Renewable Energy Online
Robin Knecht, Paul Ziethe

Renewable Energy Basics

Lehrbrief

PUBLIKATION DER BILDUNGSALLIANZ MINT.ONLINE:
UNIVERSITÄT OLDENBURG, UNIVERSITÄT KASSEL, UNIVERSITÄT STUTTGART, FERNUNIVERSITÄT IN HAGEN, FRAUNHOFER-GESELLSCHAFT, DLR-INSTITUT FÜR VERNETZTE ENERGIESYSTEME (EHEMALS NEXT ENERGY)

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung, und Forschung unter dem Förderkennzeichen 16OH12044 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor/bei der Autorin.

Renewable Energy Online

Renewable Energy Basics

edited by

Robin Knecht

(c) Carl von Ossietzky University of Oldenburg (2017)

This document has been typeset using the LaTeX2e bundle on TeX.

compilation date: 15th September 2017

Imprint:

Authors:	Robin Knecht, Paul Ziethe
Publisher:	Carl von Ossietzky University of Oldenburg
Edition:	1st edition (2017)
Editor:	Robin Knecht
Layout:	Robin Knecht

Copyright:	© 2017 Postgraduate Programme Renewable Energy. Any unauthorized reprint or use of this material is prohibited. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system without express written permission from the author/publisher.

Oldenburg, September 2017

Contents

I Mathematics Primer 1

11 Logic, Sets and Sequences 1

11.1 Logic 2

11.2 Sets 7

11.3 Sequences and series 11

11.4 Conclusion 13

11.5 Literature 14

12 Functions 1

12.1 Mapping 2

12.2 Polynomial Function 4

12.3 Curve properties 6

12.4 Exponential and Logarithmic Functions 9

12.5 Trigonometric functions 12

12.6 Literature 16

13 Calculus 1

13.1 Differentiation 2

13.2 Integration 8

13.3 Special Series: Episode 2 13

13.4 Literature 14

14 Beyond one dimension 1

14.1 Vectors and Matrices 2

14.2 Coordinate systems 9

14.3 Calculus on vectors 12

14.4 Conclusion 17

14.5 Literature 17

15 Differential Equations 1

15.1 Complex numbers 2

15.2 Systems of equations 5

15.3 Differential Equations 11

15.4 Numerics 15

15.5 Literature 16

16 Stochastics 1

16.1 Distributions of event outcomes 2

16.2 Sequences of measurements 5

16.3 Statistical measures 6

16.4 Statistical distributions 7

16.5 Statistics of measurements 11
16.6 Transitions 15
16.7 Literature 18

II Programming 1

21 MATLAB and its friends 1
21.1 Getting to know your tools 2
21.2 Data types 7
21.3 Script files 10
21.4 Special operations 13
21.5 Literature 15

22 Programming 1
22.1 The very basics 2
22.2 Plotting 4
22.3 Case differentiation 9
22.4 Loops 14
22.5 Function files 18
22.6 Literature 22

23 Modeling 1
23.1 Reading from and writing to data files 2
23.2 Interpolation 5
23.3 Fitting 8
23.4 Modeling 13
23.5 Literature 18

24 Special Features 1
24.1 Interpolation 2
24.2 Fitting 5
24.3 Multi-Linear-Regression 9
24.4 Literature 10

III Solid State Physics 1

31 Solid State Basics 1
31.1 Crystals 2
31.2 Schroedinger equation 5
31.3 Wavefunctions of electrons 6
31.4 Fermi distribution 8
31.5 Material classes 10
31.6 Energy-Wavevector Diagrams 11
31.7 Densitiy of states 15
31.8 Literature 17

32 Semiconductors 1
32.1 Conductivity 2
32.2 Charge carrier densities 3
32.3 Law of mass action 4
32.4 Doping 6
32.5 Bias, quasi fermi levels 8
32.6 Energy levels 9
32.7 Origins of quasi Fermi level gradients 11

32.8 Literature 15

33 Generation & Recombination 1

33.1 Generation 2

33.2 Recombination 5

33.3 Efficiency limit of Solar Cells 10

33.4 Literature 18

34 Junctions 1

34.1 Junction types 2

34.2 Depletion approximation 6

34.3 Variations of the pn-junction 10

34.4 Transport equations 12

34.5 Literature 15

IV Electrical Power Systems 1

41 DC Circuit Concepts 1

41.1 Voltage - Current - Power 1

41.2 Calculation in Electrical Circuits 2

41.3 Behaviour of Electrical Components 4

42 AC Circuit Concepts 1

42.1 Harmonic Wave Relation 1

42.2 Root-Mean-Square Value (Effective Value) 2

42.3 Power in a Single Phase AC Circuit 3

42.4 Basics of Complex Numbers 7

42.5 Complex Notation of Electrical Quantities 8

42.6 Components in AC Circuits 10

42.7 Phasor Diagrams 14

42.8 Complex Power 16

42.9 Compensation of Reactive Power 19

43 Balanced 3 phase AC circuits 21

43.1 Polyphase Current 1

43.2 Wye-(or Star-) Connected Load 3

43.3 Delta-Connected Load 4

43.4 Power in a Balanced 3P System 5

43.5 Time dependency of the power 6

43.6 Symmetry Behaviour 8

44 Magnetic Circuits 1

44.1 Basic Quantities 1

44.2 Analogy between Magnetic and Electric Circuits 3

44.3 Losses in Magnetic Materials 4

44.4 Faradayâs Law 5

44.5 Inductance 6

45 Transformers 1

45.1 Ideal Transformer 1

45.2 Non-ideal Transformer 4

45.3 Autotransformer 5

45.4 3-Phase Transformers 6

46 DC Machines 1

46.1 Fundamentals of Rotating Machines 1

46.2 Introduction to DC Machines 3

46.3 Construction of a DC Machine 4

46.4 Generation of an Unidirectional Voltage 6

46.5 Machine Analysis 8

46.6 Classification of DC Machines 9

46.7 Shunt DC Motor 12

46.8 Series DC Motor 15

47 Induction Machines 1

47.1 Introduction to Induction Machines 1

47.2 Construction 2

47.3 Rotating Magnetic Field 4

47.4 Running Operation 5

47.5 Induced Voltages 7

47.6 Performance Characteristics 9

47.7 Effects of Rotor Resistance 11

47.8 Speed Control 13

47.9 Modes of Operation 14

47.10 Double Fed Induction Machine 15

48 Synchronous Machines 1

48.1 Introduction to Synchronous Machines 1

48.2 Construction 2

48.3 Synchronous Generators 4

48.4 Synchronous Motors 8

48.5 Power and Torque Characteristic 10

48.6 Power Factor Control 12

48.7 Independent Single Generators 16

Part I

Mathematics Primer

12 — Functions

Learning objectives

After processing this chapter you will

- be able to trace a function
- comfortably apply polynomic, exponential and trigonometric functions
- have an idea about the Taylor and Fourier series

For many systems investigated in engineering and natural science the quantities of interest are dependent on other parameters. This behavior is formalized in functions and it is of great importance to describe, apply and interprete functions of many kinds. Especially important are polynomial, exponential and trigonometric functions which will be discussed in this chapter. A more abstract understanding of functions is also very useful in the programming of software as they consitute a very fundamental method to write good code.

12.1 Mapping

Let's look again at the sequences we encountered in the previous chapter. You could say that the integer n is associated to the element a_n. This concept of associating an element of one set (the so-called **domain**) to another set (**range**) can be generalized.

