymbol{c}, \boldsymbol{x}^* \rangle.$$

Intuitively this is clear by looking at a picture, as in Figure 10.18: Move a hyperplane orthogonal to the objective $\boldsymbol{c}$ along this direction to the highest level that still intersects a polyhedron P, and try to imagine that this intersection *does not* contain a vertex.

This crucial observation turns linear programming into a *finite* problem: we only have to test the objective function on the finite set of vertices to determine the maximizer. We have also seen (Problem Set 3) that the vertices can be identified as the solutions of the systems of equations

$$\boldsymbol{A}_I \boldsymbol{x} = \boldsymbol{b}_I,$$

for which $\boldsymbol{A}_I$ has full rank and $\boldsymbol{x} \in P$, where $I \subset \{1, \ldots, m\}$ is a subset of indices with $|I| = d$, and $\boldsymbol{A}_I, \boldsymbol{b}_I$ are the matrix and vector arising from $\boldsymbol{A}$ and $\boldsymbol{b}$ by taking the rows indexed by I. This gives a first primitive algorithm for solving the optimization problem (P).

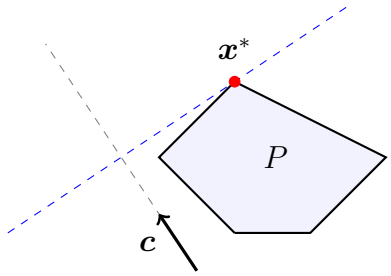
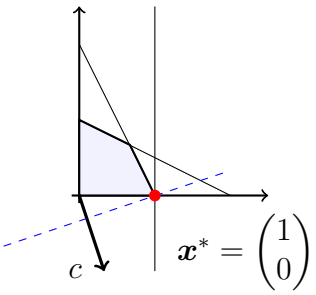


Figure 10.18: Geometry of linear programming

$$\begin{aligned} \text{maximize} \quad & x_1 - 3x_2 \\ \text{subject to} \quad & 0.5x_1 + x_2 \leq 1 \\ & x_1 + 0.5x_2 \leq 1 \\ & -x_1 \leq 0 \\ & -x_2 \leq 0 \\ & x_1 \leq 1 \end{aligned}$$



Example 10.1. Consider the linear programming problem

The matrix $\boldsymbol{A}$ and vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ are

$$\boldsymbol{A} = \begin{pmatrix} 0.5 & 1 \\ 1 & 0.5 \\ -1 & 0 \\ 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} 1 \\ -3 \end{pmatrix}.$$

We see that the minor $\boldsymbol{A}_{\{3,5\}} = \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}$, corresponding to the two parallel hyperplanes $x_1 = 0$ and $x_1 = 1$, is singular and does not define a vertex, while all other minors are invertible. For the minor $I = \{1, 2\}$, $\boldsymbol{A}_I \boldsymbol{x} = \boldsymbol{b}_I$ has the form

$$\begin{aligned} 0.5x_1 + x_2 &= 1 \\ x_1 + 0.5x_2 &= 1, \end{aligned}$$

which has the solution $\boldsymbol{x} = (2/3, 2/3)^\top$. Since $\boldsymbol{x}$ also satisfies all the other inequalities, it is a vertex. In a similar fashion we can find all the vertices as the points

$$\boldsymbol{x}_1 = \begin{pmatrix} 2/3 \\ 2/3 \end{pmatrix}, \ \boldsymbol{x}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ \boldsymbol{x}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \ \boldsymbol{x}_4 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

The values of the objective function at these vertices are

$$\langle \boldsymbol{c}, \boldsymbol{x}_1 \rangle = -\frac{4}{3}, \ \langle \boldsymbol{c}, \boldsymbol{x}_2 \rangle = 1, \ \langle \boldsymbol{c}, \boldsymbol{x}_3 \rangle = -3, \ \langle \boldsymbol{c}, \boldsymbol{x}_4 \rangle = 0.$$

It follows that the optimization problem has the optimal value 1 at $\boldsymbol{x}^* = (1, 0)^\top$.

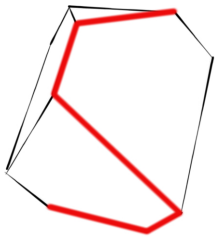
Unfortunately, this method requires solving up to $\binom{m}{n}$ systems of linear equations, which can be exponential in n in the worst case, making this method not practical.

Example 10.2. Consider the hypercube

$$\begin{aligned} -1 \leq x_1 \leq 1 \\ \cdots \\ -1 \leq x_n \leq 1. \end{aligned}$$

It has 2^n vertices that need to be checked. Even though there are so many vertices, one can travel along edge between any two vertices in at most n steps.

The simplex algorithm is a way to search through the vertices in a more clever way than just listing all of them. The idea is to start with a vertex and see if there is a vertex connected to it by an edge that has a bigger objective value. If not, we are done. If there is a neighbouring vertex, we move there and continue the process. We keep walking along the vertices of the feasible polytope until we find an optimal one. The simplex algorithm is one of the most successful



algorithms around and usually very fast, even though there examples that show that its worst case running time can be exponential.

Lecture 11

We continue the study of algorithms for linear programming. Before doing so, we introduce a **standard form** for a linear programming problem, namely

$$\text{minimize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}. \tag{P}$$

for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$. Any linear programming problem can be brought into this standard for. Start, for example, with the problem

$$\text{maximize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \tag{1.1}$$

in the form that we are used to. We first add a **slack variable** $\boldsymbol{s} \in \mathbb{R}^m$, so that we can reformulate the problem into an equalent one of the form

$$\text{maximize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}, \boldsymbol{s} \geq \boldsymbol{0}. \tag{1.2}$$

Define the big matrix $\boldsymbol{A}' \in \mathbb{R}^{m \times (n+n+m)}$ by

$$\boldsymbol{A}' := \begin{pmatrix} \boldsymbol{A} & -\boldsymbol{A} & \boldsymbol{I} \end{pmatrix},$$

consider the big vector $\boldsymbol{x}' = (\boldsymbol{x}^+, \boldsymbol{x}^-, \boldsymbol{s})^\top$, where $\boldsymbol{x}^+$ is the *positive part*, with entries x_i if $x_i \geq 0$ and 0 else, and $\boldsymbol{x}^-$ is the *negative part*, with entries $-x_i$ if $x_i < 0$ and 0 else (so that $\boldsymbol{x} = \boldsymbol{x}^+ - \boldsymbol{x}^-$), and set $\boldsymbol{c}' = (\boldsymbol{c}, -\boldsymbol{c}, \boldsymbol{0})^\top \in \mathbb{R}^{2n+m}$.

Then $\boldsymbol{x}$ is a solution of (1.2) if and only if $\boldsymbol{x}'$ is a solution of the problem

$$\text{minimize} \quad \langle \boldsymbol{c}', \boldsymbol{x}' \rangle \qquad \text{subject to} \quad \boldsymbol{A}'\boldsymbol{x}' = \boldsymbol{b},\ \boldsymbol{x}' \geq \boldsymbol{0},$$

which is in standard form (P). Note that since (P) has the form of a dual problem as derived in Lecture 9 and 10, its dual, in turn, has the form of a primal problem,

$$\text{maximize} \quad \langle \boldsymbol{b}, \boldsymbol{y} \rangle \qquad \text{subject to} \quad \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{s} \geq \boldsymbol{0}. \tag{D}$$

For everything that follows, we assume $m \leq n$, as otherwise (D) is unbounded and (P) empty.

1.1 An optimality condition

We now work with a primal dual system *in standard form*,

$$\text{minimize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}, \tag{P}$$

and

$$\text{maximize} \quad \langle \boldsymbol{b}, \boldsymbol{y} \rangle \quad \text{subject to} \quad \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{s} \geq \boldsymbol{0}. \tag{D}$$

A useful consequence of the duality theorem, which states that a primal optimal solution is also dual optimal, is a characterisation of solution tuples $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*)$. Note that for such an optimal pair,

$$\langle \boldsymbol{c}, \boldsymbol{x}^* \rangle = \langle \boldsymbol{y}^*, \boldsymbol{b} \rangle = \langle \boldsymbol{y}^*, \boldsymbol{A}\boldsymbol{x}^* \rangle = \langle \boldsymbol{A}^\top \boldsymbol{y}^*, \boldsymbol{x}^* \rangle,$$

where the first equality holds because of the duality theorem. By subtracting the last from the first expression above, we conclude

$$\langle \boldsymbol{x}^*, \boldsymbol{c} - \boldsymbol{A}^\top \boldsymbol{y}^* \rangle = \langle \boldsymbol{x}^*, \boldsymbol{s}^* \rangle = 0. \tag{PD}$$

Since $\boldsymbol{x}^* \geq \boldsymbol{0}$ and $\boldsymbol{s}^* \geq \boldsymbol{0}$, each summand in (PD) is zero. This means that the individual components of $\boldsymbol{s}^*$ and $\boldsymbol{x}^*$ satisfy

$$x_i^* \cdot s_i^* = 0,\ 1 \leq i \leq m.$$

Summarising, we have the following *optimality conditions* for linear programming: if vectors $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ are primal/dual optimal solutions to a linear programming problem, then

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} + \boldsymbol{s} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{A}^\top \boldsymbol{y} - \boldsymbol{c} &= \boldsymbol{0} \\ y_i s_i &= 0,\ 1 \leq i \leq m \\ \boldsymbol{y} &\geq \boldsymbol{0} \\ \boldsymbol{s} &\geq \boldsymbol{0}. \end{aligned} \tag{1.1}$$

We simplify this expression a bit further. Define the diagonal matrices

$$\boldsymbol{X} = \begin{pmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n \end{pmatrix}, \quad \boldsymbol{S} = \begin{pmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_n \end{pmatrix}$$

and the vector $\boldsymbol{e} = (1, 1, \ldots, 1)^\top$. Then the condition $x_i s_i = 0$ for $1 \leq i \leq n$ can be written concisely as $\boldsymbol{X}\boldsymbol{S}\boldsymbol{e} = \boldsymbol{0}$, and the whole system as

$$\begin{aligned} \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} &= \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} &= \boldsymbol{0} \\ \boldsymbol{x} &\geq \boldsymbol{0} \\ \boldsymbol{s} &\geq \boldsymbol{0}, \end{aligned} \tag{1.2}$$

Just as the optimality condition $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ serves as the basis for algorithms for unconstraint optimization, the optimality conditions for linear programming form the basis of the simplex method and of interior point methods. In the simplex method, the conditions are used to verify whether a candidate vertex is an optimal point, while primal/dual interior point methods view (15.1) as a multivariate function in $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ for which a root satisfying the inequality constraints is sought using Newton's method or similar algorithms.

1.2 Newton's method for solving equations

Recall that we have seen Newton's method in two forms: as a minimization algorithm, and as a method for finding roots of a non-linear equation. In the latter, in the one-dimensional setting we want to solve an equation

$$f(x) = 0$$

and proceed by starting with an initial guess x_0, and then computing successively

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

If f is the derivative of another function g, then this method is used to find a local minimizer of g. For a function $F = (f_1, \dots, f_n)^\top \colon \mathbb{R}^n \to \mathbb{R}^n$ we can use the same method, only that dividing by the derivative is replaced with multiplying with the inverse of the Jacobian matrix,

$$\boldsymbol{J}F(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}.$$

Newton's method then starts with an initial guess $\boldsymbol{x}_0$, and proceeds by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{J}F(\boldsymbol{x}_k)^{-1} F(\boldsymbol{x}_k).$$

In practise, one does not compute the inverse, but solves a system of equations to get the update,

$$\boldsymbol{J}F(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = F(\boldsymbol{x}_k), \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \Delta\boldsymbol{x}_k. \tag{1.1}$$

Note that we allow for adjusting the step length in Newton's method. This can be rather convenient.

We can apply the multivariate Newton's method to the equalities in the optimality condition (15.1), by considering the function $F \colon \mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m}$ defined by

$$F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \begin{pmatrix} \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} \end{pmatrix}$$

We are then looking for a root $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*)$ of this function that in addition satisfies the inequality constraints in (15.1). Before we get into the non-trivial issue enrusing non-negativity, we first have a look at what the *update step* (1.1) looks like. Computing the Jacobian for F, the updates are computed by solving

$$\begin{pmatrix} \boldsymbol{0} & \boldsymbol{A}^\top & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{X} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} \end{pmatrix}$$

Interior point method change the condition the condition $\boldsymbol{X}\boldsymbol{S}\boldsymbol{e} = \boldsymbol{0}$ to $\boldsymbol{X}\boldsymbol{S}\boldsymbol{e} = \tau\boldsymbol{e}$ for some $\tau > 0$, thus ensuring that we get non-negative solutions. Solving this problem for values $\tau \to 0$ then gives an approximation of the true solution. We discuss this in more detail in the next lecture.

Lecture 12

Recall the following primal dual standard forms of linear programming,

$$\text{minimize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}. \tag{P}$$

for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$, and

$$\text{maximize} \quad \langle \boldsymbol{b}, \boldsymbol{y} \rangle \qquad \text{subject to} \quad \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{s} \geq \boldsymbol{0}. \tag{D}$$

For what follows, we assume $m \leq n$, as otherwise (D) is unbounded and (P) empty.

Based on (P) and (D), we get the optimality conditions (see Lecture 11)

$$\begin{aligned} \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} &= \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} &= \boldsymbol{0} \\ \boldsymbol{x} &\geq \boldsymbol{0} \\ \boldsymbol{s} &\geq \boldsymbol{0}, \end{aligned} \tag{O}$$

where $\boldsymbol{X}$ is the diagonal matrix with the x_i in the diagonal, $\boldsymbol{S}$ the diagonal matrix with the s_i on the diagonal, and $\boldsymbol{e}$ the vector with all ones ($\boldsymbol{X}\boldsymbol{S}\boldsymbol{e}$ is just a compact way of writing the vector with entries $x_i s_i$). If we define the function $\mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m}$

$$F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \begin{pmatrix} \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} \end{pmatrix} \tag{12.1}$$

then what we are looking for is a *root* $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*)$ of this function, i.e., a point with $F(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*) = \boldsymbol{0}$, with the additional property that $\boldsymbol{x}^*$ and $\boldsymbol{s}^*$ be non-negative. For later reference, we record the form of the *Jacobian* matrix of F,

$$\boldsymbol{J}F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{A}^\top & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{X} \end{pmatrix}.$$

12.1 Towards an efficient interior-point method

The challenge is to find an iterative method of solving this rootfinding problem while preserving the non-negativity constraints. We discuss three approaches, in increasing order of sophistication. The third of these approaches forms the basis of *primal dual interior point methods*.

A first attempt

A first attempt at solving the problem $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \mathbf{0}$, with $\boldsymbol{x} \geq \mathbf{0}, \boldsymbol{s} \geq \mathbf{0}$, is to apply Newton's method with really small steps. For that, at each step we solve

$$\begin{pmatrix} \mathbf{0} & \boldsymbol{A}^\top & \boldsymbol{I} \\ \boldsymbol{A} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{S}^{(k)} & \mathbf{0} & \boldsymbol{X}^{(k)} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \boldsymbol{s} \end{pmatrix} = \begin{pmatrix} \boldsymbol{c} - \boldsymbol{s}^{(k)} - \boldsymbol{A}^\top \boldsymbol{y}^{(k)} \\ \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)} \\ -\boldsymbol{X}^{(k)} \boldsymbol{S}^{(k)} \boldsymbol{e} \end{pmatrix}.$$

and then compute

$$\begin{pmatrix} \boldsymbol{x}^{k+1} \\ \boldsymbol{y}^{k+1} \\ \boldsymbol{s}^{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}^{k} \\ \boldsymbol{y}^{k} \\ \boldsymbol{s}^{k} \end{pmatrix} + \alpha_k \begin{pmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \boldsymbol{s} \end{pmatrix},$$

choosing the step length α_k so that the non-negativity of $\boldsymbol{x}$ and $\boldsymbol{s}$ remains. Note that above we have a + sign in front of alpha. That is because in the system of equations above, we took the negative $-F(\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{s}_k)$.

Example 12.1. Consider the linear programming problem

$$\begin{aligned} \text{minimize} \quad & x_1 + 2x_2 - 2x_3 \\ & x_1 - 2x_3 \\ & x_2 - x_3 = -1 \\ & x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0. \end{aligned}$$

The data for this problem is given by

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \end{pmatrix},\ \boldsymbol{b} = \begin{pmatrix} 1 \\ -1 \end{pmatrix},\ \boldsymbol{c} = \begin{pmatrix} 1 \\ 2 \\ -2 \end{pmatrix}.$$

Based on this data, the function F given by

$$F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \begin{pmatrix} y_1 + s_1 - 1 \\ y_2 + s_2 - 2 \\ -2y_1 - y_2 + s_3 + 2 \\ x_1 - 2x_3 - 1 \\ x_2 - x_3 + 1 \\ x_1 s_1 \\ x_2 s_2 \\ x_3 s_3 \end{pmatrix}$$

and the Jacobian matrix

$$\boldsymbol{J}F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -2 & -1 & 0 & 0 & 1 \\ 1 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ s_1 & 0 & 0 & 0 & 0 & x_1 & 0 & 0 \\ 0 & s_2 & 0 & 0 & 0 & 0 & x_2 & 0 \\ 0 & 0 & s_3 & 0 & 0 & 0 & 0 & x_3 \end{pmatrix}.$$

With this data at hand, we can easily use a Python program to solve the problem for us, making sure that the steplength is small enough. Starting with $\boldsymbol{x}^{(0)} = \boldsymbol{s}^{(0)} = (1,1,1)^\top$, we get the sequence of step lengths

$$\alpha_0 = 0.5455,\ \alpha_1 = 0.5455,\ \alpha_2 = 1,$$

with the corresponding sequence of $\boldsymbol{x}$ vectors

$$\boldsymbol{x}^{(1)} = \begin{pmatrix} 1.1818 \\ 0 \\ 0.5455 \end{pmatrix},\ \boldsymbol{x}^{(2)} = \begin{pmatrix} 2.1736 \\ 0 \\ 0.7934 \end{pmatrix},\ \boldsymbol{x}^{(3)} = \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix}.$$

One verifies that $\boldsymbol{x}^{(3)}$ is in fact a vertext of the feasible set, and $\langle \boldsymbol{x}^{(3)}, \boldsymbol{c} \rangle = 1$ gives the optimal value.

A second attempt

In general, the method of choosing small step lengths can be slow. A variation would be to solve for

$$F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \tau \boldsymbol{e} \end{pmatrix}, \tag{12.1}$$

for a parameter $\tau > 0$. Given some fixed initial values $\boldsymbol{x} \geq 0$ and $\boldsymbol{s} \geq 0$, compute the *duality measure*

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i s_i$$

as the average of the products and using the *centering parameter* $\sigma \in (0,1)$, set $\tau = \sigma\mu$. When aiming to solve (12.1) using Newton's method, we are aiming towards a solution $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*)$ where instead of asking that $x_i^* s_i^* = 0$, we want that $x_i^* s_i^* = \sigma\mu$, a value that is strictly positive, but smaller than average of the initial value.

In an additional twist to this approach, after starting with an initial guess $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$, we perform only *one* Newton step, and then update the duality measure μ with the new values of $\boldsymbol{x}$ and $\boldsymbol{s}$. This way we arrive at the following algorithm:

- Start with $(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{s}^{(0)})$;
- For each $k \geq 0$, compute

  $$\mu^{(k)} = \frac{1}{n} \sum_{i=1}^{n} x_i s_i$$

  and σ_k. Solve

  $$\begin{pmatrix} \boldsymbol{0} & \boldsymbol{A}^\top & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S}^{(k)} & \boldsymbol{0} & \boldsymbol{X}^{(k)} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix} = \begin{pmatrix} \boldsymbol{c} - \boldsymbol{s}^{(k)} - \boldsymbol{A}^\top \boldsymbol{y}^{(k)} \\ \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)} \\ -\boldsymbol{X}^{(k)} \boldsymbol{S}^{(k)} \boldsymbol{e} + \sigma\mu^{(k)} \boldsymbol{e} \end{pmatrix}$$

and compute

$$\begin{pmatrix} \boldsymbol{x}^{k+1} \\ \boldsymbol{y}^{k+1} \\ \boldsymbol{s}^{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}^{k} \\ \boldsymbol{y}^{k} \\ \boldsymbol{s}^{k} \end{pmatrix} + \alpha_k \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix},$$

for a small enough $\alpha_k > 0$ to ensure non-negativity.

A third attempt

For the purpose of analysis, it is convenient to let an interior point method operate on vectors that satisfy the first two equalities in (O) exactly. Define the feasible and strictly feasible sets as

$$\mathcal{F} = \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) : \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0},\ \boldsymbol{s} \geq \boldsymbol{0}\}$$
$$\mathcal{F}^\circ = \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) : \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} > \boldsymbol{0},\ \boldsymbol{s} > \boldsymbol{0}\}$$

Restricting to points in $\mathcal{F}^\circ$, the computation of the Newton update in the second approach would change to

$$\begin{pmatrix} \boldsymbol{0} & \boldsymbol{A}^\top & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S}^{(k)} & \boldsymbol{0} & \boldsymbol{X}^{(k)} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ -\boldsymbol{X}^{(k)}\boldsymbol{S}^{(k)}\boldsymbol{e} + \sigma\mu^{(k)}\boldsymbol{e} \end{pmatrix}$$

In each iteration, a Newton step is taken in the direction of the *central path*. This is a curve in $\mathcal{F}^\circ$ defined as the set of solutions of

$$\begin{aligned} \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} &= \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} &= \tau\boldsymbol{e} \\ \boldsymbol{x} &> \boldsymbol{0} \\ \boldsymbol{s} &> \boldsymbol{0}, \end{aligned} \tag{12.2}$$

where $\tau > 0$. As $\tau \to 0$, any solution of this system will converge to an optimal primal-dual vector $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ for the original linear programming problem. As we will see, practical primal-dual interior point methods will try to ensure that we always move within a neighbourhood of the central path. While this third approach lends itself well to analysis, one problem is that a starting point $(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{s}^{(0)}) \in \mathcal{F}^\circ$ may be hard to find.

Lecture 13

Primal-dual interior point methods aim to solve the problem

$$\text{minimize} \quad \langle \boldsymbol{c}, \boldsymbol{x} \rangle \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0} \tag{P}$$

for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$, by applying Newton-type iterations to the optimality conditions of linear programming. More precisely, we have seen the following algorithm. Recall the feasible sets

$$\begin{aligned} \mathcal{F} &= \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) : \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0},\ \boldsymbol{s} \geq \boldsymbol{0}\} \\ \mathcal{F}^\circ &= \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) : \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} > \boldsymbol{0},\ \boldsymbol{s} > \boldsymbol{0}\} \end{aligned}$$

The a simple primal-dual interior point method can be described as follows.

- Start with $(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{s}^{(0)}) \in \mathcal{F}^\circ$;
- For each $k \geq 0$, compute the duality parameter

  $$\mu^{(k)} = \frac{1}{n} \sum_{i=1}^{n} x_i s_i$$

  and choose $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$. Solve

  $$\begin{pmatrix} \boldsymbol{0} & \boldsymbol{A}^\top & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S}^{(k)} & \boldsymbol{0} & \boldsymbol{X}^{(k)} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \boldsymbol{s} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ -\boldsymbol{X}^{(k)} \boldsymbol{S}^{(k)} \boldsymbol{e} + \sigma \mu^{(k)} \boldsymbol{e} \end{pmatrix}$$

  and compute

  $$\begin{pmatrix} \boldsymbol{x}^{k+1} \\ \boldsymbol{y}^{k+1} \\ \boldsymbol{s}^{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}^{k} \\ \boldsymbol{y}^{k} \\ \boldsymbol{s}^{k} \end{pmatrix} + \alpha_k \begin{pmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{y} \\ \Delta \boldsymbol{s} \end{pmatrix},$$

  for a small enough $\alpha_k > 0$ to ensure non-negativity.

In each iteration, a Newton step is taken in the direction of the **path**. This is a curve in $\mathcal{F}^\circ$ defined as the set of solutions of

$$\begin{aligned} \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} &= \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} &= \tau \boldsymbol{e} \\ \boldsymbol{x} &> \boldsymbol{0} \\ \boldsymbol{s} &> \boldsymbol{0}, \end{aligned} \tag{13.1}$$

where $\tau > 0$.

13.1 Path-following methods

A path-following method tries to ensure that each iterate is *close* to the central path. What it means to be close to the central path depends on the neighbourhood we choose. Here, we will look at the (one-sided) ∞-norm neighbourhood

$$\mathcal{N}_{-\infty}(\gamma) = \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{F}^\circ : x_i s_i \geq \gamma\mu,\ 1 \leq i \leq n\}$$

for some $\gamma \in (0, 1]$ (say, $\gamma = 10^{-3}$). In words, each $x_i s_i$ has to be at least some small multiple of their average value. To see what this has to do with the ∞-norm neighbourhood, consider the set of $\boldsymbol{x}$ such that

$$\|\boldsymbol{X}\boldsymbol{S}\boldsymbol{e} - \mu\boldsymbol{e}\|_\infty \leq (1-\gamma)\mu \iff \forall 1 \leq i \leq n,\ \gamma\mu \leq x_i s_i \leq 2 - \gamma,$$

and we are only interested in the lower inequality.

The so-called *long-step path-following* interior point method can then be described as follows.

- Start with $(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{s}^{(0)}) \in \mathcal{N}_{-\infty}(\gamma)$;
- For each $k \geq 0$, compute the duality parameter

  $$\mu^{(k)} = \frac{1}{n}\sum_{i=1}^{n} x_i s_i$$

  and choose $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$. Solve

  $$\begin{pmatrix} \boldsymbol{0} & \boldsymbol{A}^\top & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S}^{(k)} & \boldsymbol{0} & \boldsymbol{X}^{(k)} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ -\boldsymbol{X}^{(k)}\boldsymbol{S}^{(k)}\boldsymbol{e} + \sigma\mu^{(k)}\boldsymbol{e} \end{pmatrix}$$

  and compute

  $$\begin{pmatrix} \boldsymbol{x}^{k+1} \\ \boldsymbol{y}^{k+1} \\ \boldsymbol{s}^{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}^{k} \\ \boldsymbol{y}^{k} \\ \boldsymbol{s}^{k} \end{pmatrix} + \alpha_k \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix},$$

  for a small enough $\alpha_k \in [0, 1]$ is the largest value such that $(\boldsymbol{x}^{(k+1)}, \boldsymbol{y}^{(k+1)}, \boldsymbol{s}^{(k+1)}) \in \mathcal{N}_{-\infty}(\gamma)$.

Remark 13.1. As noted at the end of Lecture 12, to find an initial point in $\mathcal{F}^\circ$ might not be trivial. In practice one can therefore also use the algorithm described above using *infeasible* points, though in this case we have to make sure that the residual norms $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|$ and $\|\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^\top \boldsymbol{y}\|$ remain bounded.

Visualising the algorithm

The feasible set $\mathcal{F}^\circ$ usually lives in a space that can't be easily visualised, but if the dual version is two-dimensional,

$$\{\boldsymbol{y} \in \mathbb{R}^2 : \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{s} \geq \boldsymbol{0}\},$$

then we have a chance to see how the trajectories of the iterates in $\boldsymbol{y}$ look like. Consider, for example, the linear programming problem whose dual is given by

$$\begin{aligned} &\text{maximize} && y_1 + y_2 \\ &\text{subject to} && 0.2py_1 + y_2 + s_p = 1 + 0.01p^2,\ 0 \leq p \leq 10. \end{aligned}$$

The points $\boldsymbol{y} = (0,0)^\top$, $\boldsymbol{x} = (1,\ldots,1)^\top/11$ and $\boldsymbol{s} = \boldsymbol{c} - \boldsymbol{A}^\top \boldsymbol{y}$ are strictly feasible starting points. The trajectory in the $\boldsymbol{y}$-plane of the algorithm (red path) and the constraint equations (blue lines) are given in the following diagram, where the parameter $\sigma = 0.5$ was used. It is

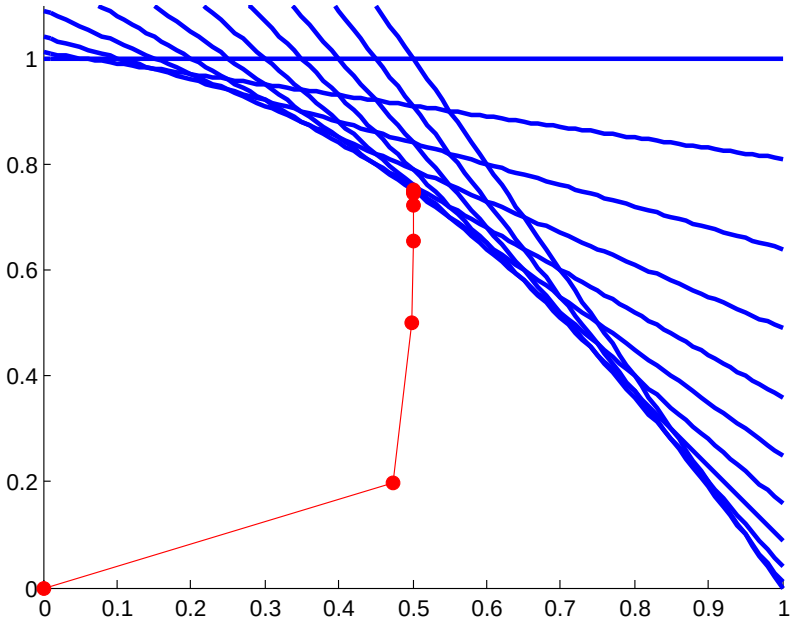


Figure 13.19: Trajectory of long-step path-following in the $y_1 - y_2$ plane.

instructive to play around with the parameter σ and to try to determine the form of the central path in this example.

Another way to visualise the trajectory is to plot the pairs $x_i s_i$ and $x_j y_j$ against each other. Figure 13.20 shows the trajectory of the above example in the $x_2 s_2 - x_5 s_5$ plane. Note that the central path, plotted in blue, is trivial in these coordinates, as it is defined by the property of the $x_i s_i = \tau$ being equal.

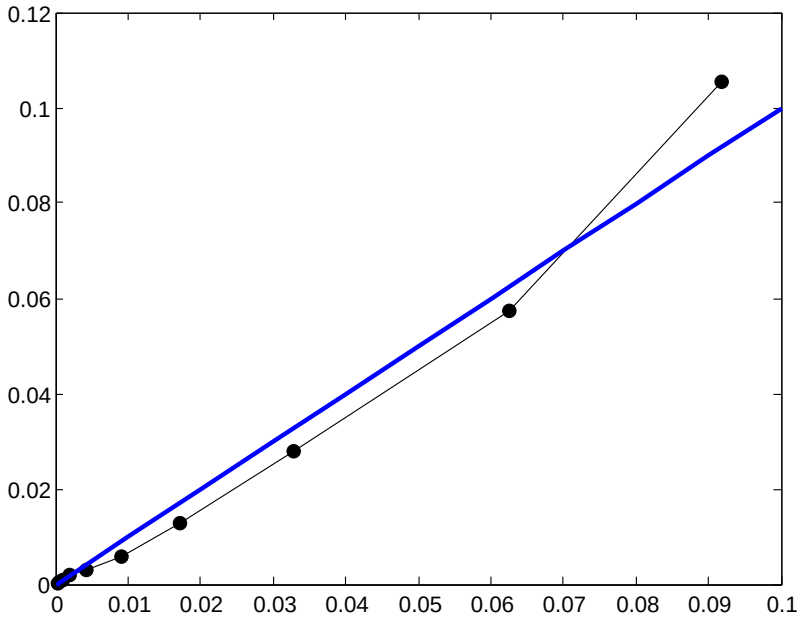


Figure 13.20: Trajectory and central path in $x_2 s_2 - x_5 s_5$ coordinates.

13.2 Analysis of Path-following

In the analysis of the long-step path-following algorithm, it is enough to establish that the duality measure $\mu^{(k)}$ converges to 0 as $k \to \infty$. The reason is that $\mu = 0$ forces all the products $x_i s_i = 0$, and since by design the other constraints are satisfied, this means that the sequence of points converges to a solution. The first theorem tells us that the μ_k decrease as k increases. An elementary proof is given in Theorem 14.3 in Nocedal and Wright. It depends crucially on the assumption that the iterates remain inside the neighbourhood $\mathcal{N}_{-\infty}(\gamma)$ of the central path.

Theorem 13.2. *Given parameters γ, $\sigma_{\min}$ and $\sigma_{\max}$, there is a constant $\delta > 0$, independent of n, such that*

$$\mu_{k+1} \leq \left(1 - \frac{\delta}{n}\right) \mu_k. \tag{13.1}$$

The next theorem gives a bound on the number of iterations needed to reduce the duality measure beyond any given ε.

Theorem 13.3. *Let $\varepsilon > 0$ and $\gamma \in (0, 1)$. Let $(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{s}^{(0)}) \in \mathcal{N}_{-\infty}(\gamma)$ be a starting point such that the duality measure satisfies $\mu^{(0)} \leq \varepsilon^{-\kappa}$ for some constant κ. Then there is an index $K = O(n \log(1/\varepsilon))$ such that for all $k > K$,*

$$\mu_k \leq \varepsilon.$$

In particular, the long-step path-following algorithm converges.

Proof. Repeatedly applying (19.4), we get

$$\mu_k \leq \left(1 - \frac{\delta}{n}\right)^k \mu_0.$$

Taking logarithms on both sides,

$$\begin{aligned}\log \mu_k \leq k \log\left(1 - \frac{\delta}{n}\right) + \log \mu_0 &\leq k \log\left(1 - \frac{\delta}{n}\right) + \kappa \log\left(\frac{1}{\varepsilon}\right) \\ &\leq k\frac{-\delta}{n} + \kappa \log\left(\frac{1}{\varepsilon}\right).\end{aligned}$$

We have $\mu_k < \varepsilon$ if

$$-k\frac{\delta}{n} + \kappa\left(\frac{1}{\varepsilon}\right) \leq \log \varepsilon,$$

of equivalently, if

$$k \geq (1+\kappa)\frac{n}{\delta}\log\left(\frac{1}{\varepsilon}\right) = K.$$

This was to be shown.

□

Part IV

Non-linear Convex Optimization

Lecture 14

So far we have only dealt with constrained optimization problems where the objective and the constraints are linear. We now turn attention to general problems of the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{f} = (f_1, \ldots, f_m)^\top$, $\boldsymbol{h} = (h_1, \ldots, h_p)$, and the inequalities are componentwise. The problem (1) is *convex*, if f and the g_i are convex, and the h_j are linear. We also denote by $\mathrm{dom}(f)$ the *domain* of f, which is the set of points $\boldsymbol{x}$ where f takes a finite value. The feasible set

$$\mathcal{F} = \{\boldsymbol{x} : f_i(\boldsymbol{x}) \leq 0,\ h_j(\boldsymbol{x}) = 0,\ 1 \leq i \leq m,\ 1 \leq j \leq \ell\}$$

for a convex constrained problem is a convex set.

14.1 Quadratic Programming and Portfolio Optimization

The simplest case of non-linear, constrained convex optimization is **quadratic programming**. Quadratic programming problems are of the form

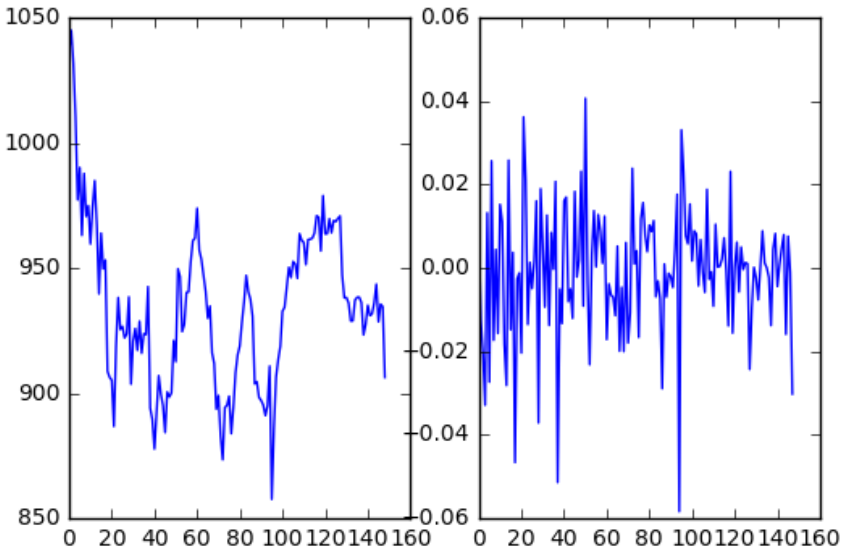
$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \end{aligned}$$

with $\boldsymbol{Q}$ symmetric and positive semidefinite. Note that this problem combines two problems we studied in some detail before: minimizing a quadratic function, and linear constraints. Just as we did for unconstrained minimization and for linear programming, we will derive optimality conditions for such problems. Before studying the theory, we first present an important application: portfolio optimization.

Mean-variance portfolio theory

If we invest an amount x^0 into a product at period 0, and at period 1 (for example, one day) the value is x^1, then the **relative return** is defined as

$$r = \frac{x^1 - x^0}{x^0}.$$



The following figure shows the price movement and the returns of a stock from January 2016 until now.

As we can't predict the future, we usually work with the **expected return** $\boldsymbol{E}[r] = \mu$, which is a statistical estimate of the future return. One naive method of estimating the future return is by taking the average of past returns, but other more sophisticated methods are possible.

In **portfolio optimization**, we have a proportion x_i of our available funds that we want to invest in a stock i (in particular, as x_i measures a proportion, we have $\sum_{i=1}^{n} x_i = 1$). We may or may not allow $x_i < 0$, which would correspond to short-selling or borrowing. Given this allocation, the overall return is $\boldsymbol{x}^\top \boldsymbol{r}$, where $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ and $\boldsymbol{r} = (r_1, \ldots, r_n)^\top$ is the vector of (relative) returns. If $\boldsymbol{\mu} = \boldsymbol{E}[\boldsymbol{r}]$ denotes the vector of expected returns, then the total expected return is

$$\mu = \boldsymbol{x}^\top \boldsymbol{\mu} = \sum_{i=1}^{n} x_i \mu_i.$$

The **risk** is an investment is measured in terms of the **variance** of the returns $r = \boldsymbol{x}^\top \boldsymbol{r}$. Let $\boldsymbol{\Sigma}$ denote the **covariance matrix**, where the (i, j)-th entry is $\mathrm{Cov}(r_i, r_j) = \boldsymbol{E}[(r_i - \mu_i)(r_j - \mu_j)]$. The (i, j)-th entry measures how much products i and j are correlated. The **variance** of the returns r is then the quadratic function

$$\boldsymbol{x}^\top \Sigma \boldsymbol{x}.$$

A portfolio optimization problem either seeks to maximize the return while bounding the

risk,

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{x}^\top \boldsymbol{\mu} \\ \text{subject to} \quad & \boldsymbol{x}^\top \boldsymbol{\Sigma} \boldsymbol{x} \leq \sigma, \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \end{aligned}$$

or minimize the risk given a certain target return μ,

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{x}^\top \boldsymbol{\Sigma} \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{x}^\top \boldsymbol{\mu} = \mu \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0. \end{aligned}$$

Both of these problems are convex optimization problems. The constraints $x_i \geq 0$ mean that we are not allowed to short-sell; when dealing with futures or options, or if we are a large institutional investor, we may drop these constraints. As we will see later, dropping the inequality constraints from the second formulation allows the problem to be solved in closed form.

A third form of the portfolio optimization problem is to combine the expected return and the risk into a single objective function, as follows:

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{\mu}^\top \boldsymbol{x} - \gamma \boldsymbol{x}^\top \boldsymbol{\Sigma} \boldsymbol{x} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0. \end{aligned}$$

Note that this is a convex quadratic problem: we can transform it into a minimization problem with the matrix $\boldsymbol{\Sigma}$ by changing the sign. The parameter γ is called a **risk aversion parameter**; it adjusts the level of risk we are willing to take. If $\gamma = 0$, then the quadratic term does not feature in the objective and we just aim to maximize the expected return. If γ is big, then the risk term weighs heavily on the objective function, and the optimal value will like be one where $\boldsymbol{x}^\top \boldsymbol{\Sigma} \boldsymbol{x}$ is small. By varying the value of γ, we get different expected return / risk (variance) trade-offs. The following code computes this trade-off curve for a portfolio of 10 stock from the FT100 index. The mean and covariance where computing by averaging over the past 60 trading days (3 months). The x-axis is the **standard deviation**, which is given as the square root of the variance (risk).

In [22]:
```
from datetime import datetime, date
import pandas as pd
from pandas_datareader import data, wb
import numpy as np
import matplotlib.pyplot as plt
from cvxpy import *
```

We first use the functionality of the Pandas module to load the price data from the Yahoo Finance web site.

In [23]:
```
START = datetime(2016,1,1)
END = date.today()
TICKER = ['ADN', 'AZN', 'EZJ', 'GSK', 'ITV', 'LSE', 'TSCO', 'PSON', 'PRU', 'DGE']
mydata = data.DataReader('TSCO', "yahoo", START, END)
dates = mydata.index
df = pd.DataFrame(index=dates, columns=TICKER)
for x in TICKER:
    mydata = data.DataReader(x, "yahoo", START, END)
    df.loc[:,x] = mydata['Adj Close']
df = df.dropna()
df.head(2)
```

The following shows a small sample of the data loaded.

	ADN	AZN	EZJ	GSK	ITV	LSE	TSCO
Date							
2016-01-04	2.41197	31.926706	84.900002	37.813187	0.4	0.002	82.800593
2016-01-05	2.41197	32.404650	86.620003	38.028194	0.4	0.001	82.741275

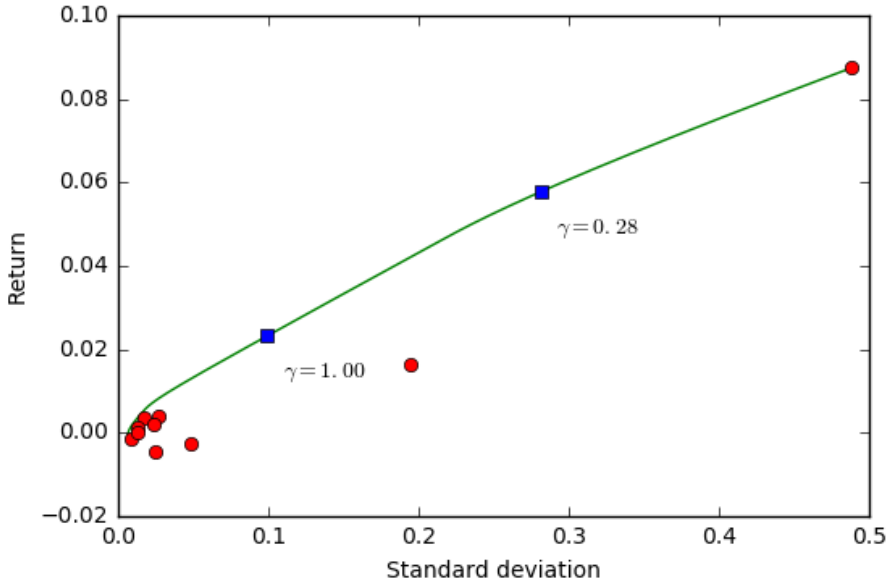
We next compute the returns and the mean and covariance.

In [24]:
```
price_matrix = df.values
returns = (price_matrix[1:]-price_matrix[0:-1])/price_matrix[0:-1]
mu = np.mean(returns[-60:,:], axis=0)
Sigma = np.cov(returns[-60:,:].T)
n = 10
```

In the next step, we start the CVXPY engine and compute the mean-variance trade-off curve. We include the constraint $x \geq 0$ to disallow for short-selling.

In [25]:
```
x = Variable(n)
gamma = Parameter(sign='positive')
ret = mu.T*x
risk = quad_form(x, Sigma)
prob = Problem(Maximize(ret - gamma*risk),
               [sum_entries(x) == 1,
                x >= 0])
```

In [26]:
```
SAMPLES = 1000
risk_data = np.zeros(SAMPLES)
ret_data = np.zeros(SAMPLES)
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
for i in range(SAMPLES):
    gamma.value = gamma_vals[i]
    prob.solve()
    risk_data[i] = sqrt(risk).value
    ret_data[i] = ret.value
```



In [27]:
```
markers_on = [290, 400]
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(risk_data, ret_data, 'g-')
for marker in markers_on:
    plt.plot(risk_data[marker], ret_data[marker], 'bs')
    ax.annotate(r"$\gamma = %.2f$"
    % gamma_vals[marker], xy=(risk_data[marker]+0.01, ret_data[marker]-0.01))
for i in range(n):
    plt.plot(sqrt(Sigma[i,i]).value, mu[i], 'ro')
plt.xlabel('Standard deviation')
plt.ylabel('Return')
plt.show()
```

The red dots represent the standard deviation and expected return of the individual stocks. The two blue dots indicate the standard deviation and expected return of the portfolio for two values of γ. Which value of γ we go for depends on how much risk we are willing to take: if we are risk-averse, we may prefer the $\gamma = 1$ value to the $\gamma = 0.028$ value, at the expense of smaller expected returns.

In practical applications there are a lot of other factors to be considered, such as whether the estimation procedure for the mean and covariance makes sense. In addition, it is often common to add terms that account for **transaction costs** into the objective funtion.

Lecture 15

In this lecture we study optimality conditions for convex problems of the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{f} = (f_1, \ldots, f_m)^\top$, $\boldsymbol{h} = (h_1, \ldots, h_p)^\top$, and the inequalities are componentwise. We assume that f and the f_i are convex, and the h_j are linear. It is also customary to write the conditions $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$ as $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, with $h_j(\boldsymbol{x}) = \boldsymbol{a}_j^\top \boldsymbol{x} - b_j$, $\boldsymbol{a}_j$ being the j-th row of $\boldsymbol{A}$.

15.1 A first-order optimality condition

So far we have seen two examples of first order optimality conditions: for unconstraint optimization ($\nabla f(\boldsymbol{x}) = 0$) and for linear programming. We now generalize these to the setting of constrained convex optimization.