Definition 12.1

A **mapping** f of a domain set D into a range set R is conceived as a rule that uniquely allocates each $a \in D$ to a distinct $b = f(a) \in R$: $f : D \longrightarrow R$

The notation used for the allocation of elements is $a \mapsto b = f(a)$.

b is called **image** of a, whereas a is called **prefiguration** of b.

A mapping can be illustrated as in Fig. 12.1.

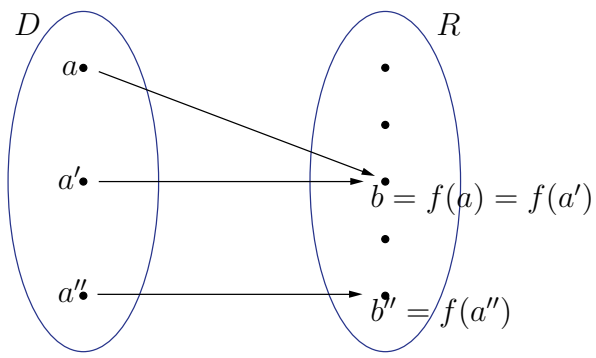

Figure 12.1: Illustration of a mapping

Figure 12.1 shows not all elements from R must be represented as image of an element from D and an element from R may be image of several elements from D. However, there must be a unique image for each element from D , i.e. from each a must originate exactly one arrow.

Notice that an image b may have several prefigurations e.g. a and a' .

Example 12.1 — Sequences as mappings.

A sequence is a special case of a mapping f where the domain $D = \mathbb{N}$. The range R may be any set and thus we can write: $f : \mathbb{N} \longrightarrow R$. For the elements we write: $n \mapsto a_n$.

Instead of mapping the term **function** is employed in particular in real and complex analysis.

In the case the range and the domain are one-dimensional sets of numbers the behavior of functions can be easily plotted in a coordinate system such as Fig. 12.2. On the horizontal axis (**abscissa**) the domain is indicated while on the vertical axis (**ordinate**) the range is indicated. The allocation of a particular domain element x to its image $y = f(x)$ is represented as a **point** (x, y) in the coordinate system.

Plotting all the points representing the allocation of the function results in a **curve**. The ability to predict the general behavior of a curve based on its formula is called curve tracing.

Example 12.2 — Multi-part definitions.

Functions may be defined differently for different subsets of the domain. An example for such a **combined function** is illustrated in Fig. 12.2 defined by

$$f(x) = \begin{cases} x^2 & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

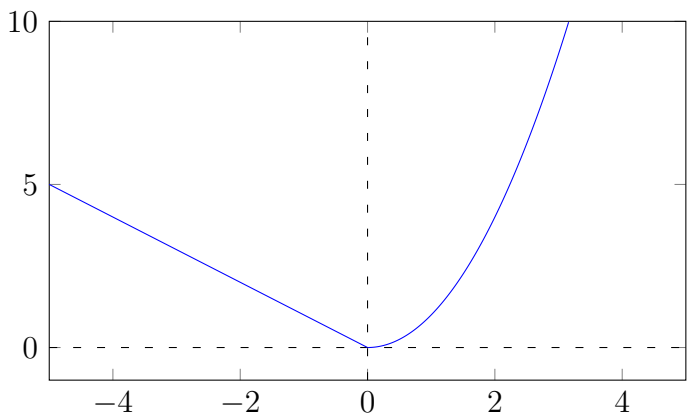

Figure 12.2: Coordinate system illustrating the curve of a combined function

Additionally functions may be composited. Consider the two functions $f : B \longrightarrow C$ and $g : A \longrightarrow B$ then the **composition** of both functions is: $f \circ g = f(g(x))$.

If for each image $b \in f(D)$ there exists only one prefiguration a then it is possible to define an **inverse** function f^{-1} to f: $f^{-1} : f(D) \subset R \longrightarrow D$. This inverse function is practically found by substituting x with y^* and y with x^* in the formula of f and solving for y^*. Graphically the inverse is constructed by mirroring the curve for f at the identity $y = x$.

12.2 Polynomial Function

Polynomial functions are very relevant in natural sciences as they deliver an easy method to parametrize any arbitrary functional behavior and as such are often used as a first tool to characterize the behavior of a quantity based on empirical measurements. Later in the course will be shown that any function can be expressed as a polynomial and they often act as approximation models to actually enable the analytical or numeric solvability of a problem.

Definition 12.2

A **polynomial** function $f(x)$ is a mathematical construct where the element x from a set of numbers is allocated to an element y of another set of numbers according to the general formular expression:

$$y = \sum_{i \in \mathbb{N}, i=0} a_i x^i \tag{12.1}$$

The a_i are called **coefficients** of the **order** i. Notice, that this definition includes an order 0.

In general the polynomial may consist of infinite summands. A polynomial is called finite of order N when the coefficients a_i for the orders $i > N$ are all 0.

Example 12.3 — Low order polynomials.

A quadratic function is a polynomial function of order 2 and can thus be written as:

$$y = a_2 x^2 + a_1 x + a_0 \tag{12.2}$$

Often alternative names for the coefficients can be found for low order polynomials and it is common to find the general formula: $y = ax^2 + bx + c$ for the quadratic function.

The polynomial of order 1 is called linear function and has the general formula:

$$y = a_1 x + a_0 \tag{12.3}$$

Its curve representation in a coordinate system is a straight line with a constant slope m and a y-offset b. Notice, that these characteristic parameters are identical to the coefficients: $m = a_1$, $b = a_0$. Therefore the expression $y = mx + b$ is also commonly found in the literature.

The set of locations $\{x_i\}$ where the curve intersects with the abscissa are called **roots** of the function. As for those values $y = 0$ they can be found by solving $f(x) = 0$.

For polynomials of order $N > 1$ in general more than one root exists. Finding them analytically can become quite a complex task and often numerical methods are employed. Yet, the method to determine the roots of quadratic functions should be known intuitively. Considering the alternative expression for quadratic functions as given in Example 12.3 the roots are given by:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{12.4}$$

12.2.1 Rational Function

A **rational function** r is the quotient of two polynomials p, q:

$$r(x) = \frac{p(x)}{q(x)} = \frac{a_0 + a_1 x + \cdots + a_m x^m}{b_0 + b_1 x + \cdots + b_n x^n} \,. \tag{12.5}$$

Rational functions are the generalization of polynomials (for the latter: $q = 1$).

Notice that the rational function is undefined when the denominator becomes zero. That means that the roots $\{x_i\}$ of the denominator are not elements of the domain D. Therefore the domain must be given as $D \setminus \{x_i\}$. The roots of the denominator are **singularities** of the rational function.

Example 12.4 — Coulomb's law.

Coulomb's law describes the force F between two electrically charged bodies with distance r.

$$F(r) = k_e \frac{|q_1 q_2|}{r^2} = \frac{C}{r^2} \tag{12.6}$$

Coulomb's constant k_e and the electrical charges q_1 resp. q_2 can be summarized into the constant C. This function can be interpreted as a rational function where the numerator p is a constant and the denominator $q = r^2$ is a quadratic function. It has a singularity at $r = 0$ (see Fig. 12.3).