Theorem 15.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex, differentiable function, and*

$$\mathcal{F} = \{\boldsymbol{x} : f_i(\boldsymbol{x}) \leq 0,\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$$

a feasible set, with f_i convex. Then $\boldsymbol{x}^*$ is an optimal point of the optimization problem

$$\textit{minimize} \quad f(\boldsymbol{x}) \quad \textit{subject to} \quad \boldsymbol{x} \in \mathcal{F}$$

if and only if for all $\boldsymbol{y} \in \mathcal{F}$,

$$\langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle \geq 0. \tag{15.1}$$

Proof. Suppose $\boldsymbol{x}^*$ is such that (1) holds. Then, since f is a convex function, for all $\boldsymbol{y} \in \mathcal{F}$ we have, by Theorem 2.10.1,

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}^*) + \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle \geq f(\boldsymbol{x}^*),$$

which shows that $\boldsymbol{x}^*$ is a minimizer in $\mathcal{F}$. To show the opposite direction, assume that $\boldsymbol{x}^*$ is a minimizer but that (1) does not hold. This means that there exists a $\boldsymbol{y} \in \mathcal{F}$ such that

$\langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle < 0$. Since both $\boldsymbol{x}^*$ and $\boldsymbol{y}$ are in $\mathcal{F}$ and $\mathcal{F}$ is convex, any point $\boldsymbol{z}(\lambda) = (1-\lambda)\boldsymbol{x}^* + \lambda \boldsymbol{y}$ with $\lambda \in [0,1]$ is also in $\mathcal{F}$. At $\lambda = 0$ we have

$$\frac{df}{d\lambda} f(\boldsymbol{z}(\lambda))|_{\lambda=0} = \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle < 0.$$

Since the derivative at $\lambda = 0$ is negative, the function $f(\boldsymbol{z}(\lambda))$ is decreasing at $\lambda = 0$, and therefore, for small $\lambda > 0$, $f(\boldsymbol{z}(\lambda)) < f(\boldsymbol{z}(0)) = f(\boldsymbol{x}^*)$, in contradiction to the assumption that $\boldsymbol{x}^*$ is a minimizer. □

Example 15.2. In the absence of constraints, $\mathcal{F} = \mathbb{R}^n$, and the statement says that

$$\forall \boldsymbol{y} \in \mathbb{R}^n \colon \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle \geq 0.$$

Given $\boldsymbol{y}$ such that $\langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle \geq 0$, then replacing $\boldsymbol{y}$ by $2\boldsymbol{x} - \boldsymbol{y}$ we also have the converse inequality, and therefore the optimality condition is equivalent to saying that $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$. We therefore recover the well-known first order optimality condition from Lecture 2.

Geometrically, the first order optimality condition means that the set

$$\{\boldsymbol{x} : \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{x} \rangle = \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{x}^* \rangle\}$$

defines a supporting hyperplane to the set $\mathcal{F}$.

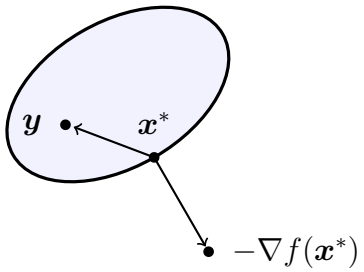


Figure 15.21: Optimality condition

15.2 Lagrangian duality

Recall the method of Lagrange multipliers. Given two functions $f(x,y)$ and $h(x,y)$, if the problem

$$\text{minimize} \quad f(x,y) \qquad \text{subject to} \quad h(x,y) = 0$$

has a solution (x^*, y^*), then there exists a parameter λ, the *Lagrange multiplier*, such that

$$\nabla f(x^*, y^*) = \lambda \nabla h(x^*, y^*). \tag{15.1}$$

In other words, if we define the *Lagrangian* as

$$\mathcal{L}(x, y, \lambda) = f(x,y) - \lambda h(x,y),$$

then (15.1) says that $\nabla \mathcal{L}(x^*, y^*, \lambda) = 0$ for some λ. The intuition is as follows. The set

$$M = \{(x, y) \in \mathbb{R}^2 : h(x, y) = 0\}$$

is a curve in $\mathbb{R}^2$, and the gradient $\nabla h(x, y)$ is perpendicular to M at every point $(x, y) \in M$. For someone living inside M, a vector that is perpendicular to M is not visible, it is zero. Therefore the gradient $\nabla f(x, y)$ is zero as viewed from within M if it is perpendicular to M, or equivalently, a multiple of $\nabla h(x, y)$.

Alternatively, we can view the graph of $f(x, y)$ in three dimensions. A maximum or minimum of $f(x, y)$ along the curve defined by $h(x, y) = 0$ will be a point at which the direction of steepest ascent $\nabla f(x, y)$ is perpendicular to the curve $h(x, y) = 0$.

Example 15.3. Consider the function $f(x, y) = x^2 y$ with the constraint $h(x, y) = x^2 + y^2 - 3$ (a circle of radius $\sqrt{3}$). The Lagrangian is the function

$$\mathcal{L}(x, y, \lambda) = x^2 y - \lambda(x^2 + y^2 - 3).$$

Computing the partial derivatives gives the three equations

$$\begin{aligned} \frac{\partial}{\partial x}\mathcal{L} &= 2xy - 2\lambda x = 0 \\ \frac{\partial}{\partial y}\mathcal{L} &= x^2 - 2\lambda y = 0 \\ \frac{\partial}{\partial \lambda}\mathcal{L} &= x^2 + y^2 - 3 = 0. \end{aligned}$$

From the second equation we get $\lambda = \frac{x^2}{2y}$, and the first and third equations become

$$\begin{aligned} 2xy - \frac{x^3}{y} &= 0 \\ x^2 + y^2 - 3 &= 0. \end{aligned}$$

Solving this system, we get six critical point $(\pm\sqrt{2}, \pm 1)$, $(0, \pm\sqrt{2})$. To find out which one of these is the minimizers, we just evaluate the function f on each of these.

We now turn to convex problems of the more general form form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{15.2}$$

Denote by $\mathcal{D}$ the *domain* of all the functions f, f_i, h_j, i.e.,

$$\mathcal{D} = \operatorname{dom}(f) \cap \operatorname{dom}(f_1) \cap \cdots \cap \operatorname{dom}(f_m) \cap \operatorname{dom}(h_1) \cap \cdots \cap \operatorname{dom}(h_p).$$

Assume that $\mathcal{D}$ is not empty and let p^* be the optimal value of (15.2).

The *Lagrangian* of the system is defined as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\mu}^\top h(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}) + \sum_{i=1}^{p} \mu_i h_i(\boldsymbol{x}).$$

The vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are called the *dual variables* or *Lagrange multipliers* of the system. The domain of $\mathcal{L}$ is $\mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$.

Definition 15.4. The *Lagrange dual* of the problem (15.2) is the function

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in \mathcal{D}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

If the Lagrangian $\mathcal{L}$ is unbounded from below, then the value is $-\infty$.

The Lagrangian $\mathcal{L}$ is linear in the λ_i and μ_j variables. The infimum of a family of linear functions is concave, so that the Lagrange dual is a concave function. Therefore the negative $-g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is a convex function.

Lemma 15.5. *For any $\boldsymbol{\mu} \in \mathbb{R}^p$ and $\boldsymbol{\lambda} \geq \mathbf{0}$ we have*

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq p^*.$$

Proof. Let $\boldsymbol{x}^*$ be a feasible point for (15.2), that is,

$$f_i(\boldsymbol{x}^*) \leq 0, \quad h_j(\boldsymbol{x}^*) = 0, \ 1 \leq i \leq m, \ 1 \leq j \leq p.$$

Then for $\boldsymbol{\lambda} \geq \mathbf{0}$ and any $\boldsymbol{\mu}$, since each $h_j(\boldsymbol{x}^*) = 0$ and $\lambda_j f_j(\boldsymbol{x}^*) \leq 0$,

$$\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j h_j(\boldsymbol{x}^*) \leq f(\boldsymbol{x}^*).$$

In particular,

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq f(\boldsymbol{x}^*).$$

Since this holds for *all* feasible $\boldsymbol{x}^*$, it holds in particular for the $\boldsymbol{x}^*$ that minimizes (15.2), for which $f(\boldsymbol{x}^*) = p^*$. □

A point $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ with $\boldsymbol{\lambda} \geq \mathbf{0}$ and $(\boldsymbol{\lambda}, \boldsymbol{\mu}) \in \operatorname{dom}(g)$ is called a *feasible point* of the dual problem.

The *Lagrange dual* of the optimization problem (15.2) is the problem

$$\text{maximize} \quad g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \qquad \text{subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0}. \tag{15.3}$$

We have seen that if q^* is the optimal value of (15.3), then $q^* \leq p^*$, and the example above implies that in the special case of linear programming we actually have $q^* = p^*$. We will see that under certain conditions, we have $q^* = p^*$ for more general problems, but this is not always the case.

Lecture 16

For convex problems of the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned} \tag{1}$$

we introduced the Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ and defined the Lagrange dual as

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in \mathcal{D}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

We saw that $g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is a lower bound on the optimal value of (1). Note that here we wrote the conditions $h_j(\boldsymbol{x}) = 0$ as system of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, since for the problem to be convex, we require that the h_j be linear functions.

Example 16.1. Consider a linear programming problem of the form

$$\begin{aligned} \text{minimize} \quad & \langle \boldsymbol{c}, \boldsymbol{x} \rangle \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned}$$

The inequality constraints are $-x_i \leq 0$, while the equality constraints are $\boldsymbol{a}_i^\top \boldsymbol{x} - b_i$. The Lagrangian has the form

$$\begin{aligned} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \langle \boldsymbol{c}, \boldsymbol{x} \rangle - \sum_{i=1}^{n} \lambda_i x_i + \sum_{j=1}^{m} \mu_j (\boldsymbol{a}_j^\top \boldsymbol{x} - b_j) \\ &= (\boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\top \boldsymbol{\mu})^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{\mu}. \end{aligned}$$

The infimum over $\boldsymbol{x}$ of this function is $-\infty$ unless $\boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\top \boldsymbol{\mu} = \boldsymbol{0}$. The Lagrange dual is therefore

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \begin{cases} -\boldsymbol{\mu}^\top \boldsymbol{b} & \text{if } \boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\top \boldsymbol{\mu} = \boldsymbol{0} \\ -\infty & \text{else.} \end{cases}$$

From Lemma 15.5 we conclude that

$$\max\{-\boldsymbol{b}^\top \boldsymbol{\mu} : \boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\top \boldsymbol{\mu} = \boldsymbol{0},\ \boldsymbol{\lambda} \geq \boldsymbol{0}\} \leq \min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0},\ \boldsymbol{x} \geq \boldsymbol{0}\}.$$

Note that if we write $\boldsymbol{y} = -\boldsymbol{\mu}$ and $\boldsymbol{s} = \boldsymbol{\lambda}$, then we get the dual version of the linear programming problem we started out with, and in this case we know that

$$\max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu}} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = p^*.$$

16.1 Constraint qualification

In the example of linear programming, we have seen that the optimal value of the dual problem is equal to the optimal value of the primal problem. In general, we have

$$d^* = \sup_{\boldsymbol{\lambda} > 0, \boldsymbol{\mu}} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \inf_{\boldsymbol{x} \in \mathcal{D}} \{f(\boldsymbol{x}) : f_i(\boldsymbol{x}) \leq 0,\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\} = p^*.$$

Once certain conditions, called *constraint qualifications*, hold, we can ensure that *strong duality* holds, which means $d^* = p^*$. One particular such constraint qualification is Slater's Theorem.

Theorem 16.2. *(Slater conditions) Assume that the interior of the domain $\mathcal{D}$ of* (1) *is non-empty, that the problem* (1) *is convex, and that there exists a point $\boldsymbol{x} \in \mathcal{D}$ such that*

$$f_i(\boldsymbol{x}) < 0,\ 1 \leq i \leq m, \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ 1 \leq j \leq p.$$

Then $d^* = p^*$, the primal optimal value coincides with the dual optimal value.

Proof. (Optional) Assume $\boldsymbol{A}$ has rank p (the number of rows). Assume moreover that p^* is finite, since if $p^* = -\infty$, then by weak duality we already have $d^* = p^*$.

Define the convex set

$$\mathcal{A} = \{(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R} : \forall \boldsymbol{x} \in \mathcal{D}, f_i(\boldsymbol{x}) \leq u_i, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{v}, f(\boldsymbol{x}) \leq t\}.$$

Then the optimal value of (1) is

$$p^* = \inf\{t : (\mathbf{0}, \mathbf{0}, t) \in \mathcal{A}\}.$$

Define the convex set $\mathcal{B}$ as

$$\mathcal{B} = \{(0, 0, s) \in \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R} : s < p^*\}.$$

The sets $\mathcal{A}$ and $\mathcal{B}$ are disjoint. To see this, assume to the contrary that there is a point $\boldsymbol{w} \in \mathcal{A} \cap \mathcal{B}$. Then since $\boldsymbol{w} \in \mathcal{B}$, $\boldsymbol{w} = (0, 0, s)$ with $s < p^*$, but also since $\boldsymbol{w} \in \mathcal{A}$, ther exists an $\boldsymbol{x} \in \mathcal{D}$ with $f_i(\boldsymbol{x}) \leq 0$, $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \mathbf{0}$, and $f(\boldsymbol{x}) \leq s < p^*$, in contradiction to the optimality of p^* as a value.

By the separating hyperplane theorem, there exist a hyperplane separating $\mathcal{A}$ and $\mathcal{B}$ (but not necessarilty strictly!), defined by a vector $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\mu}}, \nu) \neq \mathbf{0}$ and $\alpha \neq 0$ with the property that

$$(u, v, t) \in \mathcal{A} \Longrightarrow \boldsymbol{\lambda}^\top \boldsymbol{u} + \boldsymbol{\mu}^\top \boldsymbol{v} + \nu t \geq \alpha \tag{16.1}$$

and

$$(u, v, t) \in \mathcal{B} \Longrightarrow \boldsymbol{\lambda}^\top \boldsymbol{u} + \boldsymbol{\mu}^\top \boldsymbol{v} + \nu t \leq \alpha. \tag{16.2}$$

If $\tilde{\boldsymbol{\lambda}} < \mathbf{0}$ or $\nu < 0$, we could make the right-hand side of (16.1) arbitrary small, contradicting the bound by α. It follows that $\tilde{\boldsymbol{\lambda}} \geq \mathbf{0}$ and $\nu \geq 0$. Condition (16.2) simply means that $\nu t \leq \alpha$

for all $t < p^*$, so that $\nu p^* \leq \alpha$. Combining this bound with (16.1), we get the two inequalities, valid for any $\boldsymbol{x} \in \mathcal{D}$,

$$\sum_{i=1}^{m} \tilde{\lambda}_i f_i(\boldsymbol{x}) + \tilde{\boldsymbol{\mu}}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \nu f(\boldsymbol{x}) \geq \alpha \geq \nu p^*. \tag{16.3}$$

Note that the left-hand side has the form of a Lagrangian function scaled by ν. If $\nu > 0$ we can divide by μ and set $\lambda_i = \tilde{\lambda}_i/\nu$, $\mu_i = \tilde{\mu}_i/\nu$, to obtain

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \geq p^*.$$

By weak duality we have $p^* \geq g(\boldsymbol{\lambda}, \boldsymbol{\mu})$, so that we get strong duality if $\nu > 0$. If $\nu = 0$, then (16.3) implies

$$\sum_{i=1}^{m} \tilde{\lambda}_i f_i(\boldsymbol{x}) + \tilde{\boldsymbol{\mu}}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \geq 0, \tag{16.4}$$

and for a point $\tilde{\boldsymbol{x}}$ satisfying the conditions $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $f_i(\boldsymbol{x}) < 0$ for $1 \leq i \leq m$, this means that $\tilde{\boldsymbol{\lambda}} = \boldsymbol{0}$. As $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\mu}}, \nu) \neq \boldsymbol{0}$, we have $\boldsymbol{\mu} \neq \boldsymbol{0}$. Since $\tilde{\boldsymbol{\mu}}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \geq 0$ and $= 0$ for some $\tilde{\boldsymbol{x}}$ in the interior of $\mathcal{D}$, there must be a $\boldsymbol{x}$ in the interior such that $\tilde{\boldsymbol{\mu}}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) < 0$, in contradiction to (16.4) (unless $\tilde{\boldsymbol{\mu}}^\top \boldsymbol{A} = \boldsymbol{0}$, which would contradict the condition that $\boldsymbol{A}$ has maximal rank p). □

Example 16.3. The problem minimize e^{-x} subject to $x^2/y \leq 0,\ y \geq 0$ is an example of a convex problem that does not satisfy strong duality.

Example 16.4. Consider the problem

$$\text{minimize} \quad \boldsymbol{x}^\top \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

The Lagrangian is $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}) = \boldsymbol{x}^\top \boldsymbol{x} + \boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$. For any $\boldsymbol{\mu}$, we can find the infimum

$$g(\boldsymbol{\mu}) = \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu})$$

by setting the derivative of the Lagrangian to $\boldsymbol{x}$ to zero:

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}) = 2\boldsymbol{x} + \boldsymbol{A}^\top \boldsymbol{\mu} = \boldsymbol{0},$$

which gives the solution

$$\boldsymbol{x} = -\frac{1}{2} \boldsymbol{A}^\top \boldsymbol{\mu}.$$

The dual function is therefore

$$g(\boldsymbol{\mu}) = -\frac{1}{4} \boldsymbol{\mu}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{\mu} - \boldsymbol{b}^\top \boldsymbol{\mu}.$$

As the negative of a positive semidefinite quadratic function, it is concave. Moreover, we get the lower bound

$$-\frac{1}{4} \boldsymbol{\mu}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{\mu} - \boldsymbol{b}^\top \boldsymbol{\mu} \leq \inf\{\boldsymbol{x}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}.$$

The problem we started out with is convex, and if we assume that there exists a feasible primal point, then the above inequality is in fact an equality by Slater's conditions.

16.2 Karush-Kuhn-Tucker optimality conditions

Consider now a not necessarily convex problem of the form

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \end{array} \tag{16.1}$$

If p^* is the optimal solution of (16.1) and $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ dual variables, then we have seen that (this holds even in the non-convex case)

$$p^* \geq g(\boldsymbol{\lambda}, \boldsymbol{\mu}).$$

From this is follows that for any primal feasible point $\boldsymbol{x}$,

$$f(\boldsymbol{x}) - p^* \leq f(\boldsymbol{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu}).$$

The difference $f(\boldsymbol{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ between the primal objective function at a primal feasible point and the dual objective function at a dual feasible point is called the *duality gap* at $\boldsymbol{x}$ and $(\boldsymbol{\lambda}, \boldsymbol{\mu})$. For any such points we know that

$$p^*, q^* \in [g(\boldsymbol{\lambda}, \boldsymbol{\mu}), f(\boldsymbol{x})],$$

and if the gap is small we have a good approximation of the primal and dual optimal values. The duality gap can be used in iterative algorithms to define stopping criteria: if the algorithm generates a sequence of primal-dual variables $(\boldsymbol{x}^k, \boldsymbol{\lambda}^k, \boldsymbol{\mu}^k)$, then we can stop if the duality gap is less than, say, a predefined tolerance ε.

Now suppose that we have points $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ such that the duality gap is zero. Then

$$\begin{aligned} f(\boldsymbol{x}^*) &= g(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \\ &= \inf_{\boldsymbol{x}} \left(f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i^* f_i(\boldsymbol{x}) + \sum_{j=1}^{p} \mu_j^* h_j(\boldsymbol{x}) \right) \\ &\leq f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* f_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j^* h_j(\boldsymbol{x}^*) \\ &\leq f(\boldsymbol{x}^*), \end{aligned}$$

where the last inequality follows from the fact that $h_j(\boldsymbol{x}^*) = 0$ and $\lambda_i^* f_i(\boldsymbol{x}^*) \leq 0$ for $1 \leq j \leq p$ and $1 \leq i \leq m$. It follows that the inequalities are in fact equalities. From the identity

$$f(\boldsymbol{x}^*) = f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* f_i(\boldsymbol{x}^*)$$

and $\lambda_i^* \geq 0$ and $f_i(\boldsymbol{x}^*) \leq 0$ we also conclude that at such optimal points,

$$\lambda_i^* f_i(\boldsymbol{x}^*) = 0, \quad 1 \leq i \leq m.$$

This condition is known as *complementary slackness*. From the above we also see that $\boldsymbol{x}^*$ minimizes the Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$, so that the gradient of that function is zero:

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}.$$

Collecting these conditions (primal and dual feasibility, complementary slackness, vanishing gradient), we arrive at a set of optimality conditions known as the Karush-Kuhn-Tucker (KKT) conditions.

Theorem 16.5. *(KKT conditions) Let $\boldsymbol{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be primal and dual optimal solutions of* (16.1) *with zero duality gap. The the following conditions are satisfied:*

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}^*) &\leq \mathbf{0} \\ \boldsymbol{h}(\boldsymbol{x}^*) &= \mathbf{0} \\ \boldsymbol{\lambda}^* &\geq \mathbf{0} \\ \lambda_i^* f_i(\boldsymbol{x}^*) &= 0, \ 1 \leq i \leq m \\ \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla_{\boldsymbol{x}} f_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j^* \nabla_{\boldsymbol{x}} h_j(\boldsymbol{x}^*) &= \mathbf{0}. \end{aligned}$$

Moreover, if the problem is convex, then any points satisfying the KKT conditions have zero duality gap.

Lecture 17

In this lecture we introduce interior point methods for solving nonlinear convex optimization problems of the form

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{array} \tag{1}$$

with $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{p \times n}$ and $\boldsymbol{f} = (f_1, \ldots, f_m)^\top \colon \mathbb{R}^n \to \mathbb{R}^m$. The particular form of interior point methods we discuss is the *barrier method*, which differs slightly from the primal-dual method discussed for linear programming.

Recall the Karush-Kuhn-Tucker (KKT) conditions

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}^*) &\leq \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x}^* &= \boldsymbol{b} \\ \boldsymbol{\lambda}^* &\geq \boldsymbol{0} \\ \lambda_i^* f_i(\boldsymbol{x}^*) &= 0, \ 1 \leq i \leq m \\ \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla_{\boldsymbol{x}} f_i(\boldsymbol{x}^*) + \boldsymbol{A}^\top \boldsymbol{\mu}^* &= \boldsymbol{0}, \end{aligned} \tag{2}$$

where $\nabla_{\boldsymbol{x}}$ denotes the gradient of a function with respect to $\boldsymbol{x}$. As discussed in the previous lecture, for convex problems these are necessary and sufficient conditions for $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p$ being a set of optimal primal and dual solutions. By this we mean that $\boldsymbol{x}^*$ is a minimizer of (1) and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is a maximizer of the Lagrange dual function $g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ subject to $\boldsymbol{\lambda} \geq 0$, which in turn was defined as the infimum

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}),$$

where $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$ is the Lagrangian. In this section we make the additional assumption that the functions f and f_i, $1 \leq i \leq m$, are two time continuously differentiable. The reason is that we want to apply variants of Newton's method to the KKT conditions. For later reference we list some of the more popular forms of convex optimization problems that fall into our scope, with the associated KKT conditions.

Example 17.1. (LP) For a linear programming problem of the form

$$\begin{aligned} \text{minimize} \quad & \langle \boldsymbol{c}, \boldsymbol{x} \rangle \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned}$$

the KKT conditions have the form

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b} \\ \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{s} &= \boldsymbol{c} \\ \boldsymbol{s} &\geq \boldsymbol{0} \\ \boldsymbol{x} &\geq \boldsymbol{0} \\ x_i s_i &= 0, \ 1 \leq i. \end{aligned}$$

Example 17.2. (QP) A *quadratic programming problem* is a problem with quadratic objective and linear constraints of the form

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + \boldsymbol{r} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{B}\boldsymbol{x} \leq \boldsymbol{c}. \end{aligned}$$

One typical example is portfolio optimization.