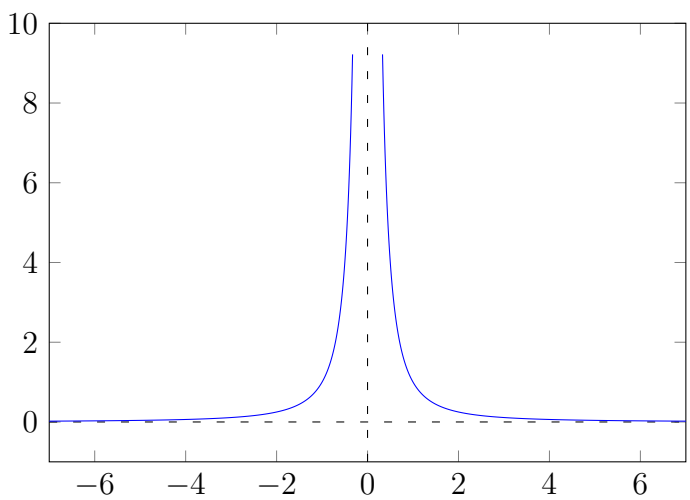

Figure 12.3: Coulomb's law with a simple pole at $r = 0$

12.3 Curve properties

12.3.1 Continuity

At the singularity in Example 12.4 the function is not continuous. When dealing with functions it is very important to know the locations of their discontinuities. Therefore they should be included in the description of a curve. The property of continuity is furthermore very important for some theorems that will be introduced in the following lessons which is why this property will be formally defined. Notice that this definition is implemented by means of sequences and their property of convergence which we introduced in the previous session.

Definition 12.3 — Continuity.

A function f is **continuous** in point a, if for all sequences x_n in the domain with limit a the values of $f(x_n)$ converge against $f(a)$:

$$x_n \to a \implies f(x_n) \to f(a) \,, \tag{12.7}$$

The intuition of this concept is as follows: Be $\delta_\varepsilon > |x - a|$ a distance in the domain then for continous functions there exists a distance $\varepsilon > 0$ in the range for which is valid that $|f(x) - f(a)| < \varepsilon$. We write $\lim\limits_{x \to a} f(x) = f(a)$.

A function is continuous on the interval D, if it is continuous at every point of D. That means that the graph is coherent, as the function has no poles or "jumps". The following example illustrates different types of discontinuities:

Example 12.5 — Discontinuities.

- The signum function $\mathrm{sign}(x)$ has a **jump** at $x = 0$: $\mathrm{sign}(0) = 0$ (see Fig. 12.4a). Though the function is well defined for $x = 0$, the limit of $f(x)$ for $x \to 0$ does not exist.
- The function $g(x) = \frac{x+1}{x^2-1}$ has discontinuities at $x = \pm 1$ (see Fig. 12.4b). For $x \to +1$ $g(x)$ approaches $\pm\infty$. g has a **pole** there and no limit exists for $x = +1$.

The expression for $g(x)$ may be reduced as the numerator is a factor of the denominator

$g(x) = \frac{x+1}{x^2-1} = \frac{x+1}{(x+1)(x-1)} = \frac{1}{x-1}$.

While $x = -1$ will still be a singularity due to the original definition the limit $\lim\limits_{x \to -1} g(x)$ exists. This is called a **removable discontinuity**. By creating a multi-part definition and adding the function value $g(-1) = -\frac{1}{2}$, $g(x)$ becomes a continuous function of $\mathbb{R} \setminus \{1\}$.

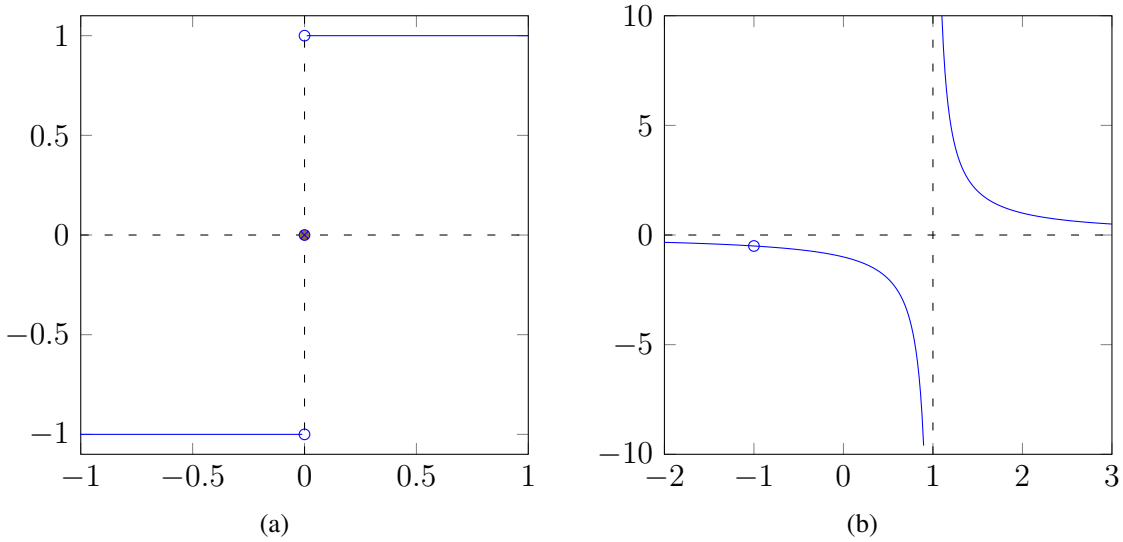

Figure 12.4: Examples of dicontinuities: (a) The signum function $f(x) = \text{sign}(x)$ features a jump at $x = 0$ (b) The rational function $g(x) = \frac{x+1}{x^2-1}$ features a pole at $x = +1$ and a removable discontinuity at $x = -1$.

12.3.2 Curve Tracing

Curve tracing is the ability to sketch the general curvature of a function based on a few characteristic points and its behavior close to the infinite. Characteristic points of rational functions are its roots, singularities and the y-offset as well as the behavior close to the infinite for very small and very large abscissa and around the singularities.

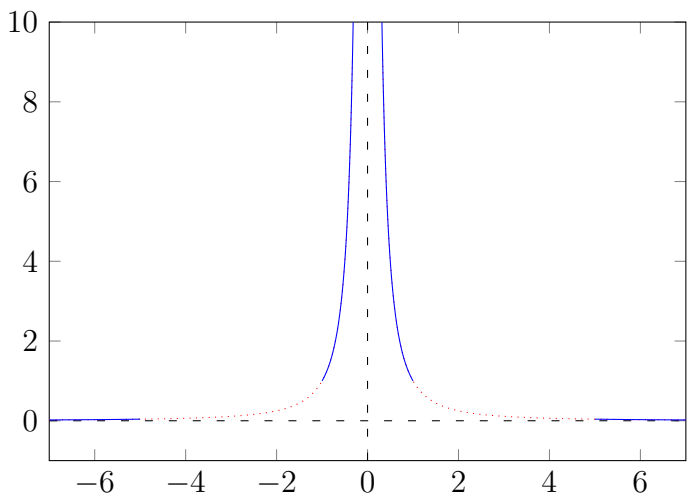

Figure 12.5: Curve tracing of Coulomb's law

Procedure

- Find the singularities of the rational function by finding the roots of the denominator polynomial q: $0 = q(x)$
- Find the roots of the rational function by finding the roots of the numerator polynomial p: $0 = p(x)$
- Find the y-offset by solving the rational function at $x = 0$: $r(x = 0)$
- Mark the y-offset and the roots with crosses in the coordinate system and the singularities with vertical dashed lines.