Many other problems can be cast in the required form, and we will see later that the framework also encompasses semidefinite programming.

17.1 The logarithmic barrier

The idea of barrier methods is to replace the constraints of the optimization problem (1) with a parametrized system of equality constraints, such that the solutions of the equality constraint system can be found, and converge to the solution of the original system. The first observation is that we can replace the inequality contstrained system

$$\text{minimize} \quad f(\boldsymbol{x}) \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \ f_i(\boldsymbol{x}) \leq 0, \ 1 \leq i \leq m,$$

with the equality constrained, but nonsmooth, problem

$$\text{minimize} \quad f(\boldsymbol{x}) + \sum_{i=1}^{m} I_-(f_i(\boldsymbol{x})), \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

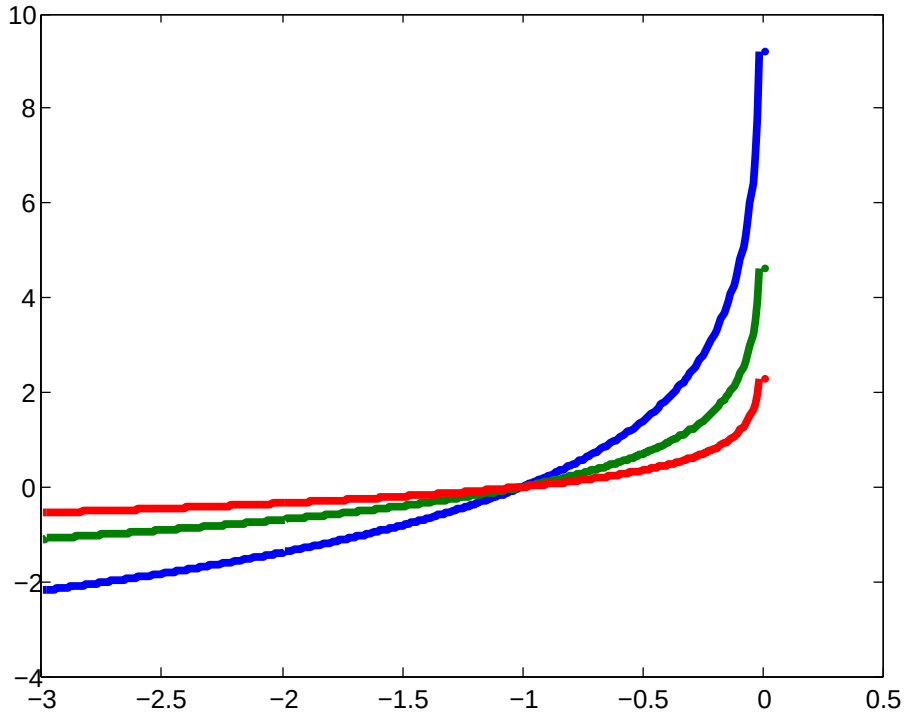
where

$$I_-(t) = \begin{cases} 0 & t \leq 0, \\ \infty & t > 0. \end{cases}$$

Clearly, the objective is $f(\boldsymbol{x})$ whenever the inequality constraints $f_i(\boldsymbol{x}) \leq 0$ are satisfied, and ∞ if not, so that points that do not satisfy the constraints are forbidden. Since this function is not easy to deal with analytically, we *approximate* it with the function

$$\hat{I}(u) := -\frac{1}{t}\ln(-u)$$

for suitable values t. The figure below shows the approximating functions for $t = 0.5, 1, 2$. As is easily seen, as t increases the approximation becomes better. An approximation to the



nonsmooth problem is given by the following equality constrained problem,

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) - \sum_{i=1}^{m} \frac{1}{t}\log(f_i(\boldsymbol{x})) \\ &\text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned}$$

The domain $\mathcal{D}$ of this problem is the set of points such that the inequalities $f_i(\boldsymbol{x}) < 0$ are strictly satisfied. One can also verify that the objective function is convex, giving rise to a convex optimization problem. The function

$$\varphi(\boldsymbol{x}) = -\sum_{i=1}^{m} \log(f_i(\boldsymbol{x}))$$

is called the *logarithmic barrier function* of the system of inequalities $f_i(\boldsymbol{x}) < 0$: the reason is that it prevents candidates $\boldsymbol{x}$ from becoming too close to the boundary of the feasible set by

becomming very large near it. With this notation, we can write the approximate problem as

$$\begin{aligned} &\text{minimize} && tf(\boldsymbol{x}) + \varphi(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{17.1}$$

Note that here we multiplied the objective with t, which does not change the optimal points (but the values). For reference, we record the gradient and Hessian of φ,

$$\begin{aligned} \nabla\varphi &= \sum_{i=1}^{m} \frac{1}{-f_i(\boldsymbol{x})} \nabla f_i(\boldsymbol{x}) \\ \nabla^2\varphi &= \sum_{i=1}^{m} \frac{1}{f_i^2(\boldsymbol{x})} \nabla f_i(\boldsymbol{x}) \nabla f_i(\boldsymbol{x})^\top + \sum_{i=1}^{m} \frac{1}{-\nabla f_i(\boldsymbol{x})} \nabla^2 f_i(\boldsymbol{x}). \end{aligned}$$

Two questions are of importance:

1. How well does a solution of (17.1) approximate one of (1), and
2. How do we solve (17.1)?

17.2 The central path

The set of solutions $\boldsymbol{x}^*(t)$ of (17.1) for $t > 0$ is called the *central path*. Points on the central path are strictly feasible: they satisfy $\boldsymbol{A}\boldsymbol{x}^*(t) = \boldsymbol{b}$ and $f_i(\boldsymbol{x}^*(t)) < 0$. Moreover, by the Lagrange multiplier theorem, they satisfy the optimality condition

$$t\nabla f(\boldsymbol{x}) + \nabla\varphi(\boldsymbol{x}) + \boldsymbol{A}^\top \boldsymbol{\mu}^* = \boldsymbol{0}$$

for some suitable Lagrange multipliers $\boldsymbol{\mu} \in \mathbb{R}^p$. An important property of the central path is that we can derive dual points $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$, and therefore (by Lagrange duality) lower bounds on the optimal value p^* of (1), from any point $\boldsymbol{x}^*$ on the central path. In fact, the central path can be derived from the solutions of a variant of the KKT conditions.

Lemma 17.3. *A point $\boldsymbol{x}$ is equal to a point $\boldsymbol{x}^*(t)$ on the central path if and only if there exist dual multipliers λ and μ such that the following conditions are satisfied.*

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}^*) &\leq \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x}^* &= \boldsymbol{b} \\ \boldsymbol{\lambda}^* &\geq \boldsymbol{0} \\ -\lambda_i^* f_i(\boldsymbol{x}^*) &= \frac{1}{t},\ 1 \leq i \leq m \\ \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla_{\boldsymbol{x}} f_i(\boldsymbol{x}^*) + \boldsymbol{A}^\top \boldsymbol{\mu}^* &= \boldsymbol{0}, \end{aligned}$$

The proof is left as an exercise. Note the analogy with the definition of the central path in the case of linear programming! Accordingly, the solution methods discussed in the context of linear programming carry over to the nonlinear case. We conclude by sketching the *barrier method*.

1. Start with a strictly feasible $\boldsymbol{x}$ and $t := t^{(0)} > 0$, $\mu > 1$ and tolerance $\varepsilon > 0$;
2. Compute $\boldsymbol{x}^*(t)$ by minimizing $tf + \varphi$ subject to $\boldsymbol{Ax} = \boldsymbol{b}$, starting with $\boldsymbol{x}$;
3. Update $\boldsymbol{x} = \boldsymbol{x}^*(t)$;
4. Stop if $m/t < \varepsilon$, otherwise set $t := \mu t$ (increase t).

The only unexplained part is how to solve the equality constrained minimization problem. One way is to use Newton's method on the optimality conditions of this problem.

Lecture 18

In this lecture we return to the task of classification. As seen earlier, examples include spam filters, letter recognition, or text classification. In this lecture we introduce a popular method for classification, **Support Vector Machines (SVMs)**, from the point of view of convex optimization.

18.1 Linear Support Vector Machines

In the simplest case there is a set of labels $\mathcal{Y} = \{-1, 1\}$ and the set of training points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is *linearly separable*: this means that there exists an affine hyperplane $h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ such that $h(\boldsymbol{x}_i) > 0$ if $y_i = 1$ and $h(\boldsymbol{x}_j) < 0$ if $y_j = -1$. We call the points for which $y_i = 1$ *positive*, and the ones for which $y_j = -1$ *negative*. The problem of finding such a hyperplane can be posed as a linear programming feasibility problem as follows: we look for a vector of *weights* $\boldsymbol{w}$ and a *bias term* b (together a $(p+1)$-dimensional vector) such that

$$\boldsymbol{w}^\top \boldsymbol{x}_i + b \geq 1, \text{ for } y_i = 1, \quad \boldsymbol{w}^\top \boldsymbol{x}_j + b \leq -1, \text{ for } y_j = -1.$$

Note that we can replace the $+1$ and -1 with any other positve or negative quantity by rescaling the $\boldsymbol{w}$ and $\boldsymbol{b}$, so this is just convention. We can also describe the two inequalities concisely as

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 \geq 0. \tag{18.1}$$

A hyperplane separating the two point sets will in general not be unique. As we want to use the linear classifier on new, yet unknown data, we want to find a separating hyperplane with best possible **margin**. Let d_+ and d_- denote the distance of a separating hyperplane to the closest positive and closest negative point, respectively. The quantity $d = d_+ + d_-$ is then called the margin or the classifier, and we want to find a hyperplane with largest possible margin.

We next show that the margin for a separating hyperplane that satisfies (18.1) is $d = 2/\|\boldsymbol{w}\|_2$. Given a hyperplane H described in (18.1) and a point $\boldsymbol{x}$ such that we have the equality $\boldsymbol{w}^\top \boldsymbol{x} + b = 1$ (the point is as close as possible to the hyperplane, also called a **support vector**), the distance of that point to the hyperplane can be computed by first taking the difference of $\boldsymbol{x}$ with a point $\boldsymbol{p}$ on H (an *anchor*), and then computing the dot product of $\boldsymbol{x} - \boldsymbol{p}$ with the unit vector $\boldsymbol{w}/\|\boldsymbol{w}\|$ orthogonal to H (see Calculus and Vectors A, math10121).

As anchor point $\boldsymbol{p}$ we can just choose a multiple $c\boldsymbol{w}$ that is on the plane, i.e., that satisfies $\langle \boldsymbol{w}, c\boldsymbol{w} \rangle + b = 0$. This implies that $c = -b/\|\boldsymbol{w}\|^2$, and consequently $\boldsymbol{p} = -(b/\|\boldsymbol{w}\|^2)\boldsymbol{w}$. The

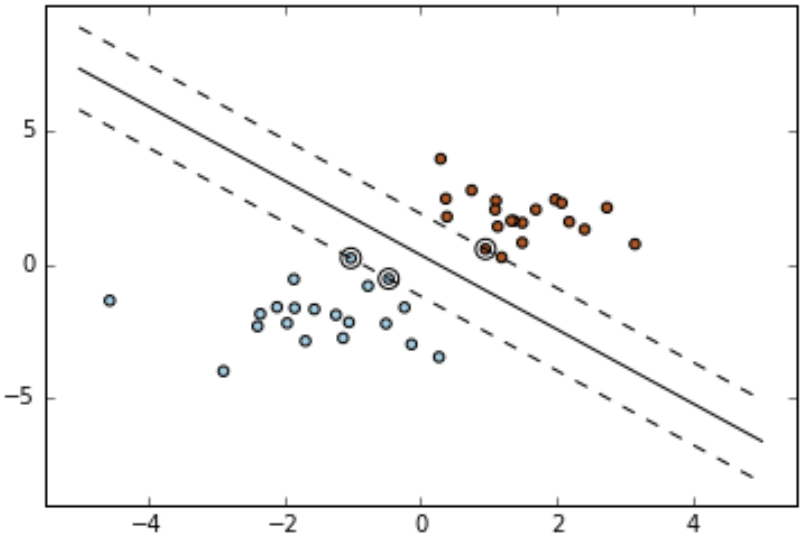


Figure 18.22: A hyperplane separating two sets of points with margin and support vectors.

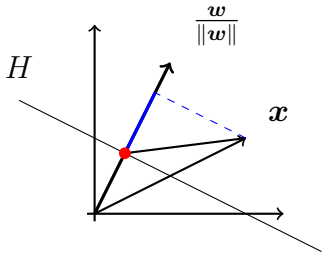


Figure 18.23: Computing the distance to the hyperplane

distance is then

$$\begin{aligned} d_+ &= \langle \boldsymbol{x} + \frac{b}{\|\boldsymbol{w}\|^2}\boldsymbol{w}, \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\rangle = \frac{\langle \boldsymbol{x}, \boldsymbol{w}\rangle}{\|\boldsymbol{w}\|} + \frac{b}{\|\boldsymbol{w}\|^2}\langle \boldsymbol{w}, \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\rangle \\ &= \frac{1-b}{\|\boldsymbol{w}\|} + \frac{b}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}. \end{aligned}$$

Similarly, we get $d_- = 1/\|\boldsymbol{w}\|$. The margin of this particular separating hyperplane is thus $d = 2/\|\boldsymbol{w}\|$. If we want to find a hyperplane with *largest* margin, we thus have to solve the quadratic optimization problem

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{subject to} && y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 \geq 0, \quad 1 \leq i \leq n. \end{aligned}$$

Note that b is also an unknown variable in this problem! The factor $1/2$ in the objective function is just to make the gradient look nicer. The Lagrangian of this problem is

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{w}^\top \boldsymbol{x}_i - \lambda_i y_i b + \lambda_i \\ &= \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} - \boldsymbol{\lambda}^\top \boldsymbol{X}\boldsymbol{w} - b\boldsymbol{\lambda}^\top \boldsymbol{y} + \sum_{i=1}^{m} \lambda_i,\end{aligned}$$

where we denote by $\boldsymbol{X}$ the matrix with the $y_i \boldsymbol{x}_i^\top$ as rows. We can then write the conditions on the gradient with respect to $\boldsymbol{w}$ and b of the Lagrangian as

$$\begin{aligned}\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \boldsymbol{w} - \boldsymbol{X}^\top \boldsymbol{\lambda} = \boldsymbol{0} \\ \frac{\partial \mathcal{L}}{\partial b}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \boldsymbol{y}^\top \boldsymbol{\lambda} = 0.\end{aligned} \tag{18.2}$$

Replacing $\boldsymbol{w}$ by $\boldsymbol{X}^\top \boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^\top \boldsymbol{y}$ by 0 in the Lagrangian function then gives the expression for the Lagrange dual $g(\boldsymbol{\lambda})$,

$$g(\boldsymbol{\lambda}) = -\frac{1}{2}\boldsymbol{\lambda}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{\lambda} + \sum_{i=1}^{m} \lambda_i.$$

Finally, changing the sign and the maximum with a minimum, we can formulate the Lagrange dual optimization problem as

$$\text{minimize} \quad \frac{1}{2}\boldsymbol{\lambda}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{\lambda} - \boldsymbol{\lambda}^\top \boldsymbol{e} \quad \text{subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}, \tag{18.3}$$

where $\boldsymbol{e}$ is the vector of all ones.

Note that there is one dual variable λ_i per data point $\boldsymbol{x}_i$. We can find the optimal value by solving the dual problem (18.3), but that does not give us automatically the weights $\boldsymbol{w}$ and the bias b. We can find the weights by $\boldsymbol{w} = \boldsymbol{X}^\top \boldsymbol{\lambda}$. As for b, this is best determined from the KKT conditions of the problem. These can be written by combining the constraints of the primal problem with the conditions on the gradient of the Lagrangian (18.2), the condition $\boldsymbol{\lambda} \geq \boldsymbol{0}$, and complementary slackness as

$$\begin{aligned}\boldsymbol{X}\boldsymbol{w} + b\boldsymbol{y} - \boldsymbol{e} &\geq \boldsymbol{0} \\ \boldsymbol{\lambda} &\geq \boldsymbol{0} \\ \lambda_i(1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) &= 0 \text{ for } 1 \leq i \leq n \\ \boldsymbol{w} - \boldsymbol{X}^\top \boldsymbol{\lambda} &= \boldsymbol{0} \\ \boldsymbol{y}^\top \boldsymbol{\lambda} &= 0.\end{aligned}$$

To get b, we can choose one of the equations in which $\lambda_i \neq 0$, and then find b by setting $b = y_i(1 - y_i \boldsymbol{w}^\top \boldsymbol{x}_i)$. With the KKT conditions written down, we can go about solving the problem of finding a maximum margin linear classifier using methods such as the barrier method.

18.2 Extensions

So far we looked at the particularly simple case where (a) the data falls into two classes, (b) the points can actually be well separated, and (c) they can be separated by an affine hyperplane. In reality, these three assumptions may not hold. We briefly discuss extensions of the basic model to account for the three situations just mentioned.

Non-exact separation

What happens when the data can not be separated by a hyperplane? In this case the constraints can not be satisfied: there is no feasible solution to the problem. We can still modify the problem to allow for *misclassification*: we want to find a hyperplane that separates the two point sets as good as possible, but we allow for some mistakes.

One approach is to add an additional set of n *slack variables* $s_1, \ldots, s_n$, and modify the constraints to

$$\boldsymbol{w}^\top \boldsymbol{x}_i + b \geq 1 - s_i, \text{ for } y_i = 1, \quad \boldsymbol{w}^\top \boldsymbol{x}_j + b \leq -1 + s_j, \text{ for } y_j = -1, \quad s_i \geq 0.$$

The i-th data point can land on the wrong side of the hyperplane if $s_i > 1$, and consequently the sum $\sum_{i=1}^n s_i$ is an upper bound on the number of errors possible. If we want to minimize the number of misclassified points, we may want to minimize this upper bound, so a sensible choice for objective function would be to add a multiple of this sum. The new problem thus becomes

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + \mu \sum_{j=1}^n s_j \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 + s_i \geq 0, \quad 1 \leq i \leq n \\ & s_i \geq 0, \quad 1 \leq i \leq n, \end{aligned}$$

for some parameter μ. The Lagrangian of this problem and the KKT conditions can be derived in a similar way as in the separable case and are left as an exercise.

Non-linear separation and kernels

The key to extending SVMs from linear to non-linear separation is the observation that the dual form of the optimization problem (18.3) depends only on the dot products $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ of the data points. In fact, the (i, j)-th entry of the matrix $\boldsymbol{X}\boldsymbol{X}^\top$ is precisely $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$!

If we map our data into a higher (possibly infinite) dimensional space $\mathcal{H}$,

$$\varphi \colon \mathbb{R}^p \to \mathcal{H},$$

and consider the data points $\varphi(\boldsymbol{x}_i)$, $1 \leq i \leq n$, then applying the support vector machine to these higher dimensional vectors will only depend on the dot products

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_j) \rangle.$$

The function K is called a **kernel function**. A typical example, often used in practice, is the Gaussian radial basis function (RBF),

$$K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2/2\sigma^2}.$$

Note that we *don't need to know how the function φ looks like*! In the equation for the hyperplane we simply replace $\boldsymbol{w}^\top \boldsymbol{x}$ with $K(\boldsymbol{w}, \boldsymbol{x})$. The only difference now is that the function ceases to be linear in $\boldsymbol{x}$: we get a non-linear decision boundary.

Multiple classes

One is often interested in classifying data into more than two classes. There are two simple ways in which support vector machines can be extended for such problems: one-vs-one and one-vs-rest. In the one-vs-one case, given k classes, we train one classifier for each pair of classes in the training data, obtaining a total of $k(k-1)/2$ classifiers. When it comes to prediction, we apply each of the classifiers to our test data and choose the class that was chosen the most among all the classifiers. In the one-vs-rest approach, each train k binary classifiers: in each one, one class corresponds to a chosen class, and the second class corresponds to the rest. By associating confidence scores to each classifier, we choose the one with the highest confidence score.

Example 18.1. An example that uses all three extensions mentioned is handwritten digit recognition. Suppose we have a series of pixels, each representing a number, and associated labels $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. We would like to train a support vector machine to recognize new digits. Given the knowledge we have, we can implement this task using standard optimization software such as CVXPY. Luckily, there are packages that have this functionality already implemented, such as the SCIKIT-LEARN package for Python. We illustrate its functioning below. The code also illustrates some standard procedures when tackling a machine learning problem:

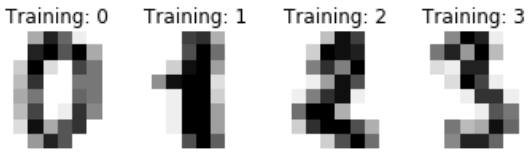
- **Separate** the data set randomly into *training data* and *test data*;
- **Create** a support vector classifier with optional parameters;
- **Train** (using FIT) the classifier with the training data;
- **Predict** the response using the test data and compare with the true response;
- **Report** the results.

An important aspect to keep in mind is that when testing the performance using the test data, we should compare the classification accuracy to a naive baseline: if, for example, 80% of the test data is classified as $+1$, then making a prediction of $+1$ for all the data will give us an accuracy of 80%; in this case, we would want our classifier to perform considerably better than getting the right answer 80% of the time!

In [28]:
```
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
from sklearn import svm, datasets, metrics
from sklearn.model_selection import train_test_split
```

In [29]:
```
digits = datasets.load_digits()

# Display images and labels
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)

# Turn images into 1-D arrays
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create classifier
svc = svm.SVC(gamma=0.001)

# Randomly split data into train and test set
X_train, X_test, y_train, y_test = train_test_split(data,
 digits.target, test_size = 0.4, random_state=0)
svc.fit(X_train, y_train)
```

Out [2]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.001,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```



Now apply prediction to test set and report performance.

In [3]:
```
predicted = svc.predict(X_test)
print("Classification report for classifier %s:\n%s\n"
      % (svc, metrics.classification_report(y_test, predicted)))
```

Out [3]:
```
Classification report for classifier SVC(C=1.0, cache_size=200,
 class_weight=None, coef0=0.0,vdecision_function_shape=None,
 degree=3, gamma=0.001, kernel='rbf', max_iter=-1, probability=False,
 random_state=None, shrinking=True, tol=0.001, verbose=False):
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        60
          1       0.97      1.00      0.99        73
          2       1.00      0.97      0.99        71
          3       1.00      1.00      1.00        70
          4       1.00      1.00      1.00        63
          5       1.00      0.98      0.99        89
          6       0.99      1.00      0.99        76
          7       0.98      1.00      0.99        65
          8       1.00      0.99      0.99        78
          9       0.99      1.00      0.99        74

avg / total       0.99      0.99      0.99       719
```

In [4]:
```
import skimage
from skimage import data
from skimage.transform import resize
from skimage import io
import os
```

Now try this out on some original data!