- Choose a very small (i.e. negative) number and investigate whether the function becomes increasingly negative or positive or approaches a limit.
  - Determine whether p or q is of larger order, this polynomial will dominate close to infinity.
    - In the case q is larger the rational function approaches 0.
      - Determine whether 0 is approached from the positive or the negative by the sign of the largest order summand.
    - In the case p is larger the rational function becomes infinite for large values.
      - Determine whether the largest order is even or odd. In the case it is odd both the negative and positive infinite branch act the same. In the case it is even both behave oppositely.
      - Determine whether the small number becomes increasingly negative or positive.
  - Repeat for very large numbers
- For each singularity choose a value only slightly smaller and larger than the value of the singularity. Determine whether the values approach positive or negative infinity or a constant value.
- Mark the trends in the coordinate system
- Connect the marked lines and points.

Example 12.6 — Curve tracing of Coulomb's law.

Singularities: $q(r) = r^2 = 0$ for $r = 0$
Roots: $p(r) = C \neq 0$ therefore no roots exist
y-Offset: As $r = 0$ is not an element of the domain there exists no y-offset

The larger order polynomial is q with order 2 (which is even). As such the rational function approaches zero for both positive and negative large values. The sign of the largest order summand is positive therefore the curve approaches the x-axis from the positive side. As the order of q is even this is also valid for very large negative values.

For values slightly lower as the singularity the value of $F(r)$ becomes positively infinite. The same is true for slightly larger values.

Thus we draw the information into a plot like in Fig. 12.5. Connecting these pieces of information gives a result very close to the actual behavior.

12.4 Exponential and Logarithmic Functions

Exponential functions are maybe the most important functions for natural processes. They are used whenever a number of entities is governed by probability or changing rates. For example the growth of biomass, the decay of radioactive material or the absorption of light in matter. Furthermore this type of functions is also fundamental for other sciences such as economics where power functions are used to calculate the effect of an interest rate on capital. Logarithmic functions are the inverse function of power functions and also play an important role in the empiric analysis of physical systems.

12.4.1 General exponential functions

Definition 12.4

For a general $a > 0$ an **exponential** function y is defined by

$$y = c \cdot a^x \tag{12.8}$$

where c is a constant coefficient and a is called the **base**.

There exist a few rules for the calculation with exponential functions which you should be well comfortable with:

Theorem 12.1 — Calculation rules for exponentials.

$$\begin{aligned} a^x a^y &= a^{x+y} \\ a^x / a^y &= a^{x-y} \\ (a^x)^y &= a^{xy} \end{aligned} \tag{12.9}$$

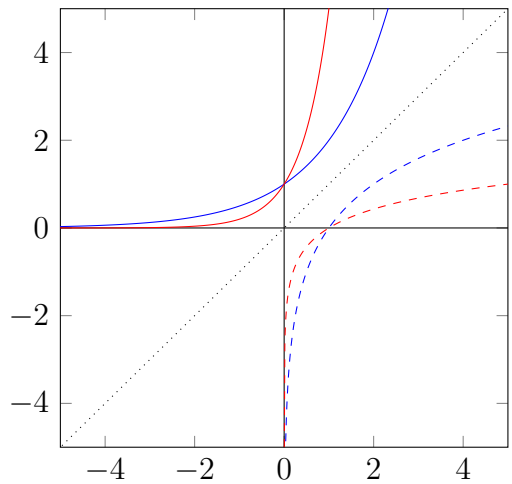

Figure 12.6: Exponential functions with different bases (blue=2, red=5) and their inverse logarithmic functions (dashed). The identity function (dots) is added to emphasize the graphical construction by mirroring.

The inverse of an exponential function is a **logarithmic** function. Just as there exist exponential

functions with different bases this is true for logarithmic functions which is the reason why the base should always be noted as an index: $y = \log_a(x)$.

Unfortunately there exists no easy analytical expression for this function which is why for many years mathematicians had to consult logarithmic tables.

Just as for the calculations with exponentials you should remember the corresponding rules for the calculations with logarithms:

Theorem 12.2 — Calculation rules for logarithms.

$$\begin{aligned} \log_a x + \log_a y &= \log_a (xy) \\ \log_a x - \log_a y &= \log_a (x/y) \\ \log_a x^t &= t \log_a x \end{aligned} \tag{12.10}$$

Let's apply those calculation rule to find a very useful relationship.

Example 12.7 — Change of base.

Consider $y = b^x$:

We can find an expression for x by taking the logarithm to the base b on both sides:
$\log_b y = \log_b b^x = x$

Alternatively the same can be applied with any base a: $\log_a y = \log_a b^x = x \log_a b$

On the right hand side we want to see the x alone, thus we rephrase: $\frac{\log_a y}{\log_a b} = x$

Equating the two expressions for x illustrates that the logarithm to different bases just differs by a constant factor: $\frac{1}{\log_a b} \log_a y = \log_b y$

For this reason a few reference bases have established from which all the other bases can be calculated. On calculators LOG usually stands for the logarithm to the base of 10 which alternatively is also often abbreviated $\lg$.

12.4.2 'The' exponential function

There exists a special base called **Euler's number** $e = 2.71828\ldots$ which has certain advantageous properties as we will see in a future lesson. For this yet unspecified reason in natural sciences exponential functions are almost exclusively expressed in terms of Euler's number. The logarithm with Euler's number as its base is called **natural logarithm** $\ln$.

Example 12.8 — Exponentials in terms of Euler's number.

Consider the general expression $y = c \cdot a^x$:

Applying the natural logarithm on both sides gives: $\ln y = \ln(c \cdot a^x) = \ln c + x \ln a$

Exponentiating these expressions results in: $e^{\ln y} = e^{\ln c + x \ln a} = e^{\ln c} e^{x \ln a}$

This simplifies to: $y = ce^{x \ln a} = ce^{\lambda x}$ with $\lambda = \ln a$.

Thus any general exponential function can be expressed in the form: $y = ce^{\lambda x}$ which is the form usually used in scientific literature.

In the case λ is positive we talk of **exponential growth**, the case λ being negative is called **exponential decay**. Notice that the coefficient c corresponds to an initial value $c = y(x = 0)$. The reciprocal value of λ has the same unit as x and therefore usually has a more intuitive interpretation. However, for many applications other characteristic quantities are used such as the well known **half-life** $T_{1/2}$, i.e. the time required until only half of the initial value c remains. The half-life is reasonable for exponential decay, for exponential growth we use the doubling time T_2, i.e. the time until the initial value has doubled.

12.4.3 Application of logarithms in data analysis

The dramatic changes in magnitude of exponential functions make it impractical to plot them in the usual coordinate system. Using a so-called **semilogarithmical plot** $\ln y(x) = \ln c + \lambda x$ the exponential curve becomes a straight line which can more easily be analyzed. Notice, that the slope $m = \lambda$ and the y-offset b corresponds to the coefficient c via $b = \ln c$.

Another situation where the properties of logarithms are useful is the determination of the power exponent a of a power function $y = x^a$. Applying a logarithm to both sides gives:
$\ln y = \ln x^a = a \ln x$. Thus the power exponent can be extracted as the slope of a so-called **double-logarithmic** (or: log-log) plot.

Example 12.9 — Stefan-Boltzmann law.

Figure 12.7 shows the power emission P of a black body with temperature T which seems to be a power function with unknown exponent. The double-logarithmic plot reveals a straight line confirming that indeed the power emission is a power function of the temperature. For the slope and as such the exponent we obtain $a = 4$. Thus we can express the power emission in terms of temperature by the Stefan-Boltzmann law $P = \sigma T^4$ with the constant coefficient σ.