In [5]:
```
mydigit1 = io.imread('images/digit9.png')
mydigit2 = io.imread('images/digit4.png')
plt.figure(figsize=(8, 4))
plt.subplot(1,2,1)
plt.imshow(mydigit1, cmap=plt.cm.gray_r, interpolation='nearest')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(mydigit2, cmap=plt.cm.gray_r, interpolation='nearest')
plt.axis('off')
plt.show()
```

In [6]:
```
smalldigit1 = resize(mydigit1, (8,8))
smalldigit2 = resize(mydigit2, (8,8))
mydigits = np.concatenate((np.round(15*(np.ones((8,8))-
                        smalldigit1[:,:,0])).reshape((64,1)).T,
            np.round(15*(np.ones((8,8))-
            smalldigit2[:,:,0])).reshape((64,1)).T),axis=0)
# After some preprocessing, make prediction
guess = svc.predict(mydigits)
print guess
```

```
[9 4]
```

Part V

Semidefinite Programming

Lecture 19

Semidefinite programming (SDP) is a far-reaching generalization of linear programming (LP), in which vectors are replaced by matrices and the non-negativity condition is replaced with the condition that the matrices be positive semidefinite.

19.1 Semidefinite programming

Recall the linear programming formulation

$$\begin{array}{ll} \text{minimize} & \langle \boldsymbol{c}, \boldsymbol{x} \rangle \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \tag{LP}$$

with $\boldsymbol{c}, \boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$. Semidefinite programming is a far-reaching generalization of linear programming, in which the vector space $\mathbb{R}^n$ is replaced by the vector space of real symmetric matrices,

$$\mathrm{SYM}_n = \{\boldsymbol{X} \in \mathbb{R}^{n \times n} : x_{ij} = x_{ji}, 1 \leq i < j \leq n\},$$

with the trace inner product,

$$\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \boldsymbol{X} \bullet \boldsymbol{Y} = \mathrm{tr}(\boldsymbol{X}\boldsymbol{Y}) = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} y_{ij},$$

where $\mathrm{tr}(\boldsymbol{X}) = \sum_{i=1}^{n} x_{ii}$ is the *trace* of a matrix $\boldsymbol{X}$. The matrix $\boldsymbol{A}$ in (LP) is replaced by a linear map $\mathcal{A} \colon \mathrm{SYM}_n \to \mathbb{R}^m$. So far, this new setting does not fall out of the general framework of linear programming (note that (LP) does not make any reference on the nature of the inner product and the linear map $\boldsymbol{A}$). The point of departure of semidefinite programming is that the inequality constraints $\boldsymbol{x} \geq \boldsymbol{0}$ are replaced with the constraint $\boldsymbol{X} \succeq \boldsymbol{0}$, meaning that $\boldsymbol{X}$ is positive semidefinite. A *semidefinite programming* (SDP) problem thus has the form

$$\begin{array}{ll} \text{minimize} & \boldsymbol{C} \bullet \boldsymbol{X} \\ \text{subject to} & \mathcal{A}(\boldsymbol{X}) = \boldsymbol{b} \\ & \boldsymbol{X} \succeq \boldsymbol{0}. \end{array} \tag{SDP-P}$$

Remark 19.1. Recall that $\boldsymbol{X} \succeq \mathbf{0}$ means that for all $\boldsymbol{v} \in \mathbb{R}^n$, $\boldsymbol{v}^\top \boldsymbol{X} \boldsymbol{v} \geq 0$. Therefore, SDP can be viewed as linear programming with an uncountable number of inequality constraints. Recall also that a symmetric matrix is positive semidefinite if and only if the (necessarily real) eigenvalues are non-negative. While the feasible sets of linear programming are polyhedra, the feasible sets of SDP are called *spectrahedra*, since they are "polyhedra in the spectrum (set of eigenvalues)".

The vector space SYM_n has dimension $d := n(n+1)/2$, the number of entries on or above the main diagonal. Therefore, if we identify $\mathrm{SYM}_n \cong \mathbb{R}^d$, the linear map $\mathcal{A}$ can be identified with a "big" $m \times d$ matrix. Alternatively, the condition $\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}$ can be seen as collection of m linear constraints, each of the form $\boldsymbol{A}_i \bullet \boldsymbol{X} = b_i$ with a symmetric matrix $\boldsymbol{A}_i$, so that we can express (SDP-P) as

$$\begin{array}{ll} \text{minimize} & \boldsymbol{C} \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i, \quad 1 \leq i \leq m, \\ & \boldsymbol{X} \succeq \mathbf{0}. \end{array}$$

Example 19.2. Consider an SDP with $n = 3$ and $m = 2$ (3×3 symmetric matrices and 2 constraints), with the following data

$$\boldsymbol{A}_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{pmatrix}, \quad \boldsymbol{A}_2 = \begin{pmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 11 \\ 19 \end{pmatrix}.$$

The decision variable is the symmetric matrix

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{12} & x_{22} & x_{23} \\ x_{13} & x_{23} & x_{33} \end{pmatrix}.$$

The objective function is

$$\boldsymbol{C} \bullet \boldsymbol{X} = x_{11} + 4x_{12} + 6x_{13} + 9x_{22} + 7x_{33},$$

where we used the symmetry $x_{ij} = x_{ji}$ to simplify the expression. Working out the constraints $\boldsymbol{A}_i \bullet \boldsymbol{X} = b_i$ in the same way, we arrive at the following form of the SDP:

$$\begin{array}{ll} \text{minimize} & x_{11} + 4x_{12} + 6x_{13} + 9x_{22} + 7x_{33} \\ \text{subject to} & x_{11} + 2x_{13} + 3x_{22} + 14x_{23} + 5x_{33} = 11 \\ & 4x_{12} + 16x_{13} + 6x_{22} + 4x_{33} = 19 \\ & \boldsymbol{X} \succeq \mathbf{0}. \end{array}$$

As mentioned in Remark 19.1, were it not for the positive semidefinite constraint, the above would be an old-fashioned LP in the variables $\boldsymbol{x} = (x_{11}, x_{12}, x_{13}, x_{22}, x_{23}, x_{33})^\top$.

Linear programming is a special case of semidefinite programming. To see this, start with the problem (LP) and create the new matrices

$$\boldsymbol{A}_i = \mathrm{diag}(a_{i1}, \ldots, a_{in}), \quad \boldsymbol{C} = \mathrm{diag}(c_1, \ldots, c_n).$$

Then the problem (LP) is equivalent to the SDP

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{C} \bullet \boldsymbol{X} \\ \text{subject to} \quad & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i, \ 1 \leq i \leq m, \\ & x_{ij} = 0, \ i < j, \\ & \boldsymbol{X} \succeq \boldsymbol{0}. \end{aligned}$$

In fact, any solution of the above has the form $\boldsymbol{X} = \operatorname{diag}(x_{11}, \ldots, x_{nn})$, which we identify with a vector $\boldsymbol{x} \in \mathbb{R}^n$, with the semidefinite constraint, applied to the diagonal matrix, translating into $x_i \geq 0$ for $1 \leq i \leq n$.

19.2 Semidefinite programming duality

Just as with linear programming, we can formulate a dual problem

$$\begin{aligned} \text{maximize} \quad & \langle \boldsymbol{y}, \boldsymbol{b} \rangle \\ \text{subject to} \quad & \sum_{i=1}^{m} y_i \boldsymbol{A}_i + \boldsymbol{S} = \boldsymbol{C} \\ & \boldsymbol{S} \succeq \boldsymbol{0}. \end{aligned} \tag{SDP-D}$$

Example 19.3. The dual to the problem in Example 19.2 is the problem

$$\begin{aligned} \text{maximize} \quad & 11y_1 + 19y_2 \\ \text{subject to} \quad & y_1 \begin{pmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{pmatrix} + y_2 \begin{pmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{pmatrix} + \boldsymbol{S} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{pmatrix}, \\ & \boldsymbol{S} \succeq \boldsymbol{0}. \end{aligned}$$

One can also write the constraints more compactly by eliminating $\boldsymbol{S}$, in terms of a matrix whose entries are linear functions of y_1 and y_2 and which is required to be positive semidefinite.

We call $\boldsymbol{X}$ a feasible matrix for (SDP-P) if it satisfies the constraints, and $(\boldsymbol{y}, \boldsymbol{S})$ a feasible pair for (SDP-D) if $\boldsymbol{y}$ and $\boldsymbol{S}$ satisfy the constraints. As with the nonlinear optimization setting from Lecture 16, we call the *duality gap* the difference $\boldsymbol{C} \bullet \boldsymbol{X} - \langle \boldsymbol{b}, \boldsymbol{y} \rangle$ between the primal and dual objective functions at feasible points.

Theorem 19.4. *Let $\boldsymbol{X}$ be a feasible matrix for* (SDP-P) *and let $(\boldsymbol{y}, \boldsymbol{S})$ be a feasible pair for* (SDP-D)*. Then the* duality gap *satisfies*

$$\boldsymbol{C} \bullet \boldsymbol{X} - \langle \boldsymbol{b}, \boldsymbol{y} \rangle = \boldsymbol{S} \bullet \boldsymbol{X} \geq 0.$$

Moreover, if the duality gap is zero, then $\boldsymbol{X}$ and $(\boldsymbol{y}, \boldsymbol{S})$ are optimal points for (SDP-P) *and* (SDP-D)*, respectively.*

Recall that $\boldsymbol{X} \in \mathrm{SYM}_n$ if and only if $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^\top$, with $\boldsymbol{D}$ diagonal (containing the eigenvalues) and $\boldsymbol{Q}$ orthogonal (that is, $\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}$). If in addition $\boldsymbol{X} \succeq \boldsymbol{0}$, then the diagonal entries are nonnegative. We need some more facts about positive semidefinite matrices.

Lemma 19.5. *Let $\boldsymbol{X} \in \mathcal{S}_+^n$. Then:*

1. *The diagonal entries satisfy $x_{ii} \geq 0$ for $1 \leq i \leq n$;*
2. *If $x_{ii} = 0$ for some i, then $x_{ij} = x_{ji} = 0$ for all $1 \leq j \leq n$ (if a diagonal is zero, then the whole corresponding row and column is zero).*

Proof of Theorem 19.4. (Optional.) Note that

$$\boldsymbol{C} \bullet \boldsymbol{X} - \langle \boldsymbol{b}, \boldsymbol{y} \rangle = \sum_{i=1}^{m} y_i \boldsymbol{A}_i \bullet \boldsymbol{X} + \boldsymbol{S} \bullet \boldsymbol{X} - \langle \boldsymbol{b}, \boldsymbol{y} \rangle = \boldsymbol{S} \bullet \boldsymbol{X},$$

since $\sum_{i=1}^m y_i \boldsymbol{A}_i \bullet \boldsymbol{X} = \sum_{i=1}^m y_i b_i = \langle \boldsymbol{y}, \boldsymbol{b} \rangle$. We next show that if $\boldsymbol{X} \succeq \boldsymbol{0}$ and $\boldsymbol{S} \succeq \boldsymbol{0}$, then $\boldsymbol{X} \bullet \boldsymbol{S} \geq 0$. Assume $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^\top$ and $\boldsymbol{S} = \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top$, with $\boldsymbol{Q}, \boldsymbol{P}$ orthogonal and $\boldsymbol{D}, \boldsymbol{E}$ diagonal. Then

$$\boldsymbol{X} \bullet \boldsymbol{S} = \operatorname{tr}(\boldsymbol{S}\boldsymbol{X}) = \operatorname{tr}(\boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^\top \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top) = \operatorname{tr}(\boldsymbol{D}\boldsymbol{Q}^\top \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q}),$$

where we used that $\operatorname{tr}(\boldsymbol{A}\boldsymbol{B}) = \operatorname{tr}(\boldsymbol{B}\boldsymbol{A})$. The last expression equals

$$\sum_{i=1}^{n} \boldsymbol{D}_{ii} (\boldsymbol{Q}^\top \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q})_{ii} \geq 0, \tag{19.1}$$

since $\boldsymbol{D}_{ii} \geq 0$ and $\boldsymbol{Q}^\top \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q} \succeq \boldsymbol{0}$, which implies that the diagonal entries are also nonnegative by Lemma 19.5.

Now if $\boldsymbol{X} \bullet \boldsymbol{S} = 0$, then by (19.1), $\sum_{i=1}^n \boldsymbol{D}_{ii} (\boldsymbol{Q}^\top \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q})_{ii} = 0$. This means that for each i, either $D_{ii} = 0$, or $(\boldsymbol{Q}^\top \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q})_{ii} = 0$. In the latter case, by Lemma 19.5, the whole j-th row of this matrix is zero. If follows that

$$\boldsymbol{D}(\boldsymbol{Q}^\top \boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q}) = (\boldsymbol{D}\boldsymbol{Q}^\top)(\boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q}) = \boldsymbol{0}.$$

Since $\boldsymbol{A}\boldsymbol{B} = \boldsymbol{0}$ implies $\boldsymbol{B}\boldsymbol{A} = \boldsymbol{0}$, we get that $\boldsymbol{X}\boldsymbol{S} = (\boldsymbol{P}\boldsymbol{E}\boldsymbol{P}^\top \boldsymbol{Q})(\boldsymbol{D}\boldsymbol{Q}^\top) = \boldsymbol{0}$. □

As with nonlinear convex optimization, some mild conditions (also known as Slater's condition) ensure that the primal and dual optimal values coincide.

Theorem 19.6. *Let p^* be the optimal value of* (SDP-P) *and d^* the optimal value of* (SDP-D)*. If there exists a feasible matrix $\boldsymbol{X} \succ \boldsymbol{0}$ for* (SDP-P)*, and a feasible pair $(\boldsymbol{y}, \boldsymbol{S})$ for* (SDP-D) *with $\boldsymbol{S} \succ \boldsymbol{0}$, then $p^* = d^*$.*

The theory of interior point methods carries over to semidefinite programming, giving us efficient (in theory and practice) algorithms for solving such problem. In the next lecture we will address some concrete problems that can be solved efficiently using semidefinite programming, and for which linear or quadratic programming are not enough.

Lecture 20

The standard form of a semidefinite programming problem is

$$\begin{array}{ll} \text{minimize} & \boldsymbol{C} \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i, \quad 1 \leq i \leq m, \\ & \boldsymbol{X} \succeq \boldsymbol{0}, \end{array} \tag{SDP-P}$$

where $\boldsymbol{C}$ and $\boldsymbol{A}_i$, $1 \leq i \leq m$, are symmetric $n \times n$ matrices (elements of the vector space SYM_n). At first sight, it is not clear why solving such problem should be of interest. In this lecture we discuss a useful application to *discrete* optimization problems that are usually difficult in practice.

20.1 Semidefinite relaxation

A graph is a pair $G = (V, E)$, where $V = \{1, \ldots, n\}$ consists of the *vertices*, and $E \subseteq V \times V$ consists of *edges* connecting some of the vertices. We will consider undirected edges, i.e., (i, j) will be the same edge as (j, i).

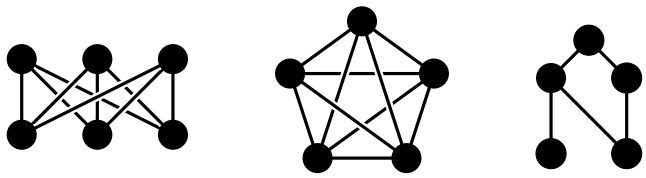


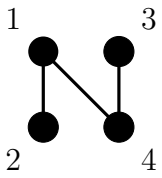
Figure 20.24: Some graphs

Note that for graphs, we only care about the incidences (which node is connected to which other node), the images are just visualizations and the distances and placements of nodes are pure convenience. Graphs are an important tool in areas such as network analysis, and many discrete computational problems can be formulated as graph problems. One such problem is MAXCUT. Given a graph $G = (V, E)$, a *cut* is a pair $(S, V \backslash S)$, where $S \subseteq V$ is a subset of vertices. The *edge set* of the cut is the set

$$E(S, V \backslash S) = \{e \in E : e \text{ connects a vertex in } S \text{ with a vertex in } V \backslash S\}.$$

The size of a cut is the number of edges in it, and the MAXCUT problem is the problem of finding a cut of maximal size.

Example 20.1. Consider the graph with $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (3, 4), (1, 4)\}$. There



are 7 nontrivial cuts, namely

$$\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}$$

Note that we don't list all the non-empty subsets of $\{1, 2, 3, 4\}$, as that would amount to counting every cut twice (every set would feature as S and as $V\backslash S$). The sizes of these cuts (number of edges joining S with $V\backslash S$) is

$$2, 1, 1, 2, 1, 3, 2.$$

The maximum cut size is therefore 3 (it can't be any bigger, since we only have three edges), and one such cut can be realized by taking the set $S = \{1, 3\}$ and $V\backslash S = \{2, 4\}$.

The problem MAXCUT (like many other graph problems) has the property that it is NP-hard. This means that the decision version of this problem (given a cut, does it have size at least k?) is NP-complete, which in turns means that checking the statement is easy (just count the number of edges in the cut), but *finding* a cut of size at least k appears to be difficult: there does not seem to be a way of doing this that is fundamentally faster than to test all the possible cuts. A famous conjecture, P≠NP, implies that there does not exist an efficient (polynomial time) algorithm that would, in general, be able to find a cut of a given size, or even a maximal cut.

Two common approaches to deal with difficult problems are *approximation* and *randomization*. In approximation, one gives up on finding the best solution and concentrates on finding one that is good enough. In randomization, one takes into account that an algorithm may fail with small probability. Both approximation and randomization may lead to efficient algorithms that can solve a problem well enough for "all practical purposes".

In the case of MAXCUT, an approximation algorithm $\mathcal{A}$ takes as input a graph G and outputs a set $S \subseteq V$, written as $\mathcal{A}(G) = S$. If by $\omega(S)$ we denote the size of the cut induced by S, and by $\mathrm{Opt}(G)$ the largest size of a cut, then the algorithm $\mathcal{A}$ is said to provide an δ-approximation (for some $\delta > 0$), if

$$\omega(\mathcal{A}(G)) \geq \delta \cdot \mathrm{Opt}(G).$$

In words, the algorithms provides a cut that is optimal up to a factor δ. A randomized algorithm is one that is allowed to use internal coin flips. Such an algorithm is a δ-approximation algorithm, if the expected value

$$\mathrm{E}[\omega(\mathcal{A}(G))] \geq \delta \cdot \mathrm{Opt}(G).$$

We are interested in approximation algorithms that run in *polynomial time*. This means that the number of computational steps is a polynomial in the input size. Interior point methods for convex optimization and semidefinite programming are examples of polynomial time algorithms, and it is a remarkable fact that *MaxCut* can be solved approximatively using semidefinite programming.

20.2 The Goemans-Williamson Algorithm

The MAXCUT problem can be formulated as an integer programming problem as follows. Introduce variables $x_1, \ldots, x_n$, where n is the number of vertices. A cut is then an assignment $x_i = 1$ (meaning that $i \in S$) or $x_i = -1$ (meaning that $i \in V \backslash S$). If $e = (i, j)$ is an edge in the cut, then $x_i x_j = -1$, and $x_i x_j = 1$ otherwise. We therefore have

$$\frac{1}{2}(1 - x_i x_j) = \begin{cases} 1 & \text{if } (i,j) \in E(S, V\backslash S) \\ 0 & \text{if } (i,j) \notin E(S, V\backslash S). \end{cases}$$

The size of a cut is then the sum $\sum_{(i,j)\in E} \frac{1}{2}(1 - x_i x_j)$, and the problem of finding the maximum cut can be written as

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - x_i x_j) \\ \text{subject to} \quad & x_i^2 = 1,\ 1 \le i \le n. \end{aligned} \tag{20.1}$$

We now look for a *relaxation* of the problem: a new problem with a bigger constraint set that also contains the solutions of (20.1), but possibly more. As a first step, replace $x_i \in \{-1, 1\}$ with vectors $\boldsymbol{u}_i \in \{\pm \boldsymbol{e}_1\}$, where $\boldsymbol{e}_1 = (1, 0, \ldots, 0)^\top$ is a unit vector, and $x_i x_j$ with $\boldsymbol{u}_i^\top \boldsymbol{u}_j$. The optimization is then over sets of vectors $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$, and can be written as

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - \boldsymbol{u}_i^\top \boldsymbol{u}_j) \\ \text{subject to} \quad & \boldsymbol{u}_i \in \{\pm \boldsymbol{e}_1\}. \end{aligned}$$

The relaxation now consists in enlarging the constraint set to allow for *any* unit vectors $\boldsymbol{u}_i \in S^{n-1}$, where $S^{n-1} = \{\boldsymbol{x} : \|\boldsymbol{x}\| = 1\}$ is the unit sphere.

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - \boldsymbol{u}_i^\top \boldsymbol{u}_j) \\ \text{subject to} \quad & \|\boldsymbol{u}_i\| = 1. \end{aligned} \tag{20.2}$$

As the constraint set of the relaxation is bigger, the solution will be at least as large as the solution of (20.1) (ideally, the same, but we can't guarantee this).

For any set of unit vectors $\boldsymbol{u}_i$, let $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$ be the matrix with unit vectors $\boldsymbol{u}_i$ as columns, and consider

$$\boldsymbol{X} = \boldsymbol{U}^\top \boldsymbol{U}.$$

Then $x_{ii} = \boldsymbol{u}_i^\top \boldsymbol{u}_i = 1$ and the matrix $\boldsymbol{X}$ is symmetric and positive definite. Conversely, any symmetric positive definite matrix $\boldsymbol{X}$ can be written as $\boldsymbol{X} = \boldsymbol{U}^\top \boldsymbol{U}$ (Cholesky factorization), and if in addition $x_{ii} = 1$, then the columns of $\boldsymbol{U}$ necessarily have unit length. We conclude that Problem (20.2) is equivalent to the following SDP:

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - x_{ij}) \\ \text{subject to} \quad & x_{ii} = 1 \\ & \boldsymbol{X} \succeq \boldsymbol{0}. \end{aligned} \tag{20.3}$$

Write $\mathrm{SDP}(G)$ for the optimal value of (20.3). From the above discussion we have

$$\mathrm{SDP}(G) \geq \mathrm{Opt}(G).$$

This is the case, because the solution of (20.1) is contained in the feasible set of (20.3), but the latter has more options.

To recover the cut, we proceed as follows.

1. Solve (20.3) to accuracy ε to obtain a matrix $\boldsymbol{X}^*$ such that

$$\sum_{(i,j)\in E} \frac{1}{2}(1 - x_{ij}^*) \geq \mathrm{SDP}(G) - \varepsilon.$$

   This can be done in polynomial time using, for example, interior point methods.

2. Perform a Cholesky factorization $\boldsymbol{X}^* = (\boldsymbol{U}^*)^\top \boldsymbol{U}^*$ and extract the columns $\boldsymbol{u}_1^*, \ldots, \boldsymbol{u}_n^*$, which are unit vectors and satisfy

$$\sum_{(i,j)\in E} \frac{1}{2}(1 - (\boldsymbol{u}_i^*)^\top \boldsymbol{u}_j^*) \geq \mathrm{Opt}(G) - \varepsilon.$$

3. Finally, we want a way to recover a partition of the graph without loosing too much. We thus need to assign to each $\boldsymbol{u}_i$ a value $x_i \in \{-1, 1\}$ according so some rule, and declare the result to be our partition. To do so, we *randomly* choose a vector $\boldsymbol{p} \in S^{n-1}$ and then set

$$x_i = \begin{cases} 1 & \text{if } \boldsymbol{u}_i^\top \boldsymbol{p} \geq 0, \\ -1 & \text{else.} \end{cases}$$

There we have our algorithm: it generates a cut. Let's call the above algorithm, which takes a graph G and produces a set S by the above steps, $\boldsymbol{A}$.