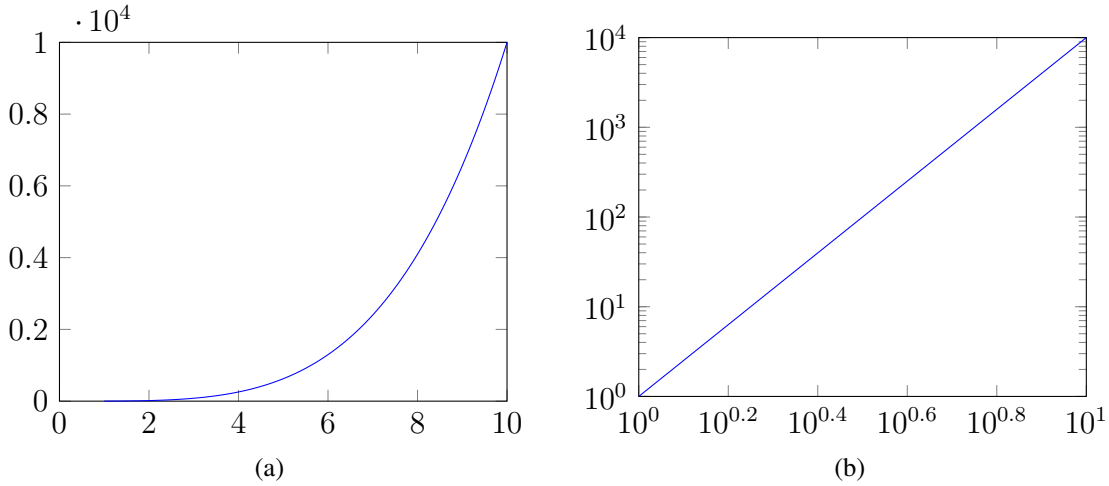

Figure 12.7: (a) Stefan-Boltzmann law on a linear scale (b) Stefan-Boltzmann law in double logarithmic presentation

12.5 Trigonometric functions

Trigonometric functions have many applications in natural science. For instance they are used to describe alternating currents and electromagnetic waves which are important for renewable energies. Furthermore trigonometric functions are used for calculations of geometry whenever the angle between objects play a role or for anything that involves rotations such as for example the position of the sun in the sky. This application in geometry extends also to any equations where physical quantities have a direction. Another practical use of trigonometric functions is yet another mathematical series used especially when descibing periodic functions.

12.5.1 The three major trigonometric functions

There exist quite a handful of trigonometric functions but here we will focus on the three most important ones.

Definition 12.5 — Definitions.

Consider the right triangle depicted in Fig. 12.8. Using the reference angle α we call the edge b **adjacent**, a **opposite** and c **hypotenuse**. Upscaling the triangle in size does not change the relationship between the triangle edges a, b and c. These relationships are therefore characteristic for the given triangle and we define certain ratios:

$$\sin\alpha = \frac{a}{c} \text{ (Sine)} \tag{12.11}$$

$$\cos\alpha = \frac{b}{c} \text{ (Cosine)} \tag{12.12}$$

$$\tan\alpha = \frac{\sin\alpha}{\cos\alpha} = \frac{a}{b} \text{ (Tangent)} \tag{12.13}$$

Thus knowing two angles of the triangle (α and the right angle) as well as one triangle edge the missing angle and edge lengths can be calculated. The missing angle as $\beta = 90° - \alpha$ due to the law of angular sums and the missing lengths using the trigonometric functions. Notice that at the moment the domain of these trigonometric functions is just $(0...90°)$ and that they are defined in terms of an angle instead of a pure number.

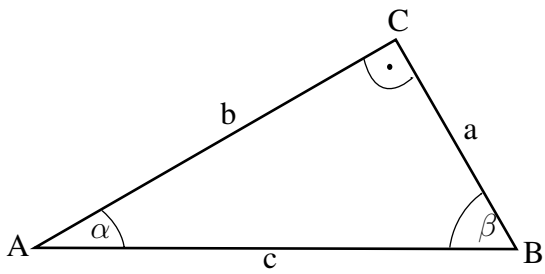

Figure 12.8: Graphical presentation of the basic trigonometric functions

Also notice that these functions are not given in terms of arithmetic expressions. Only for a few specific angles the values can be calculated using the law of Pythagoras:

Example 12.10 — Selected values of trigonometric functions via Pythagoras.

The law of Pythagoras being valid for right triangles (such as Fig. 12.8) states: $a^2 + b^2 = c^2$

- For an angle $\alpha = 45°$: $a = b$.
  Therefore we can write: $2a^2 = c^2 \Rightarrow c = \sqrt{2}a$ Using the definition of the sine we can thus write: $\sin 45° = \frac{a}{\sqrt{2}a} = \frac{1}{\sqrt{2}}$
  As $a = b$ the same is valid for the cosine: $\sin 45° = \cos 45°$
  For the tangent we obtain very easily: $\tan 45° = \frac{a}{a} = 1$
- For an equilateral triangle ($\alpha = \beta = \gamma = 60°$, a=b=c) we construct a right triangle by dividing it in half. Thus the hypotenuse of the new right triangle has the length a while the adjacent is $\frac{1}{2}a$. Thus: $\cos 60° = 0.5$
  Using Pythagoras we find the length h of the opposite of this right 'half' triangle : $\left(\frac{a}{2}\right)^2 + h^2 = a^2$
  Solving for h we obtain: $h = \frac{\sqrt{3}}{2}a$ and thus $\sin 60° = \frac{\sqrt{3}}{2}$, $\tan 60° = \sqrt{3}$

For other values mathematicians in the past refered to tables.

12.5.2 The unit circle

Consider a circle with radius $r = 1$ (unit circle) drawn around the origin of a coordinate system as in Fig. 12.9a. We define the positive abscissa as $0°$ increasing counter clockwise. Using this definition for any angle α of the full circle $(0...360°)$ a right triangle can be constructed using the radius as the hypotenuse.

As the hypotenuse is equal to 1 the values of the sine and cosine correspond to the coordinates of the triangle corner P on the unit circle $(\sin\alpha, \cos\alpha)$.

Notice that the **arc length** s of the unit circle increases linearly with α. The circumference of the unit circle is $2\pi r = 2\pi$ which corresponds to the full $360°$. The relationship between s and α is:

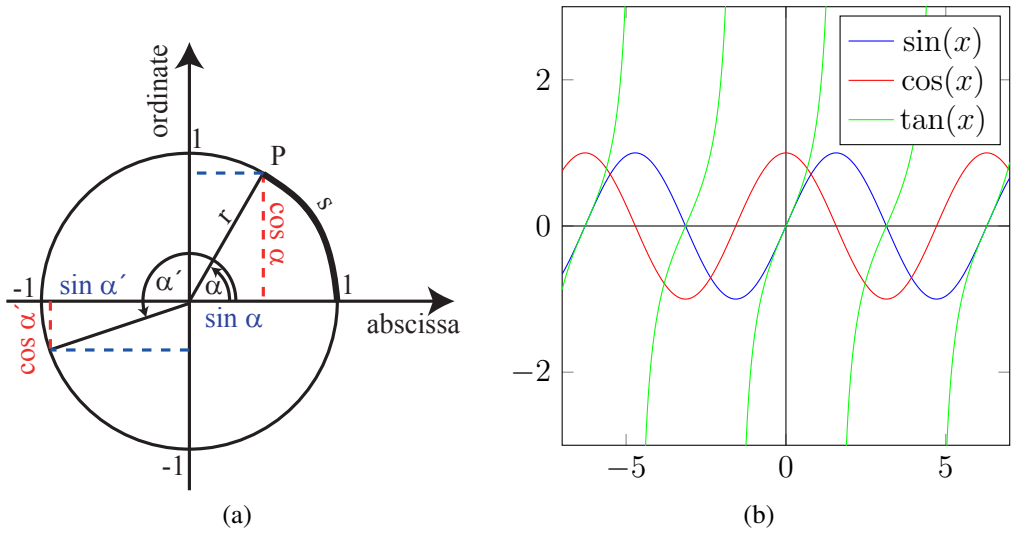

Figure 12.9: (a) The basic trigonometric functions as measures in a unit circle (b) Trigonometric functions in terms of radians