Theorem 20.2. *The Goemans-Williamson algorithms generates a cut $\mathcal{A}(G) = S$ and satisfies*

$$\mathrm{E}[\omega(\mathcal{A}(G))] \geq 0.8785672 \cdot \mathrm{Opt}(G).$$

That is, the algorithm is expected to generate a cut that is at least as 0.8785672 times as good as the optimal cut (that is, could be about 10 percent smaller but not more). By the law of large numbers, repeating the algorithm will likely give a result close to the expectation.

Appendix

This appendix contain a summary of background material from linear algebra and calculus. Much of the content should be familiar to some degree, and the purpose is to bring it back to attention. Important concepts are **highlighted** in the notes.

1 Asymptotic notation

An **algorithm** is a sequence of instructions carried out by a computer. Important features of an algorithm are computation time (measured as the number of operations or the number of iterations needed to reach a solution) and accuracy. One is mainly interested in the **orders of magnitude** of these quantities, and not so much in their exact values. A convenient notation for this purpose is the asymptotic O-notation.

Let $f, g\colon \mathbb{R} \to \mathbb{R}$ be two functions. Then:

- $f(n) \in O(g(n))$ as $n \to \infty$ if there exists a constant $C > 0$ and an integer n_0 such that $|f(n)| \leq C|g(n)|$ for $n > n_0$;
- $f(x) \in O(g(x))$ as $x \to 0$ if there exists a constant $C > 0$ and a real number $\varepsilon > 0$ such that $|f(x)| \leq C|g(x)|$ for $|x| < \varepsilon$.

We omit "$n \to \infty$" or "$x \to 0$" when it is clear from the context. One often finds statements such as $f(n) = O(g(n))$ or $f(x) = 1 + x + O(x^2)$; the first is equivalent to $f(n) \in O(g(n))$, while the second should be read as $f(x) = 1 + x + g(x)$ for a function $g(x) \in O(x^2)$. The following examples illustrate the O-notation.

- $\sqrt{n} + n^2 \in O(n^2)$ as $n \to \infty$
- $n^5 \in O(e^n)$ as $n \to \infty$
- $10^{100} \in O(1)$ as $n \to \infty$
- $x^3 \in O(x^2)$ as $x \to 0$
- $\sin(x) \in O(x)$ as $x \to 0$
- $e^x = 1 + x + O(x^2)$ as $x \to 0$

The notation $f(n) \in \Omega(g(n))$ as $n \to \infty$ means that $g(n) \in O(f(n))$ as $n \to \infty$, and $f(n) \in o(g(n))$ as $n \to \infty$ means that for *all* $C > 0$ there exists n_0 such that $|f(n)| < C|g(n)|$ for $n > n_0$. If $g(n) \neq 0$ for sufficiently large n, this is equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. One defines $\Omega(g(x))$ and $o(g(x))$ as $x \to 0$ analogously.

2 Linear Algebra

We restrict to linear algebra over the field of real numbers $\mathbb{R}$, as this is the setting that is of most interest in optimization. A **vector** in $\mathbb{R}^n$ and its **transpose** are written as

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = (x_1, \ldots, x_n)^\top, \quad \boldsymbol{x}^\top = (x_1, \ldots, x_n),$$

with **coordinates** $x_i \in \mathbb{R}$ for $1 \leq i \leq n$. The zero vector is denoted by $\mathbf{0}$, while $\boldsymbol{e}$ is the vector with every coordinate equal to 1. If $\lambda_1, \lambda_2 \in \mathbb{R}$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, then $\lambda_1 \boldsymbol{x} + \lambda_2 \boldsymbol{y}$ is the vector with coordinates $\lambda_1 x_i + \lambda_2 y_i$ for $1 \leq i \leq n$.

In $\mathbb{R}^n$ we have the Euclidean (or standard) **inner product** (or scalar product)

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

The Euclidean inner product is **bilinear**: for $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$,

$$\langle \alpha \boldsymbol{x}_1 + \beta \boldsymbol{x}_2, \boldsymbol{y} \rangle = \alpha \langle \boldsymbol{x}_1, \boldsymbol{y} \rangle + \beta \langle \boldsymbol{x}_2, \boldsymbol{y} \rangle, \quad \langle \boldsymbol{x}, \alpha \boldsymbol{y}_1 + \beta \boldsymbol{y}_2 \rangle = \alpha \langle \boldsymbol{x}, \boldsymbol{y}_1 \rangle + \beta \langle \boldsymbol{x}, \boldsymbol{y}_2 \rangle,$$

symmetric ($\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle$) and satisfies $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$, with equality if and only if $\boldsymbol{x} = \mathbf{0}$. Two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are called **orthogonal**, if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$.

Example 1.3. The vectors $(1, 1)^\top$ and $(1, -1)^\top$ are orthogonal in $\mathbb{R}^2$, while $(1, 1)^\top$ and $(2, -1)^\top$ are not.

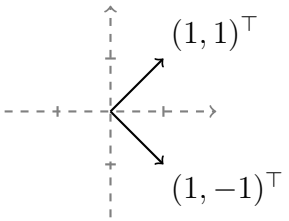


Figure 2.25: Orthogonal vectors

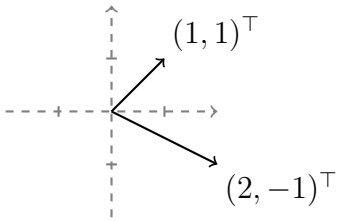


Figure 2.26: Non-orthogonal vectors

Linear subspaces

A **linear subspace** is a subset $V \subseteq \mathbb{R}^n$ such that for any $\boldsymbol{x}, \boldsymbol{y} \in V$ and for all $\alpha, \beta \in \mathbb{R}$, $\alpha \boldsymbol{x} + \beta \boldsymbol{y} \in V$. In particular, the sets $\{\mathbf{0}\}$ and $\mathbb{R}^n$ are linear subspaces.

Example 1.4. The linear subspaces of $\mathbb{R}^2$ are $\{\mathbf{0}\}$, lines through the origin, and $\mathbb{R}^2$. The linear subspaces of $\mathbb{R}^3$ are $\{\mathbf{0}\}$, lines and planes through the origin, and $\mathbb{R}^3$.

A **linear combination** of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^n$ is an expression of the form $\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i$, where $\lambda_i \in \mathbb{R}$ for $1 \leq i \leq k$. The set of linear combinations

$$V = \operatorname{span} \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} := \left\{ \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i : \lambda_i \in \mathbb{R} \right\}$$

forms a linear subspace of $\mathbb{R}^n$. It is the intersection of all linear subspaces that contain $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$. The vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ are **linearly independent** if $\sum_{i=1}^k \lambda_i \boldsymbol{x}_i = 0$ implies $\lambda_1 = \cdots = \lambda_k = 0$. A minimal set of vectors that span a linear subspace V is called a **basis** of this subspace, and the number of elements in a basis is the **dimension** of the linear subspace. The elements of a basis are always linearly independent, and a maximal linearly independent set in a vector subspace V is a basis. If $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ is a basis of a subspace V, then every $\boldsymbol{x} \in V$ has a *unique* representation $\boldsymbol{x} = \sum_{i=1}^k \lambda_i \boldsymbol{b}_i$. A basis is **orthogonal** if $\langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle = 0$ for $i \neq j$, and **orthonormal** if in addition $\langle \boldsymbol{b}_i, \boldsymbol{b}_i \rangle = 1$ for $1 \leq i \leq k$. The unique expression of $\boldsymbol{x} \in V$ as linear combination of an orthonormal basis $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ of V is given by

$$\boldsymbol{x} = \sum_{i=1}^{k} \langle \boldsymbol{x}, \boldsymbol{b}_i \rangle \boldsymbol{b}_i.$$

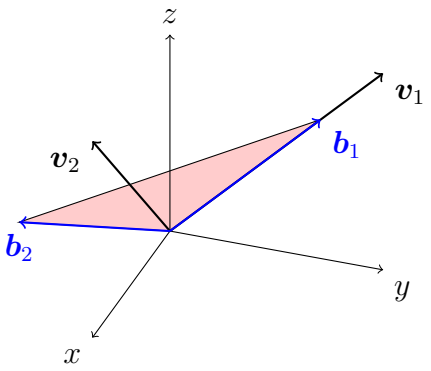
The **standard basis** of $\mathbb{R}^n$ is the orthonormal basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$, where $\boldsymbol{e}_i$ has a 1 in the i-th coordinate and 0 elsewhere.

Example 1.5. The vectors $\boldsymbol{v}_1 = (0, 1, 1)^\top$ and $\boldsymbol{v}_2 = (1, 0, 1)^\top$ span a linear subspace of $\mathbb{R}^3$ of dimension 2, but they are not orthogonal. The vectors

$$\boldsymbol{b}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \boldsymbol{b}_2 = \frac{1}{\sqrt{3}} \begin{pmatrix} \sqrt{2} \\ -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

form an orthonormal basis of V. The vector $\boldsymbol{x} = (1, 1, 2)^\top$ lives in V, and its representation in terms of $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ is

$$\boldsymbol{x} = \frac{3}{\sqrt{2}} \boldsymbol{b}_1 + \sqrt{\frac{3}{2}} \boldsymbol{b}_2.$$



The **direct sum** of two vector subspaces $V, W \subset \mathbb{R}^n$ with $V \cap W = \{\boldsymbol{0}\}$ is

$$V \oplus W = \{\boldsymbol{v} + \boldsymbol{w} : \boldsymbol{v} \in V, \boldsymbol{w} \in W\}.$$

The **orthogonal complement** of a subspace $V \subseteq \mathbb{R}^n$ is the set

$$V^\perp = \{\boldsymbol{x} \in \mathbb{R}^n : \forall \boldsymbol{y} \in V : \langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0\}.$$

The vector space $\mathbb{R}^n$ is the direct sum of V and its orthogonal complement,

$$\mathbb{R}^n = V \oplus V^\perp. \tag{1}$$

If $V = \text{span}\,\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$, then $V^\perp = \{\boldsymbol{y} : \langle \boldsymbol{x}_1, \boldsymbol{y}\rangle = \cdots \langle \boldsymbol{x}_k, \boldsymbol{y}\rangle = 0\}$; to check whether a vector $\boldsymbol{y}$ is in the orthogonal complement of V we therefore only need to check whether $\boldsymbol{y}$ is orthogonal to a spanning set (for example, a basis) of V.

Example 1.6. The vector $\boldsymbol{x} = (-1, -1, 1)^\top$ is orthogonal to the basis $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ from Example 1.5. It is therefore orthogonal to the whole plane $V = \text{span}\,\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ spanned by these vectors. The orthogonal complement of V is the line $\{\lambda \boldsymbol{x} : \lambda \in \mathbb{R}\}$.

The **direct product** of two vector subspaces $V \subseteq \mathbb{R}^n$, $W \subseteq \mathbb{R}^m$ is defined as

$$V \times W = \{(\boldsymbol{v}, \boldsymbol{w}) \in \mathbb{R}^{n+m} : \boldsymbol{v} \in V, \boldsymbol{w} \in W\}.$$

where $(\boldsymbol{v}, \boldsymbol{w})$ is the vector whose first n coordinates coincide with $\boldsymbol{v}$, and the last m coordinates coincide with $\boldsymbol{w}$. In particular, $\mathbb{R}^n \times \mathbb{R}^m = \mathbb{R}^{n+m}$.

Linear maps

An $m \times n$ **matrix**

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix},$$

represents a **linear map** from $\mathbb{R}^n$ to $\mathbb{R}^m$ by means of

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}, \quad y_i = \sum_{j=1}^{n} a_{ij} x_j.$$

For example,

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}.$$

The columns of a matrix are vectors, and we sometimes write

$$\boldsymbol{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$$

for the matrix whose columns are given by the vectors $\boldsymbol{a}_i$. If $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and $n = m + k$, then $\boldsymbol{A}$ can be written as **block matrix**,

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix},$$

with $\boldsymbol{A}_{11} \in \mathbb{R}^{m \times m}$, $\boldsymbol{A}_{22} \in \mathbb{R}^{k \times k}$, $\boldsymbol{A}_{12} \in \mathbb{R}^{m \times k}$ and $\boldsymbol{A}_{21} \in \mathbb{R}^{k \times m}$. The sum and difference of matrices of the same size are defined component-wise.

The $n \times n$ matrix $\mathbf{1}$ is the matrix with 1 on the diagonal and 0 elsewhere, while $\mathbf{0}$ is the matrix consisting of only zeros. A matrix is **diagonal** if all the off-diagonal elements are 0,

lower-triangular if all the elements above the diagonal are 0, and **upper-triangular** if all the elements below the diagonal are 0. A **block-diagonal** matrix is a block matrix, with all blocks outside the main diagonal consisting of zero-matrices $\mathbf{0}$.

The **transpose** $\boldsymbol{A}^\top$ is the matrix with entries $a'_{ij} := a_{ji}$. It is the matrix $\boldsymbol{A}$ mirrored on the diagonal from top left to bottom right. A matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is called **symmetric** if $\boldsymbol{A}^\top = \boldsymbol{A}$. The set of symmetric matrices in $\mathbb{R}^{n\times n}$ is denoted by $\mathcal{S}^n$.

The **product** of an $m \times p$ matrix $\boldsymbol{A}$ with a $p \times n$ matrix $\boldsymbol{B}$,

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B},$$

is the $m \times n$ matrix $\boldsymbol{C}$ whose (i, j)-th entry is given by

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

It represents a composition of maps $\mathbb{R}^n \to \mathbb{R}^p \to \mathbb{R}^m$. The number of columns of $\boldsymbol{A}$ has to equal the number of rows of $\boldsymbol{B}$ for this definition to make sense. Products of block matrices or of block matrices with vectors can be carried out block-wise. If, for example, $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)^\top$ with $\boldsymbol{x}_1 \in \mathbb{R}^{1\times m}$ and $\boldsymbol{x}_2 \in \mathbb{R}^{1\times k}$, then

$$\begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{11}\boldsymbol{x}_1 + \boldsymbol{A}_{12}\boldsymbol{x}_2 \\ \boldsymbol{A}_{21}\boldsymbol{x}_1 + \boldsymbol{A}_{22}\boldsymbol{x}_2 \end{pmatrix}.$$

The matrix $\mathbf{1}$ satisfies $\mathbf{1}\boldsymbol{A} = \boldsymbol{A}$ and $\boldsymbol{A}\mathbf{1} = \boldsymbol{A}$, whenever the dimensions are such that this is defined. In general, even if $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$, $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$.

Example 1.7. Let

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 1 & 4 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}.$$

Then

$$\boldsymbol{A}\boldsymbol{B} = \begin{pmatrix} 8 & 7 \\ 14 & 11 \end{pmatrix}, \quad \boldsymbol{B}\boldsymbol{A} = \begin{pmatrix} 5 & 16 \\ 5 & 14 \end{pmatrix}.$$

If we consider a vector $\boldsymbol{x} \in \mathbb{R}^n$ as an $n \times 1$ matrix and the transpose as an $1 \times n$ matrix, then for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \cong \mathbb{R}^{n\times 1}$ we have

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\top \boldsymbol{y}.$$

The transpose of a product satisfies $(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$. From this it follows that for any matrix, $\boldsymbol{A}^\top \boldsymbol{A}$ is symmetric. For any matrix $\boldsymbol{A}$ we have

$$\langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{x} \rangle = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}^\top \boldsymbol{x})^\top \boldsymbol{x} = \langle \boldsymbol{A}^\top \boldsymbol{x}, \boldsymbol{x} \rangle.$$

It follows from this that if a matrix is symmetric, then it is also self-adjoint, which means that $\langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{x} \rangle = \langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{x} \rangle$.

The **rank** of a matrix $\boldsymbol{A}$, $\mathrm{rk}(\boldsymbol{A})$, is the maximum number of linearly independent rows or columns of $\boldsymbol{A}$. The **kernel** and **image** of $\boldsymbol{A}$ are the linear subspaces

$$\ker \boldsymbol{A} := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = 0\}, \quad \operatorname{im} \boldsymbol{A} = \{\boldsymbol{A}\boldsymbol{x} : \boldsymbol{x} \in \mathbb{R}^n\}.$$

The dimensions are given by $\dim \ker \boldsymbol{A} = n - \mathrm{rk}(\boldsymbol{A})$ and $\dim \mathrm{im}\, \boldsymbol{A} = \mathrm{rk}(\boldsymbol{A})$. While $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ represents a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$, the transpose $\boldsymbol{A}^\top$ represents a map in the other direction, and the image of $\boldsymbol{A}^\top$ coincides with the orthogonal complement of the kernel of $\boldsymbol{A}$, $(\ker \boldsymbol{A})^\perp = \mathrm{im}\, \boldsymbol{A}^\top$. In particular, in view of (1) we have the direct sum decomposition

$$\mathbb{R}^n = \ker \boldsymbol{A} \oplus \mathrm{im}\, \boldsymbol{A}^\top.$$

A **system of linear equations**

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\ \vdots \qquad\qquad \vdots \qquad &\vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

is written as a matrix vector product

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2}$$

where the $m \times n$ matrix $\boldsymbol{A}$ is defined as above, and $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$. If the columns of $\boldsymbol{A}$ are linearly independent, then the system of equations can have at most one solution, and otherwise it has infinitely many solutions (this is the case if $n > m$). If $n = m$, then the system (2) has a unique solution if and only if the matrix $\boldsymbol{A}$ is **invertible** or **non-singular**. This is the case if the rows of $\boldsymbol{A}$ (or equivalently, the columns of $\boldsymbol{A}$) are linearly independent. If $\boldsymbol{A}$ is not invertible, it is called **singular**.

If $\boldsymbol{A}$ is invertible, there exists a matrix $\boldsymbol{A}^{-1}$ (the **inverse**) such that

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \mathbf{1}.$$

The solution of (2) is then given by $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$. The following conditions on a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ are equivalent:

1. $\boldsymbol{A}$ is invertible,
2. $\mathrm{rk}(\boldsymbol{A}) = n$,
3. $\ker \boldsymbol{A} = \{\mathbf{0}\}$,
4. $\mathrm{im}\, \boldsymbol{A} = \mathbb{R}^n$,
5. the rows of $\boldsymbol{A}$ are linearly independent,
6. the columns of $\boldsymbol{A}$ are linearly independent,
7. $\det(\boldsymbol{A}) \neq 0$,

where the determinant is

$$\det(\boldsymbol{A}) = \sum_{\sigma\in S_n} \mathrm{sgn}(\sigma) a_{1\sigma(1)} \cdots a_{n\sigma(n)},$$

and S_n is the group of permutations of $[n] = \{1, \ldots, n\}$, with $\mathrm{sgn}(\sigma)$ the sign of the permutation (parity of the number of inversions).

Example 1.8. For two- and three-dimensional matrices

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

the determinants are

$$\begin{aligned} \det(\boldsymbol{A}) &= a_{11}a_{22} - a_{12}a_{21}, \\ \det(\boldsymbol{B}) &= a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31}). \end{aligned}$$

A matrix $\boldsymbol{Q}$ is **orthogonal** if $\boldsymbol{Q} = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n)$, with $\langle \boldsymbol{q}_i, \boldsymbol{q}_j \rangle = \delta_{ij}$, and

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$$

As the (i, j)-th entry of $\boldsymbol{Q}^\top \boldsymbol{Q}$ are given by $\langle \boldsymbol{q}_i, \boldsymbol{q}_j \rangle$, the orthogonality of $\boldsymbol{Q}$ can succinctly be characterized by the requirement $\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{1}$. In particular, $\boldsymbol{Q}^\top = \boldsymbol{Q}^{-1}$, and the columns (and rows) of an orthogonal matrix form an orthonormal basis of $\mathbb{R}^n$. Orthogonal matrices have the property that $\langle \boldsymbol{Q}\boldsymbol{x}, \boldsymbol{Q}\boldsymbol{y} \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$. From this it follows that orthogonality of vectors is preserved under orthogonal transformations. The determinant of an orthogonal matrix is $\det(\boldsymbol{Q}) = 1$. As the product of orthogonal matrices is again orthogonal, the set of orthogonal $n \times n$ matrices forms a group, commonly denoted by $O(n)$.

Example 1.9. Consider the three matrices,

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}.$$

The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are symmetric, while $\boldsymbol{C}$ is not. The matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ are invertible, with inverse

$$\boldsymbol{B}^{-1} = \begin{pmatrix} -0.4 & 0.6 \\ 0.6 & -0.4 \end{pmatrix}, \quad \boldsymbol{C}^{-1} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}.$$

The matrix $\boldsymbol{A}$ is not invertible, since the second column is a multiple of the first. The kernel of $\boldsymbol{A}$ the linear span of $(-2, 1)^\top$. The matrix $\boldsymbol{C}$ is orthogonal (this can be seen by checking that the columns or rows are orthonormal, or looking at the expression of the inverse above).

Eigenvalues

A vector $\boldsymbol{u} \neq \boldsymbol{0}$ is an **eigenvector** of $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, if there exists a $\lambda \in \mathbb{C}$ such that

$$\boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{u}.$$

Such a number λ is called an **eigenvalue** of $\boldsymbol{A}$. Note that the eigenvectors are only defined up to scaling: if $\boldsymbol{u}$ is an eigenvector, then so is $\lambda \boldsymbol{u}$ for any non-zero $\lambda \in \mathbb{R}$.

From the definition of the determinant, the function $\lambda \mapsto \det(\lambda \mathbf{1} - \boldsymbol{A})$ is a polynomial of degree at most n, called the **characteristic polynomial** of $\boldsymbol{A}$. The eigenvalues are the roots of this polynomial,

$$\det(\lambda \mathbf{1} - \boldsymbol{A}) = 0.$$

The eigenvalues can be complex numbers, and appear in complex conjugate pairs. If the matrix $\boldsymbol{A}$ is symmetric, then the eigenvalues are all real numbers. Two important quantities, the **determinant** and the **trace** of a matrix (corresponding, up to sign, to the highest and lowest coefficient of the characteristic polynomial) can be expressed in terms of the eigenvalues:

$$\det(\boldsymbol{A}) = \lambda_1 \cdots \lambda_n, \quad \operatorname{trace}(\boldsymbol{A}) := a_{11} + \cdots + a_{nn} = \lambda_1 + \cdots + \lambda_n.$$

A matrix has a zero eigenvalue if and only if it is singular. Eigenvalues may occur with multiplicity.

Norms

A **norm** in $\mathbb{R}^n$ is a function $\|\cdot\|$ that satisfies the following three properties

1. $\|\boldsymbol{x}\| \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{x} = 0$ if and only if $\boldsymbol{x} = \mathbf{0}$;
2. $\|\lambda \boldsymbol{x}\| = |\lambda| \|\boldsymbol{x}\|$ for $\lambda \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^n$;
3. $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Three important examples of norms are the following:

1. The 1-norm: $\|\boldsymbol{x}\|_1 = \sum_{i=1}^n |x_i|$;
2. The 2-norm: $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$;
3. The ∞-norm: $\|\boldsymbol{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$.