$\frac{s}{2\pi} = \frac{\alpha}{360}$ or $s = \alpha\frac{\pi}{180}$. Very often the size of the angle is given in terms of the arc length and in this case we call the unit of the angle the **radian**.

This difference becomes especially important when programming software. Usually these trigonometric functions are provided by the programming language. Refer to the function reference if the function expects degrees or radians and translate the numeric value accordingly otherwise you will end up with messy data.

Until now, we have restricted our domain to $0...360°$ or $0...2\pi$ radians. However, it is no stretch of mind to imagine that we continue with the rotation of the unit circle pointer. Naturally the sine and cosine values will now repeat themselves again and again. This behavior is called **periodicity**.

Figure 12.9b illustrates the function values of the three main trigonometric functions in terms of increasing arc length. Notice the periodicity of the functions beyond 2π. We call 2π the length of one **period** T. In fact we have now even extended the domain into the negative numbers.

Also notice that the sine and cosine are essentially the same function only shifted by a quarter of a period: $\frac{\pi}{2} = 90°$. This can be easily understood by rotating Fig. 12.9a by $90°$.

As the sides of the triangle can maximally become as large as the radius the values of the sine and cosine functions alternate between -1 and 1. Notice that whenever the cosine becomes zero the tangent is undefined and features a discontinuity.

Example 12.11 — Generator signal.

The voltage signal U created in a generator behaves like a sine function over time t. The general expression is: $U(t) = U_0 \sin(\omega t + \phi)$

U_0 is called the **amplitude** of the signal and basically scales the unit circle. The signal repeats itself after the time period T. The **angular frequency** ω is a measure for the speed of one rotation of the unit circle pointer and defined by $\omega = \frac{2\pi}{T}$. The 2π in the numerator indicates that the angular frequency has the units radians/time. ϕ is the **phase shift** of the signal.

Usually there are multiple lines realized in a generator. For example in a three phase generator the lines are physically $120°$ apart. As a consequence the voltage signals of the three lines are phase shifted by $120°$.

The inverse of trigonometric functions are called **arc functions** and for the example of the arc sine are written as: $sin^{-1}x$ or $\arcsin x$

12.5.3 Calculation rules

For the calculations with trigonometric functions there exist a few rules which are useful to be remembered.

Theorem 12.3 — Calculation rules.

Pythagorean identity:
$\sin^2 \alpha + \cos^2 \alpha = 1$

Angle transformations:
$\sin(\alpha_1 \pm \alpha_2) = \sin \alpha_1 \cos \alpha_2 \pm \sin \alpha_2 \cos \alpha_1$

$\cos(\alpha_1 \pm \alpha_2) = \cos \alpha_1 \cos \alpha_2 \mp \sin \alpha_1 \sin \alpha_2$

For calculations with arbitrary triangles these helpful rules exist:

$\frac{a}{\sin \alpha} = \frac{b}{\sin \beta} = \frac{c}{\sin \gamma}$ (Law of Sines)

$c^2 = a^2 + b^2 - 2ab \cos \gamma$ (Law of Cosines)

12.5.4 Series and trigonometric functions

As already indicated mathematical series play a major role in everything involving functions. Here we want to present two different series that are very important for many applications.

Taylor-Series

This series was already indicated in the section about polynomials. The **Taylor-Series** is a polynomial function of infinite order which can present any function. We will skip the details about the construction of the Taylor Series for now and directly present the Taylor-Series of the sine function in arithmetic terms via the following polynomial[1]:

[1] The ! indicates the factorial operator, i.e. $n! = 1 \cdot 2 \cdot \ldots \cdot n$

$$\sin(x) = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!} \tag{12.14}$$

Obviously it is clear that having infinite terms is very impractical. Therefore the Taylor Series is often truncated and written as a finite sum plus a remainder term. If the remainder term is sufficiently small the finite sum is an appropriate approximation of the function in the neighborhood of a reference point x_0. Obviously the more the polynomial is truncated the worse the approximation and the smaller the neighborhood where the Taylor polynomial is still a good approximation.

Thus for small x we may write: $\sin(x) \approx \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!}$

Notice that for very small x the $\sin$ function can be approximated by the identity function: $y = x$

Fourier-Series

Similarly to how the Taylor-Series is a special representation of any function this can be achieved by other mathematical constructs as well. In many situations especially when discussing arbitrary periodic functions other representations might be more useful to the description of the system behavior.

The **Fourier-Series** is such a mathematical construct which instead of polynomials (as in the Taylor series) uses the sine and cosine functions to describe a function in the general form:

$$f(x) = \tfrac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(\tfrac{2\pi}{T}nx) + \sum_{n=1}^{\infty} b_n \sin(\tfrac{2\pi}{T}nx)$$

The coefficients $a_0, \{a_n\}, \{b_n\}$ depend on the original function. We will show in a later session how they can be determined. In principle any function in a finite range can be expressed by the Fourier-Series by making T reasonably large enough.

The Fourier Series are very important for many fields of physics such as anything related to signals or filters, the description of waves (such as light) but were originally conceived to describe heat propagation in solids.

Part II

Programming

22 — Programming

Learning objectives

After processing this chapter you ...

- are able to use and create MATLAB functions
- are capable of creating scientifically valid graphs
- can explain the basic programming concepts of case differentiation and loops

The purpose of our last session was to obtain and install MATLAB, get an overview of its graphical user interface, do some first steps and understand how we can get script files going. Admittedly we have used so far MATLAB more as a fancy calculator rather than do any real programming with it.

This time we will cover actual programming so that at the end of the session you are in principle able to solve most programming tasks. On the way we will get to know some more of MATLAB's features that make our life a little more easy.

22.1 The very basics

In some of the exercises from the last session we have already introduced a few programming concepts without explicitly naming them. Here we want to repeat and give some more details since you have now acquired some experience.

Variables

It is fundamental to programming that we are able to store values in a memory location for later use. Instead of specifying an adress on a hard drive higher programming languages have introduced the concept of variables. Using variables the value of a specific memory location can be obtained by a name which is much easier to remember than a hard drive adress. The assignment of an adress to the variable name is done in the background and we can simply use the variable name.

There exist a few rules for variable names which you should consider (see also Ref. [**?**]). Variable names have to start with a letter and may be followed afterwards by other letters but also digits and underscores. Special characters like question marks and similar are not allowed. Notice that MATLAB is case sensitive meaning that it matters if a letter is written in lower or in upper case.

Furthermore the variable name may not be one of the very few reserved keywords of the programming language (e.g. `function`, `end`, `pi`, `null`, ...). Also make sure that your variable names are really unique and you do not use the same name twice by accident. This happens quite easily as for example 'i' is a very common name for a counter variable.

While you are free to name your variables however you want to, it is advisable to stick to some kind of coherent convention to make your code more easy to read (e.g. variable names always start with a lowercase letter).

In contrast to many other programming languages, in MATLAB we do not have to declare a variable and assign it a data type before it can be used. Instead we can just introduce a variable by assigning a value to it using the assignment operator = (the value may of course be the result of a calculation or statement).

⟨*variable name*⟩ = ⟨*value*⟩;

Notice that after this command is executed the variable appears in the workspace (even in the case the calculation result is not displayed in the command line). By clicking on the variable on the workspace you can check the type, dimension and value the variable currently holds. This is especially important if the program does not the intended results and you are looking for errors.

While this way of introducing a variable by assignment is very comfortable it has the downside that the type of a variable can change during the course of the program which can yield unexpected results. Therefore please make sure that your variable names are unique.

We have already used variables in the last session for example in Exercise 21.1: `b=4`

Functions

The main advantage of using MATLAB is that it provides a large library by means of functions such that the user does not have to program every tiny bit her/himself. You can recognize functions easily by a name immediately followed by parantheses which might contain a number of arguments.

`functionname()` or `functionname(arg1, arg2, ..., argn)`

Functions can be called by their name including the parantheses which (if applicable) contain values for the expected arguments. We have already used functions in the last session for example in Exercise 21.1: `sin(90)`

There we have also learned that it is necessary to consult the documentation to know what the function is doing exactly or what kind of arguments it expects. Here for example the function with the name `sin` expects that the argument is in radians and therefore does not interprete the argument 90 as degree.

The most important feature of functions is that they may return a value to the caller. Thus they can be used wherever a calculation, statement or value is expected as for example in the assignment as discussed above.

```
b=sin(90); %The variable b now holds the return value of the
   function sin() with argument 90
```

Functions may not necessarily need arguments. We also do not necessarily need to use the return value at any point in the program. It is possible that there exist different functions of the same name but with a different number of arguments. This concept is called *overriding* and MATLAB will automatically call the function with the fitting set of expected arguments.

Notice that operations like addition, multiplication and such in principle are also just functions. For example the addition takes two arguments $a, b \in \mathbb{R}$ and assigns a return value $c \in \mathbb{R}$. Thus in principle we would have to perform an addition as: `c=plus(a,b)`

However, as this function is performed so often MATLAB provides a special operator + which makes the call of this function much quicker and more intuitive. Later in this chapter you will get to know some more operators.

Consider you are now passing two arguments $a, b \in \mathbb{C}$ to the `plus` function. MATLAB will automatically detect this and choose the addition for complex numbers. This is an example for overriding functions. This concept extends to the respective operators as well.

At the end of this session we will discuss how to write and employ our own functions in MATLAB but for now we will just introduce a few of the functions MATLAB provides for us.

Function	Meaning
abs()	absolute value of a real number or magnitude of a complex number
ceil()	ceiling function: rounds up a floating-point number
floor()	rounds down a floating-point number
fix()	rounds a floating-point number towards zero
mod()	arithmetic remainder function
rem()	remainder after division
rat()	rational approximation of a floating-point number
round()	rounds to nearest integer
sign()	sign function
sqrt()	square root
log()	natural logarithm
log10()	logarithm to th ebase of 10
log2()	logarithm to the base of 2
exp()	Exponentiation with Euler number
interp1()	Linear interpolation
roots()	Roots of a polynomic function

Of course there are many other functions, please consult the large MATLAB documentation to identify and understand the functions you might need.

Exercise 22.1 — Reverse engineering.

Given is an array `X` of monotonous decreasing data. Use the MATLAB documentation to find out what the expression below is doing.

```
find([0 diff(sign(X))]~=0);
```

Make sure that you understand it, you will need to use it later.

22.2 Plotting

The result of Exercise 21.5 was a data table with coordinates of the sun at different points in time. While it is certainly nice to have this data table usually you would want to represent this kind of information visually as a graph. Thus we will discuss how to display graphs in MATLAB.

Plot types

Given the x-data in an array called `X` and the y data in an array called `Y` (with the same length as `X`) a standard plot of data in a cartesian xy-coordinate system is obtained by calling the function

```
plot(X,Y)
```

Upon execution of this function a new window pops up displaying the graph. This window is called in MATLAB a *figure*.

Often you might want to show two plots in the same graph simultaneously (see Fig. 22.1a). This can be achieved with the function:

```
plot(X,Y, X2, Y2)
```

The X data table does not need to be the same for both plots and can for example be of different length. Notice that here the concept of overriding functions applies as well.

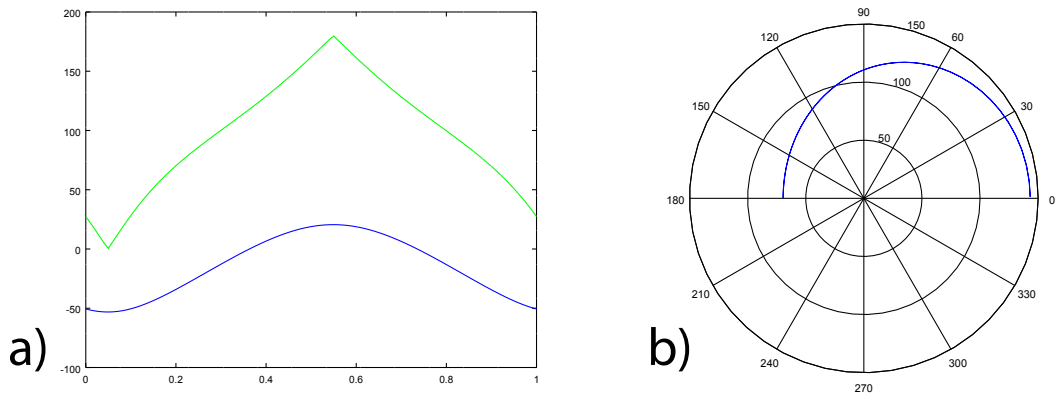

Figure 22.1: a) Simultaneous plot of two data sets in a single cartesian graph, b) Some data represented in polar coordinates (Source: own representation)

In the case your data is not represented well in a cartesian coordinate system and you rather want to display it in another coordinate system there exist many further functions:

Please check Ref. [**?**] to find the exact function for the plot you are interested in and the required options. Considering our example from Exercise 21.5 where the results are the elevation and azimuth angles we would rather display the data in a polar coordinate system (see Fig. 22.1b).

```
polar(X,Y)
```