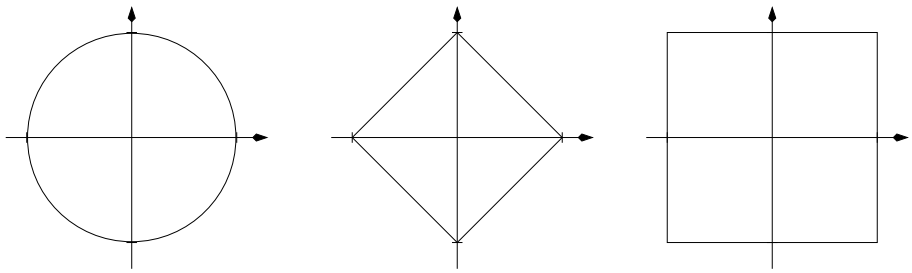
Example 1.10. Let $\boldsymbol{x} = (2, -3, 4)^\top$. The $\|\boldsymbol{x}\|_1 = 9$, $\|\boldsymbol{x}\|_2 = \sqrt{29}$, and $\|\boldsymbol{x}\|_\infty = 4$.

Note that the 2-norm, also called **Euclidean norm**, can be defined as

$$\|\boldsymbol{x}\|_2^2 = \boldsymbol{x}^\top \boldsymbol{x} = \langle \boldsymbol{x}, \boldsymbol{x} \rangle,$$

that is, it is the norm induced by the Euclidean inner product. From this it follows that the 2-norm does not change under orthogonal transformations: if $\boldsymbol{Q} \in O(n)$ and $\boldsymbol{x} \in \mathbb{R}^n$, then $\|\boldsymbol{Q}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$. Orthogonal transformations in $\mathbb{R}^2$ and $\mathbb{R}^3$ correspond to rotations and reflections, so it is intuitively clear that these don't change distances.

The **unit sphere** with respect to a norm is the set $\{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| = 1\}$, and the (closed) **unit ball** is the set $\{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| \leq 1\}$. The unit spheres with respect to the 2-norm, the 1-norm and the ∞-norm in $\mathbb{R}^2$ are shown in the following diagram.



The unit sphere with respect to the 2-norm in $\mathbb{R}^n$ is usually denoted by S^{n-1}.

The 1-, 2- and ∞-norms are equivalent, in the sense that they can be bounded in terms of each other. In particular,

$$\|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty, \quad \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_1 \leq n\|\boldsymbol{x}\|_\infty. \tag{3}$$

The inner product and the 2-norm are related the **Cauchy-Schwarz** inequality,

$$|\langle \boldsymbol{x}, \boldsymbol{y}\rangle| \leq \|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2,$$

with equality if and only if $\boldsymbol{x}$ and $\boldsymbol{y}$ are linearly dependent. As a consequence of the Cauchy-Schwarz inequality we get

$$-1 \leq \frac{\langle \boldsymbol{x}, \boldsymbol{y}\rangle}{\|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2} \leq 1.$$

The **angle** between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is the number $\theta \in [0, 2\pi)$ such that

$$\cos(\theta) = \frac{\langle \boldsymbol{x}, \boldsymbol{y}\rangle}{\|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2}.$$

If $\boldsymbol{x}$ and $\boldsymbol{y}$ are orthogonal, then $\cos(\theta) = 0$ and $\theta \in \{\pi/2, 3\pi/2\}$.

Norms are an important device to measure the size of vectors. In order to measure the amount by which a linear transformation (matrix) distorts vectors, we need the concept of **matrix norms**. A matrix norm is a function on the set of matrices that is a norm when considering a matrix as a vector, and in addition satisfies the condition

$$\|\boldsymbol{AB}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|.$$

The most important examples are given by the **operator norms**. Given a vector norm $\|\boldsymbol{x}\|$, the associated matrix norm is defined as

$$\|\boldsymbol{A}\| = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{Ax}\|}{\|\boldsymbol{x}\|} = \max_{\boldsymbol{x}:\ \|\boldsymbol{x}\|\leq 1} \|\boldsymbol{Ax}\|.$$

The matrix norms $\|\boldsymbol{A}\|_1, \|\boldsymbol{A}\|_2, \|\boldsymbol{A}\|_\infty$ are the operator norms that arise when using the 1-, 2- and ∞-norms. They can conveniently characterized as follows

- $\|\boldsymbol{A}\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$;
- $\|\boldsymbol{A}\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$;

- $\|\boldsymbol{A}\|_2 = \sqrt{\lambda_{\max}(\boldsymbol{A}^\top \boldsymbol{A})}$.

Here, $\lambda_{\max}$ denotes the largest eigenvalue of the symmetric matrix $\boldsymbol{A}^\top \boldsymbol{A}$. If $\boldsymbol{A}$ is symmetric, then $\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{A}^2$, and the eigenvalues of $\boldsymbol{A}^2$ are the squares of the eigenvalues of $\boldsymbol{A}$. It follows that for symmetric $\boldsymbol{A}$, $\|\boldsymbol{A}\|_2 = \lambda_{\max}(\boldsymbol{A})$.

Example 1.11. Let

$$\boldsymbol{A} = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}.$$

Then $\|\boldsymbol{A}\|_1 = \|\boldsymbol{A}\|_\infty = 3$ and $\|\boldsymbol{A}\|_2 = 3$.

A special case is the **dual norm** of a vector norm: to a given norm is defined as

$$\|\boldsymbol{x}\|^* = \max_{\boldsymbol{y}\colon \|\boldsymbol{y}\| \le 1} \langle \boldsymbol{x}, \boldsymbol{y} \rangle.$$

This is the operator norm of $\boldsymbol{x}^\top$, considered as a $1 \times d$ matrix. The dual of the dual norm is the norm itself. The dual norm of the 2-norm is again the 2-norm, while the 1-norm and the ∞-norm are dual to each other.

In addition to the operator norms, an important matrix norm is the **Frobenius norm** of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$,

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j=1}^{n} a_{ij}^2}.$$

This is just the 2-norm of $\boldsymbol{A}$ interpreted as a vector in $\mathbb{R}^{n^2}$. The 2-norm and the Frobenius norm have the important property of being **orthogonal invariant**, which means that for any $\boldsymbol{Q} \in O(n)$,

$$\|\boldsymbol{QA}\|_2 = \|\boldsymbol{AQ}\|_2 = \|\boldsymbol{A}\|_2, \quad \|\boldsymbol{QA}\|_F = \|\boldsymbol{AQ}\|_F = \|\boldsymbol{A}\|_F.$$

Orthogonal invariance allows to simplify a matrix without changing the norm.

Positive semidefinite matrices

If $\boldsymbol{A} \in \mathcal{S}^n$ is a symmetric matrix and $\boldsymbol{u} \in \mathbb{R}^n$ an eigenvector with $\|u\|_2 = 1$ and corresponding eigenvalue λ, then $\boldsymbol{u}^\top \boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{u}^\top \boldsymbol{u} = \lambda$. In particular, the largest and smallest values of an eigenvalue are given by

$$\lambda_1 = \max_{\boldsymbol{u}\colon \|\boldsymbol{u}\|_2 = 1} \boldsymbol{u}^\top \boldsymbol{A}\boldsymbol{u}, \quad \lambda_n = \min_{\boldsymbol{u}\colon \|\boldsymbol{u}\|_2 = 1} \boldsymbol{u}^\top \boldsymbol{A}\boldsymbol{u}.$$

A symmetric matrix $\boldsymbol{A}$ is called **positive semidefinite**, written $\boldsymbol{A} \succeq \boldsymbol{0}$, if for all non-zero $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \ge 0$, and **positive definite**, written $\boldsymbol{A} \succ \boldsymbol{0}$, if $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$. Equivalently, a symmetric matrix is positive semidefinite if all its eigenvalues are non-negative, and positive definite if they are all positive. The set of positive semidefinite symmetric matrices in $\mathbb{R}^{n \times n}$ is denoted by $\mathcal{S}^n_+$, while the set of positive definite matrices is $\mathcal{S}^n_{++}$.

An **inner product** (or scalar product) on $\mathbb{R}^{n \times n}$ is a function

$$\langle \cdot, \cdot \rangle \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, \quad (\boldsymbol{x}, \boldsymbol{y}) \mapsto \langle \boldsymbol{x}, \boldsymbol{y} \rangle$$

that is bilinear (linear in each of the two arguments), symmetric ($\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle$), and satisfies $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$, with $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$. The standard inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\top \boldsymbol{y}$ is an example, and the notation $\langle \cdot, \cdot \rangle$ usually refers to this product. More generally, every matrix $\boldsymbol{A} \in \mathcal{S}^n_{++}$ defines an inner product by

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{A}} := \langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{y} \rangle = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y}.$$

The associated norm is $\|\boldsymbol{x}\|_{\boldsymbol{A}} = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle_{\boldsymbol{A}}}$. The unit sphere with respect to this norm,

$$E = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = 1\},$$

is an **ellipsoid**, where the 2-norms of the largest and smallest axes are the largest and smallest eigenvalues of $\boldsymbol{A}^{-1}$.

Matrix decompositions

Matrices can be represented as products of simpler matrices. Important examples are:

1. QR **decomposition**. A matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as

   $$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R},$$

   where $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ is an orthogonal, and $\boldsymbol{R} \in \mathbb{R}^{m \times n}$ an upper triangular matrix. Gram-Schmidt orthogonalisation produces such a decomposition.

2. LU **decomposition**. A square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ can be written as

   $$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

   where $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ is a lower triangular, and $\boldsymbol{U} \in \mathbb{R}^{n \times n}$ an upper triangular matrix. Gaussian eliminations produces such a decomposition.

3. **Symmetric eigenvalue decomposition**. A symmetric matrix $\boldsymbol{A} \in \mathcal{S}^n$ can be written as

   $$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top,$$

   where $\boldsymbol{Q}$ is an orthogonal matrix, with the eigenvectors as rows, and $\boldsymbol{\Lambda}$ a *diagonal* matrix with the eigenvalues on the diagonal.

4. **Cholesky decomposition**. A positive definite symmetric matrix $\boldsymbol{A} \in \mathcal{S}^n_{++}$ can be factored as

   $$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^\top,$$

   with $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ lower-triangular and with strictly positive diagonal entries.

One of the most powerful matrix decompositions is the **singular value decomposition** (SVD). It states that any $m \times n$ matrix $\boldsymbol{A}$ can be written as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top,$$

where $\boldsymbol{U}$ is a $m \times m$ orthogonal matrix, $\boldsymbol{V}$ an $n \times n$ orthogonal matrix, and $\boldsymbol{\Sigma}$ is a diagonal matrix with the **singular values** $\sigma_1 \geq \cdots \geq \sigma_{\min\{m,n\}}$ on the diagonal. The singular values are the square roots of the eigenvalues of $\boldsymbol{A}^\top \boldsymbol{A}$. The singular values are related to the matrix 2-norm and Frobenius norm of $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ as follows:

$$\|\boldsymbol{A}\|_2 = \sigma_1(\boldsymbol{A}), \quad \|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{n} \sigma_i^2(\boldsymbol{A})}.$$

If $\boldsymbol{A}$ is symmetric, the singular values are the absolute values of the eigenvalues of $\boldsymbol{A}$.

Matrix decompositions can help reduce a problem into one involving simpler (orthogonal, triangular) matrices. For example, to solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, one can first compute $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$, solve the simpler system of equations $\boldsymbol{Q}\boldsymbol{y} = \boldsymbol{b}$ by computing $\boldsymbol{y} = \boldsymbol{Q}^\top \boldsymbol{b}$, and then solve the triangular system $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{y}$ by back-substitution.

Example 1.12. Consider the matrix $\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$. The QR decomposition is

$$\boldsymbol{Q} = \frac{1}{\sqrt{5}} \begin{pmatrix} -2 & 1 \\ 1 & 2 \end{pmatrix}, \quad \boldsymbol{R} = \frac{1}{\sqrt{5}} \begin{pmatrix} -5 & 4 \\ 0 & 3 \end{pmatrix},$$

the LU decomposition is

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 \\ -1/2 & 1 \end{pmatrix}, \quad \boldsymbol{U} = \begin{pmatrix} 2 & -1 \\ 0 & 3/2 \end{pmatrix},$$

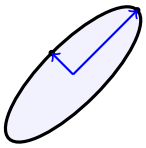
the symmetric eigenvalue decomposition and the SVD are given by

$$\boldsymbol{\Lambda} = \boldsymbol{\Sigma} = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{Q} = \boldsymbol{U} = \boldsymbol{V} = \begin{pmatrix} \cos(\pi/4) & \sin(\pi/4) \\ -\sin(\pi/4) & \cos(\pi/4) \end{pmatrix},$$

and the Cholesky decomposition is given by

$$\boldsymbol{L} = \frac{1}{\sqrt{2}} \begin{pmatrix} 2 & 0 \\ -1 & \sqrt{3} \end{pmatrix}.$$

The eigenvalue decomposition shows how to visualize the ellipse $\{\boldsymbol{A}\boldsymbol{x} : \|\boldsymbol{x}\|_2 = 1\}$:



Applying the transformation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{x}$ corresponds to rotating the vector $\boldsymbol{x}$ clockwise by an angle of $\pi/4$, then stretching by a factor of 3 in the x-direction, and then rotating back.

3 Calculus

We write $C([a,b]) = C^0([a,b])$ for the set of continuous functions on an interval $[a,b]$, and for $k \geq 1$ we write $C^k([a,b])$ for the set of functions continuous on $[a,b]$, and whose first k derivatives $f', \ldots, f^{(k)}$ exist and are continuous on (a,b). In the above definition we allow $a = -\infty$ or $b = \infty$. If $a, b \in \mathbb{R}$, then any function $f \in C([a,b])$ is bounded. The **infimum** (largest lower bound) and **supremum** (smallest upper bound) of a function f on an interval $[a,b]$ are defined as

$$\inf_{x\in[a,b]} f(x) = \max\{y \in \mathbb{R} : \forall x \in [a,b], f(x) \geq y\},$$
$$\sup_{x\in[a,b]} f(x) = \min\{y \in \mathbb{R} : \forall x \in [a,b], f(x) \leq y\}.$$

Again, we allow the "values" $-\infty$ and ∞. If the infimum is attained (i.e., there exists x^* such that $f(x^*) = \inf_{x\in[a,b]} f(x)$), then we write $\min_{x\in[a,b]} f(x)$, and similarly $\max$ if the supremum is attained. Any $f \in C([a,b])$ for $a, b \in \mathbb{R}$ attains its infimum and supremum on $[a,b]$.

Three important concepts are the **Intermediate Value Theorem**, the **Mean Value Theorem**, and the **Taylor expansion**.

Theorem (Intermediate Value Theorem). If $f \in C([a,b])$ and if y satisfies

$$\inf_{x\in[a,b]} f(x) \leq y \leq \sup_{x\in[a,b]} f(x),$$

then there exists $\xi \in [a,b]$ such that $f(\xi) = y$. In particular, the infimum and supremum are attained.

Theorem (Mean Value Theorem). Let $f \in C^1([a,b])$ and let $x, x_0 \in (a,b)$ with $x \neq x_0$. Then there exists a number $\xi \in (x_0, x)$ (or (x, x_0) if $x < x_0$) such that

$$f(x) = f(x_0) + f'(\xi)(x - x_0).$$

This can also be written as

$$f'(\xi) = \frac{f(x) - f(x_0)}{x - x_0}.$$

The Mean Value Theorem is a special case of the Taylor expansion.

Theorem (Taylor expansion). Let $f \in C^{(n)}([a,b])$ and let $x, x_0 \in (a,b)$ with $x \neq x_0$. Then there exists $\xi \in (x, x_0)$ (or (x, x_0) if $x < x_0$) such that

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$$

The first $(n+1)$ terms of the above sum can be seen as an approximation to the function f that becomes more accurate as n increases. The last term is known as the *truncation error* in numerical approximation.

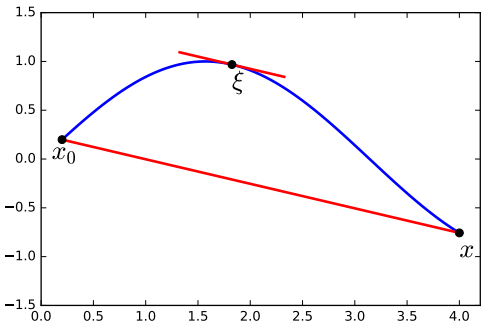


Figure 3.1: MVT: there exists a point at which the slope (derivative) is the same as that of the secant connecting $(x_0, f(x_0))$ and $(x, f(x))$.

As an example, consider the Taylor expansion of the sine function at $x_0 = 0$,

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

The Taylor approximation to different orders is illustrated in the following figure.

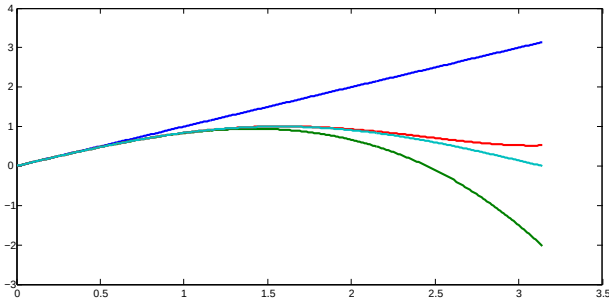


Figure 3.2: Taylor expansion of $\sin(x)$.

A special case is of the Mean Value Theorem is **Rolle's Theorem**.

Theorem (Rolle's Theorem)**.** Let $f \in C^1([a, b])$ with $f(a) = f(b)$. Then there exists a number $\xi \in (a, b)$ such that $f'(\xi) = 0$.

The intuition is that if you walk across mountains and arrive at a point with the same elevation as where you started, then there must be places on the way where the slope is 0, that is, a local maximum or minimum. Of importance is also the following variant of the the Mean Value Theorem.

Theorem (**Integral Mean Value Theorem**)**.** Let $f, g \in C([a, b])$ and assume that $f(x)$ does not change sign on $[a, b]$. Then there exists a $\xi \in (a, b)$ such that

$$\int_a^b f(x)g(x)\,dx = g(\xi)\int_a^b f(x)\,dx.$$

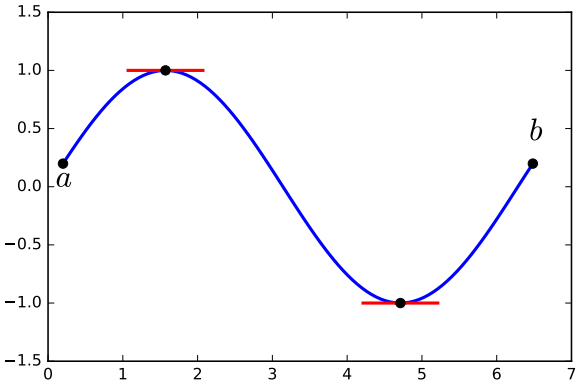


Figure 3.3: Rolle's Theorem: there exist points with "flat" slope (derivative zero).

Topology

The **open ball** of radius ε around $\boldsymbol{p} \in \mathbb{R}^n$ is defined as

$$B^n(\boldsymbol{p}, \varepsilon) = \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{p}\|_2 < \varepsilon\}.$$

We write $B^n := B^n(\mathbf{0}, 1)$ for the (open) **unit ball**. A subset $U \subseteq \mathbb{R}^n$ is called **open** if for every $p \in U$ there exists an $\varepsilon > 0$ such that $B(\boldsymbol{p}, \varepsilon) \subset U$. A set C is **closed** if $\overline{C} = \mathbb{R}^n \backslash C$ is open. The **closure** $\operatorname{cl} S$ of a set $S \subseteq \mathbb{R}^n$ is the intersection of all closed sets containing S, while the **interior** $\operatorname{int} S$ is the union of all open sets contained in S. The **boundary** of S is defined as $\operatorname{bd} S = \operatorname{cl} S \backslash \operatorname{int} S$. For example, the boundary of the open unit ball is the **unit sphere**

$$\operatorname{bd} B^n = S^{n-1} = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\|_2 = 1\}.$$

The superscript $n - 1$ refers to the fact that this set is a manifold of dimension $n - 1$. A **neighbourhood** N of a point $\boldsymbol{x} \in \mathbb{R}^n$ is a set such that there exists an open set U with $\boldsymbol{x} \in U \subseteq N$. An **open neighbourhood** is a neighbourhood that is open. Note that if $U_1 \subseteq \mathbb{R}^n$ and $U_2 \subseteq \mathbb{R}^m$ are open sets, then the product $U_1 \times U_2 \subseteq \mathbb{R}^{n+m}$ is also open.

Any subset $S \subseteq \mathbb{R}^n$ inherits a topological structure from $\mathbb{R}^n$, where the open sets in S are the sets of the form $U \cap S$, with $U \subseteq \mathbb{R}^n$ open. If a set S is contained in a lower-dimensional linear subspace $V \subset \mathbb{R}^n$, say, with $\dim V = k < n$, then S is always closed. However, it can be open *relative to its linear span*,

$$\operatorname{span}(S) = \left\{ \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i : \lambda_1, \ldots, \lambda_k \in \mathbb{R}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in S \right\}.$$

We call a set S **relatively open** or **relatively closed** if it is open or closed in the induced topology on $\operatorname{span}(S)$. Another way of defining this notion goes as follows. Let $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ be an orthonormal basis of $\operatorname{span}(S)$, with $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k)$, and consider the map

$$\varphi_{\boldsymbol{B}} \colon \mathbb{R}^k \to \operatorname{span}(S), \quad \varphi_{\boldsymbol{B}}(\boldsymbol{x}) = \sum_{i=1}^{k} x_i \boldsymbol{b}_i.$$

Then a set S is relatively open or relatively closed if the preimage

$$\varphi_{\boldsymbol{B}}^{-1}(S) = \{\boldsymbol{x} : \varphi_{\boldsymbol{B}}(\boldsymbol{x}) \in S\}$$

is open or closed in $\mathbb{R}^k$. Based on these notions, one defines the **relative closure** $\operatorname{relcl} S$ and **relative interior** $\operatorname{relint} S$ just as before.

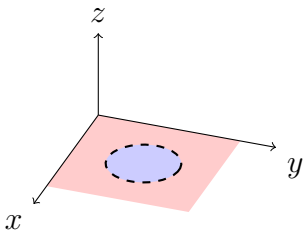


Figure 3.4: The disk without boundary on the xy-plane is relatively open, and is the relative interior of the disk with boundary.

A subset $S \subseteq \mathbb{R}^n$ is **bounded** if there exists number $M > 0$ such that $\|\boldsymbol{x}\|_2 < M$ for all $\boldsymbol{x} \in S$. Invoking the equivalence of norms (3) one sees that this definition does not depend on the norm chosen. A set $K \subseteq \mathbb{R}^n$ is called **compact** if it is closed and bounded. Equivalently, every cover of K with open sets contains a finite subcover. A function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ is **continuous** if for every open set $U \subset \mathbb{R}^m$,

$$f^{-1}(U) := \{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) \in U\}$$

is an open subset of $\mathbb{R}^n$. A function defined on a subset $S \subseteq \mathbb{R}^n$ is said to be continuous if it is continuous on the induced topology. The set of continuous functions $f\colon S \to \mathbb{R}^m$ is denoted by $C(S, \mathbb{R}^m) = C^0(S, \mathbb{R}^m)$. If $f \in C(K, \mathbb{R})$, where K is compact, then f is bounded, and attains its infimum and supremum there: there exist $\boldsymbol{x}_*, \boldsymbol{x}^* \in K$ such that

$$\inf_{\boldsymbol{x}\in K} f(\boldsymbol{x}) = f(\boldsymbol{x}_*), \quad \sup_{\boldsymbol{x}\in K} f(\boldsymbol{x}) = f(\boldsymbol{x}^*).$$

A weaker notion is that of a **Lipschitz continuous** function. A function $f\colon S \to \mathbb{R}^m$ is called Lipschitz continuous with Lipschitz constant $L > 0$, if for all $\boldsymbol{x}, \boldsymbol{y} \in S$,

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

Notions about continuity of functions can be conveniently stated in terms of sequences. A **sequence** of points $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}} \subset \mathbb{R}^n$ (for short, $\{\boldsymbol{x}_k\}$) **converges** to $\boldsymbol{x} \in \mathbb{R}^n$ as $k \to \infty$ with respect to a norm $\|\cdot\|$, written $\boldsymbol{x}_k \to \boldsymbol{x}$, if the sequence of numbers $\|\boldsymbol{x}_k - \boldsymbol{x}\|$ converges to 0,

$$\lim_{k\to\infty} \|\boldsymbol{x}_k - \boldsymbol{x}\| = 0.$$

Formally, this means that for every $\varepsilon > 0$ there exists an index k_0, such that for all $k > k_0$, $\|\boldsymbol{x}_k - \boldsymbol{x}\| < \varepsilon$. From the equivalence of norms (3) it follows that if a sequence converges with respect to one norm, it also converges with respect to the other norms.

A **subsequence** of a sequence $\{\boldsymbol{x}_k\}$ is an infinite subset of S. A **limit point** for a sequence $S = \{\boldsymbol{x}_k\}$ is a point $\boldsymbol{x}$ that is the limit of an subsequence of S. Formally, for every $\varepsilon > 0$ there exists a k_0 such that $\|\boldsymbol{x}_k - \boldsymbol{x}\| < \varepsilon$ *for some* $k > k_0$. A sequence $\{\boldsymbol{x}_k\}$ is called a **Cauchy sequence** if for every $\varepsilon > 0$ there exists an index $k_0 > 0$, such that for all $k, \ell > k_0$, $\|\boldsymbol{x}_k - \boldsymbol{x}_\ell\|_2 < \varepsilon$. The vector space $\mathbb{R}^n$ with the 2-norm (or any other norm) is a **Banach space**, which means that every Cauchy sequence contains a convergent subsequence.

All the topological notions discussed earlier have an interpretation in terms of sequences and limits:

1. A set C is closed if and only if for every sequence $\{x_k\} \subset C$, all limit points are in C;
2. The closure of a set S is the set of all limit points of sequences in S;
3. A set K is compact, if and only if every sequence of points $\{x_k\}$ in K has a limit point in K.

Given a function $f\colon \Omega \to \mathbb{R}^m$, where $\Omega \subseteq \mathbb{R}^n$, and $\boldsymbol{x} \in \Omega$, then f is **continuous at** $\boldsymbol{x}^*$ if

$$\lim_{\boldsymbol{x} \to \boldsymbol{x}^*} f(\boldsymbol{x}) = f(\boldsymbol{x}^*).$$

Formally, for every $\varepsilon > 0$ there exists a $\delta > 0$ such that whenever $\|\boldsymbol{x} - \boldsymbol{x}^*\| < \delta$, $\|f(\boldsymbol{x}) - f(\boldsymbol{x}^*)\| < \varepsilon$. This means that for every sequence of points $\{\boldsymbol{x}_k\}$ with $\lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}^*$, the sequence $f(\boldsymbol{x}_k) \to f(\boldsymbol{x}^*)$ as $k \to \infty$ with respect to some norm on $\mathbb{R}^m$.

Differentiable functions

A function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ is called (Fréchet) **differentiable** at $\boldsymbol{x}_0 \in \mathbb{R}^n$ if there exists a linear map $\boldsymbol{J}f(\boldsymbol{x}_0)\colon \mathbb{R}^n \to \mathbb{R}^m$, such that

$$\lim_{\boldsymbol{h}\to\boldsymbol{0}} \frac{\|f(\boldsymbol{x}_0 + \boldsymbol{h}) - f(\boldsymbol{x}_0) - \boldsymbol{J}f(\boldsymbol{x}_0)\boldsymbol{h}\|_2}{\|\boldsymbol{h}\|_2} = 0.$$

A function is differentiable on an open subset $U \subseteq \mathbb{R}^n$ if it is differentiable at every $\boldsymbol{x} \in U$. If $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^\top$ is differentiable, then all the partial derivatives exist, and $\boldsymbol{J}f(\boldsymbol{x}_0)$ is represented by the **Jacobian matrix**

$$\boldsymbol{J}f(\boldsymbol{x}_0) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix},$$

where the partial derivatives are evaluated at $\boldsymbol{x}_0$. If all the partial derivatives exist and are continuous in a neighbourhood of $\boldsymbol{x}_0$ (called **continuously differentiable**), then f is differentiable at $\boldsymbol{x}_0$.

If $m = 1$, then $J(\boldsymbol{x}_0)$ is the transpose of the **gradient** $\nabla f(\boldsymbol{x}_0)$ of f at $\boldsymbol{x}_0$,

$$\nabla f(\boldsymbol{x}_0) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)^\top.$$

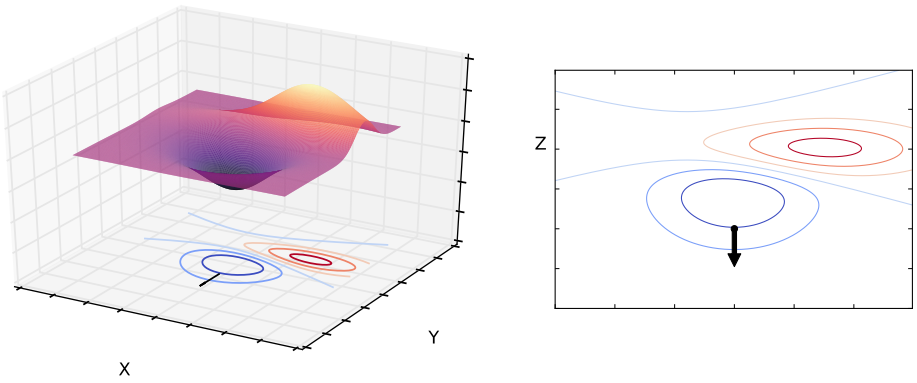


Figure 3.5: A surface, level sets, and the gradient

The gradient points in the direction in which f increases the most.

A convenient way to visualise a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ is through **level sets** $\{\boldsymbol{x} \in \mathbb{R}^2 : f(\boldsymbol{x}) = c\}$. For each $c \in \mathbb{R}$, such a level set defines a curve in $\mathbb{R}^2$, the curve on which the function value does not change. The gradient is always orthogonal to the level set, pointing in the direction in which f increases the most (see Figure 3.5).

If the gradient, considered as a map $\mathbb{R}^n \to \mathbb{R}^n$, is itself differentiable at $\boldsymbol{x}_0$, then the Jacobian matrix of the gradient is called the **Hessian matrix**,

$$\nabla^2 f(\boldsymbol{x}_0) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$

Since

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i},$$

the Hessian is a symmetric matrix.

The **directional derivative** $D_{\boldsymbol{x}_0} f(\boldsymbol{x}_0)$ of a function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ in direction $\boldsymbol{v} \in \mathbb{R}^n$ at $\boldsymbol{x}_0$ is defined as

$$D_{\boldsymbol{v}} f(\boldsymbol{x}_0) = \lim_{h \to 0} \frac{f(\boldsymbol{x}_0 + h\boldsymbol{v}) - f(\boldsymbol{x}_0)}{h}.$$

In the special case where $\boldsymbol{v} = \boldsymbol{e}_i$, we obtain the partial derivative with respect to x_i,

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}_0) = D_{\boldsymbol{e}_i} f(\boldsymbol{x}_0).$$

If $f\colon \mathbb{R}^n \to \mathbb{R}$ is differentiable with continuous derivative near $\boldsymbol{x}_0$, then

$$D_{\boldsymbol{v}} f(\boldsymbol{x}_0) = \nabla f(\boldsymbol{x}_0)^\top \boldsymbol{v} = \langle \nabla f(\boldsymbol{x}_0), \boldsymbol{v} \rangle.$$

If $f\colon \mathbb{R}^n \to \mathbb{R}^m$ and $g\colon \mathbb{R}^m \to \mathbb{R}^p$ are differentiable in a neighbourhood of $\boldsymbol{x}_0 \in \mathbb{R}^n$ and $f(\boldsymbol{x}_0) \in \mathbb{R}^m$, respectively, then the composition $h = g \circ f\colon \mathbb{R}^n \to \mathbb{R}^p$ is continuously differentiable in a neighbourhood of $\boldsymbol{x}_0$, and the Jacobian matrix is defined by the **chain rule**:

$$Jh(\boldsymbol{x}_0) = Jg(f(\boldsymbol{x}_0))Jf(\boldsymbol{x}_0).$$

If $n = 1$, then $f\colon \mathbb{R} \to \mathbb{R}^n$ is called a **curve**, and we write

$$Jf = \frac{\mathrm{d}f}{\mathrm{d}t} = \dot{f} = (\dot{f}_1, \ldots, \dot{f}_n)^\top \in \mathbb{R}^n$$

for the derivative of the curve. If $\dot{f}(t_0) = \boldsymbol{v} \in \mathbb{R}^n$ and $g\colon \mathbb{R}^n \to \mathbb{R}$, then by the chain rule, the derivative of $g \circ f\colon \mathbb{R} \to \mathbb{R}$ is the directional derivative of g in the direction $\boldsymbol{v}$,

$$\frac{\mathrm{d}g \circ f(t_0)}{\mathrm{d}t} = \langle \nabla g(f(t_0)), \boldsymbol{v} \rangle.$$

Before going on to deal with higher derivative, we state the generalisation of the Mean Value Theorem to higher dimensions.

Theorem (Multivariate Mean Value Theorem)**.** Let $f \in C^1(U)$ for an open set U with $\boldsymbol{x}_0, \boldsymbol{x} \in U$, $\boldsymbol{x} \neq \boldsymbol{x}_0$. Then there exists $t \in (0, 1)$ such that

$$f(\boldsymbol{x}) - f(\boldsymbol{x}_0) = \langle \nabla f(t\boldsymbol{x} + (1-t)\boldsymbol{x}_0), \boldsymbol{x} - \boldsymbol{x}_0 \rangle.$$

Note that $t\boldsymbol{x} + (1-t)\boldsymbol{x}_0$ parametrises the line segment connecting $\boldsymbol{x}$ and $\boldsymbol{x}_0$.

For a tuple of natural number $\alpha = (\alpha_1, \ldots, \alpha_n)$, set $|\alpha| = \sum_{i=1}^n \alpha_i$, and define the higher order partial derivative

$$D^\alpha f(\boldsymbol{x}) = \frac{\partial^{|\alpha|} f(\boldsymbol{x})}{\partial^{\alpha_1} x_1 \cdots \partial^{\alpha_n} x_n}.$$

For a set $S \subseteq \mathbb{R}^n$, denote by $C^k(S, \mathbb{R}^m)$ the set of functions f such that all partial derivatives $D^\alpha f$ with $|\alpha| \leq k$ exists and are continuous on $\mathrm{int} S$. If $m = 1$, we write $C^k(S) := C^k(S, \mathbb{R})$.

Define, for a vector $\boldsymbol{x}$ and multi-index α,

$$\boldsymbol{x}^\alpha := x_1^{\alpha_1} \cdots x_n^{\alpha_n}.$$

We then have the **Taylor expansion** around a point $\boldsymbol{x}_0$,

$$f(\boldsymbol{x}) = \sum_{|\alpha| \leq k} D^\alpha f(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0)^\alpha + \sum_{|\alpha| = k} r_\alpha(\boldsymbol{x})(\boldsymbol{x} - \boldsymbol{x}_0)^\alpha,$$

with $r_\alpha(\boldsymbol{x}) \to 0$ as $\boldsymbol{x} \to \boldsymbol{x}_0$.

If a differentiable function $f(\boldsymbol{x})$ has a local minimum or maximum at a point $\boldsymbol{x}$, then this point satisfies $\nabla f(\boldsymbol{x}) = 0$, that is, it is a critical point. The **Lagrange multiplier theorem** says something about local extrema under certain constraints.

Theorem (**Lagrange multipliers**)**.** Let $\boldsymbol{x}^*$ be maximum of $f(\boldsymbol{x})$ under the constraint $g(\boldsymbol{x}) = c$ (that is, a maximum among all points $\boldsymbol{x}$ such that $g(\boldsymbol{x}) = c$). Then there exist a $\lambda \in \mathbb{R}$ such that

$$\nabla f(\boldsymbol{x}^*) = \lambda \nabla g(\boldsymbol{x}^*).$$

The **Lagrangian** of a function $f(\boldsymbol{x})$ with constraint $g(\boldsymbol{x}) = c$ is the function $\Lambda\colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ defined by

$$\Lambda(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda(g(\boldsymbol{x}) - c).$$

The Lagrange multiplier theorem then says that if $\boldsymbol{x}^*$ is a maximum point of $f(\boldsymbol{x})$ under the constraint $g(\boldsymbol{x}) = c$, then there exists $\lambda \in \mathbb{R}$ such that the pair $(\boldsymbol{x}^*, \lambda)$ is a critical point of the Lagrangian $\Lambda(\boldsymbol{x}, \lambda)$.

The **Implicit Function Theorem** is one of the most important results in analysis, and underlies much of differential geometry and physics. Let $F\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ by a function that is continuously differentiable in a neighbourhood of a point $(\boldsymbol{x}_0, \boldsymbol{y}_0)$, with $\boldsymbol{x}_0 \in \mathbb{R}^n$ and $\boldsymbol{y}_0 \in \mathbb{R}^m$. The Jacobian $J_{\boldsymbol{x}}(\boldsymbol{x}_0, \boldsymbol{y}_0)$ with respect to the first set of n coordinates consists of the first n columns of the Jacobian matrix,

$$J_{\boldsymbol{x}}(\boldsymbol{x}_0, \boldsymbol{y}_0) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\boldsymbol{x}_0, \boldsymbol{y}_0) & \cdots & \frac{\partial f_1}{\partial x_n}(\boldsymbol{x}_0, \boldsymbol{y}_0) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\boldsymbol{x}_0, \boldsymbol{y}_0) & \cdots & \frac{\partial f_m}{\partial x_n}(\boldsymbol{x}_0, \boldsymbol{y}_0) \end{pmatrix}$$

The interpretation is that we consider f as a function in only the first set of coordinates, with the remaining ones (denoted by $\boldsymbol{y}$) considered as parameters.

Theorem (Implicit Function Theorem). Let $f\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ be k times continuously differentiable in an open neighbourhood of $(\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathbb{R}^n \times \mathbb{R}^m$, and assume that $f(\boldsymbol{x}_0, \boldsymbol{y}_0) = \mathbf{0}$. Assume further that the Jacobian $J_{\boldsymbol{x}} f(\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathbb{R}^{n \times n}$ in the first n coordinates is *non-singular* at $(\boldsymbol{x}_0, \boldsymbol{y}_0)$. Then there exists an open neighbourhood $\boldsymbol{y}_0 \in U_{\boldsymbol{y}} \subseteq \mathbb{R}^m$, and a function $h \in C^k(U_{\boldsymbol{y}}, \mathbb{R}^n)$ such that

- $h(\boldsymbol{y}_0) = \boldsymbol{x}_0$,
- $f(h(\boldsymbol{y}), \boldsymbol{y}) = 0$ for $\boldsymbol{y} \in U_{\boldsymbol{y}}$.

Moreover, the Jacobian of h is given by

$$Jh(\boldsymbol{y}) = -Jf_{\boldsymbol{y}}(h(\boldsymbol{y}), \boldsymbol{y})(J_{\boldsymbol{x}} f(h(\boldsymbol{y}), \boldsymbol{y}))^{-1}$$

for all $\boldsymbol{y} \in U_{\boldsymbol{y}}$.

Example 1.13. Let $f(x, y) = x^2 + y^2 - 1$ and $(x_0, y_0) = (1, 0)$. The Jacobian of f in the first coordinate is

$$\frac{\partial f}{\partial x}(1, 0) = 2 \neq 0,$$

which is non-singular. Choosing the neighbourhood $U_y = (-1, 1)$, the open interval between -1 and 1, we get the function $h\colon U_y \to \mathbb{R}$ as

$$h(y) = \sqrt{1 - y^2}.$$

This function is defined and differentiable on $(-1, 1)$, and satisfies

$$f(h(y), y) = h(y)^2 + y^2 - 1 = (1 - y^2) + y^2 - 1 = 0, \quad y \in U_y.$$

The derivative of h can be computed using the chain rule:

$$\frac{\mathrm{d} f(h(y), y)}{\mathrm{d} y} = \frac{\partial f}{\partial x}(h(y), y)\frac{\mathrm{d} h}{\mathrm{d} y}(y) + \frac{\partial f}{\partial y}(h(y), y)\frac{\mathrm{d} y}{\mathrm{d} y}$$

from which we get

$$\frac{\mathrm{d}h}{\mathrm{d}y}(y) = -\frac{\partial f}{\partial y}(h(y), y)\left(\frac{\partial f}{\partial x}(h(y), y)\right)^{-1} = -y(1-y^2)^{-3/2}.$$

In this example, the implicit function theorem just gives the usual way of parametrising part of the circle.

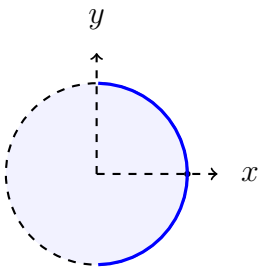


Figure 3.6: The blue arc is parametrised by $h(y) = \sqrt{1-y^2}$ for $y \in (-1, 1)$.

4 Finite precision arithmetic

In practical applications, one often cannot simply plug numbers into formulae and get all the exact results. Most numerical data also requires an infinite amount of storage (just try to store π on a computer!), but a piece of paper or a computer only has limited space. These are some of the reasons that lead us to work with **approximations**.

Measuring errors

To measure the quality of approximations, we use the concept of **relative error**. Given a quantity x and a computed approximation $\hat{x}$, the **absolute error** is given by

$$E_{\mathrm{abs}}(\hat{x}) = |x - \hat{x}|,$$

while the *relative error* is given as

$$E_{\mathrm{rel}}(\hat{x}) = \frac{|x - \hat{x}|}{|x|}.$$

The benefit of working with relative errors is that they are scale invariant. Absolute errors can be meaningless at times: for example, an error of one hour is irrelevant when estimating the age of Stan the Tyrannosaurus Rex at Manchester Museum, but it is crucial when determining the time of a lecture. That is because in the former one hour corresponds to a relative error is of the order 10^{-11}, while in the latter it is of the order 1.

Floating point and significant figures

The established way of representing real numbers on computers is using **floating-point arithmetic**. In the double precision version of the IEEE standard for floating-point arithmetic, a number is represented using 64 bits, where a bit is either 1 or 0. A number is written

$$x = \pm f \times 2^e,$$

where f is a fraction in $[0, 1]$, represented using 52 bits, and e is the exponent, using 11 bits, and one bit is for the sign. There are largest possible numbers, and there are gaps between representable numbers. The largest and smallest numbers representable in this form are of the order of $\pm 10^{308}$, enough for most practical purposes. A bigger concern are the gaps, which means that the results of many computations almost always have to be rounded to the closest floating-point number.

When going through calculations without using a computer, we usually use the terminology of **significant figures** (s.f.) and work with 4 significant figures in base 10. For example, in base 10, $\sqrt{3}$ equals 1.732 to 4 significant figures. To count the number of significant figures in a given number, start with the first non-zero digit from the left and, moving to the right, count all the digits thereafter, counting final zeros if they are to the right of the decimal point. For example, 1.2048, 12.040, 0.012048, 0.0012040 and 1204.0 all have 5 significant figures (s.f.). In rounding or truncation of a number to n s.f., the original is replaced by the closest number with n s.f. An approximation $\hat{x}$ of a number x is said to be **correct to n significant figures** if both $\hat{x}$ and x round to the same n s.f. number.

Remark 1.14. Note that final zeros to left of the decimal point may or may not be significant: the number 1204000 has a least 4 significant figures, but without any more information there is no way of knowing whether or not any more figures are significant. When 1203970 is rounded to 5 significant figures to give 1204000, an explanation that this has 5 significant figures is required. This could be made clear by writing it in scientific notation: 1.2040×10^6. In some cases we also have to agree whether to round up or round down: for example, 1.25 could equal 1.2 or 1.3 to two significant figures. If we agree on rounding up, then to say that $a = 1.2048$ to 5 s.f. means that the exact value of a satisfies $1.20475 \leq a < 1.40485$.

Example 1.15. Suppose we want to find the solution to the quadratic equation

$$ax^2 + bx + c = 0.$$

The two solutions to this problem are given by

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \tag{1.1}$$

In principle, to find x_1 and x_2 one only needs to evaluate the expressions for given a, b, c. Assume, however, that we are only allowed to compute to four significant figures, and consider the particular equation

$$x^2 + 39.7x + 0.13 = 0.$$

Using the formula 1.1, we have, always rounding to four significant figures,

$$a = 1, b = 39.7, c = 0.13,$$

$$b^2 = 1576.09 = 1576 \text{ (to 4 s.f.) }, 4ac = 0.52 \text{ (to 4 s.f.)},$$
$$b^2 - 4ac = 1575.48 = 1575 \text{ (to 4 s.f.) }, \sqrt{b^2 - 4ac} = 39.69.$$

Hence, the computed solutions (to 4 significant figures) are given by

$$\overline{x}_1 = -0.005,\ \overline{x}_2 = -39.69$$

The exact solutions, however, are

$$x_1 = -0.0032748...,\ x_2 = -39.6907...$$

The solution x_1 is completely wrong, at least if we look at the relative error:

$$\frac{|\overline{x}_1 - x_1|}{|x_1|} = 0.5268.$$

While the accuracy can be increased by increasing the number of significant figures during the calculation, such effects happen all the time in scientific computing and the possibility of such effects has to be taken into account when designing numerical algorithms.

By analysing what causes the error it is sometimes possible to modify the method of calculation in order to improve the result. In the present example, the problems are being caused by the fact that $b \approx \sqrt{b^2 - 4ac}$, and therefore

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-39.7 + 39.69}{2}$$

causes what is called "catastrophic cancellation". A way out is provided by the observation that the two solutions are related by

$$x_1 \cdot x_2 = \frac{c}{a}. \tag{1.2}$$

When $b > 0$, the calculation of x_2 according to (1.1) shouldn't cause any problems, in our case we get -39.69 to four significant figures. We can then use (1.2) to derive $\overline{x}_1 = c/(a\overline{x}_2) = -0.00327$.

There are other potential sources of error besides those introduced by rounding operations.

1. Overflow
2. Errors in the model
3. Human or measurements errors
4. Truncation or approximation errors

The first is rarely an issue, as we can represent numbers of order 10^{308} on a computer. The second two are important factors that need to be addressed when working on real-world problems. The fourth has to do with the fact that many computations are done approximately rather than exactly. For computing the exponential, for example, we might use a method that gives the approximation

$$e^x \approx 1 + x + \frac{x^2}{2}.$$

As it turns out, many optimization problems work with approximations of the functions of interest, and the solution found is only an approximation to the "true" solution of the problem. Quantifying the quality of such an approximation is an important aspect in the design and analysis of optimization algorithms.