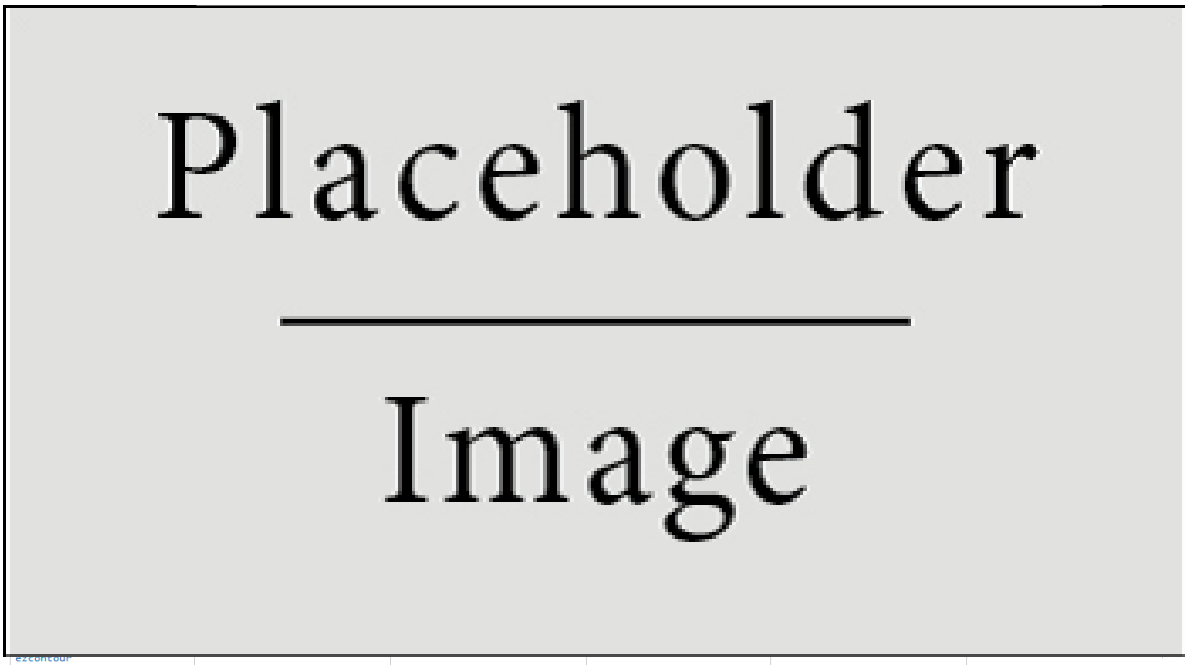

Figure 22.2: Overview of various plot types (Source: [**?**])

Plot options

The standard graphs obtained by these functions though are very minimalistic and do not satisfy scientific standards. Therefore we need to provide additional information in order to create proper graphs. Following the most common functions are given, for other functions please check the MATLAB documentation.

Add title

```
title('Graph of Sine and Cosine Between -2\pi and 2\pi')
```

Add axislabels

```
xlabel('-2\pi< x < 2\pi') % x-axislabel
ylabel('sine and cosine values') % y-axislabel
```

Add legend

```
legend('y = sin(x)','y = cos(x)')
```

Set graph display ranges

```
axis([minX,maxX,minY,maxY])
```

Plot formatting

You might not always be happy with the style of the plot MATLAB delivers to you. Every plotting function provides optional arguement to specify the line style. For example see Ref. [**?**] for details on the `plot` function.

The line style specification is given as an argument in single quotation marks to MATLAB and follows a code explained in the respective function documentation.

For example to create a dotted red line with open circles as data point markers the following code can be employed.

```
plot(X,Y,':or')
```

Multiple graphs

Notice that with all of the previous graphing functions we have never done anything with the return values, in fact it might not even be clear what kind of entity this return value might be at all. When assigning the return value of a plotting function to a variable it will hold a so called *handle* to the graph. Using this handle we can refer to the graph later which is useful when we are dealing with many graphs at once.

Without using the handles all the graphing functions will be employed on the latest figure. That includes the plotting functions meaning that when repeatedly calling a plotting function the figure will only show the latest result overwriting all of the plots before. In the case you want a new figure window to pop up use the command

```
figure
```

Now all subsequent plots will be displayed in this new window.

Sometimes you might simply want to add a plot to a previously existing graph. Here the command

```
hold
```

might prove useful which prevents the graph from being overwritten. Notice however, that in contrast to the alternative `plot(X1, Y1, X2, Y2)` the color of the newly added plots does not change automatically and you might want to assign a contrasting format yourself.

Example 22.1 — Sun Position over day with plot.

In the following example you can find a sample solution to Exercise 21.5 enhanced by a polar representation of the calculated data.

sim07_sunposition_plot.m

```
%Initial Parameters
DayOfYear=309; %Nov 5th in the year 2015
TimeZone=+2; %Difference to Greenwich Mean Time GMT
SummerTime=0; %1=TRUE, 0=FALSE
Longitude=8.213889;  %Oldenburg (in degrees)
Latitude=53.143889; %Oldenburg (in degrees)
EarthTilt=23.45; %degrees

TimeArray=0:0.01:1; %time as fraction of day
LT=TimeArray*24; %Local time, in hours
LSTM=15*(TimeZone+SummerTime);
B=(360/365)*(DayOfYear-81); %Position on the orbit
EoT = 9.87*sind(2*B)-7.53*cosd(B)-1.5*sind(B); %minutes
TC=4*(Longitude-LSTM)+EoT; %in minutes
LST=LT+TC/60;
HRA=15*(LST-12); %Hour angle, 0 at noon

Declination = asind(sind(EarthTilt)*sind(B));
Elevation = asind(sind(Declination)*sind(Latitude)+cosd(
   Declination)*cosd(Latitude)*cosd(HRA));
Azimuth=acosd((sind(Declination)*cosd(Latitude)-cosd(
   Declination)*sind(Latitude)*cosd(HRA))./cosd(Elevation)); %
   here elementwise division has to be added

plot(TimeArray, Elevation)
hold
plot(TimeArray, Azimuth, "g")

figure
polar(deg2rad(Azimuth), (90-Elevation))
```

Practice your plotting ability yourself

Exercise 22.2 — $I(U)$ characteristic of a photovoltaic device & load.

Plot the current-voltage characteristic of a photovoltaic device and an arbitrary load in the same graph (current I, voltage U).

The $I(U)$ behaviour of the load is given by Ohm's law: $I = U/R$ (choose a suitable R).

The $I(U)$ behaviour of the photovoltaic device is modelled using the Schockley equation:

$$I = I_{ph} - I_0 \left(e^{\frac{qU}{nkT}} - 1 \right)$$

The photocurrent I_{ph} is proportional to the irradiation intensity G according to:

$I_{ph} = G \times 3.4 \times 10^{-3} Am^2/W$

As for the remaining parameters use:

$$\begin{aligned} I_0 &= 100\ pA \\ k/q &= 8.617 \times 10^{-5}\ V/K \\ n &= 0.9 \\ T &= 300\ K \\ P &= 1000\ W/m^2 \end{aligned}$$

22.3 Case differentiation

The problem

Let us return to the example of the sun's position for a moment. When running Example 22.1 we obtain the two plots shown in Fig. 22.1. In Fig. 22.1a the blue line illustrates the elevation of the sun which seems reasonable, yet, the green line representing the azimuth seems quite strange.

According to Ref. [**?**] the azimuth is defined with $\psi = 0$ when the sun is due North increasing into the direction of the East ($\psi = 90$), passing the South ($\psi = 180$) and returning via the West ($\psi = 270$). Thus we would expect the azimuth to be monotonously increasing over time. Instead our plot rather suggests that the sun suddenly changes its direction. This is obviously not plausible, so what happened?

We obtained the formula $\psi(\delta, \varphi, \alpha, HRA)$ for the azimuth from Ref. [**?**]. Yet, at the very bottom of Ref. [**?**] it is explained that this equation is in fact only correct in the morning and that it has to be modified under certain conditions.

$$Azimuth = \psi(\delta, \varphi, \alpha, HRA) \qquad \text{, for } LST < 12 \text{ or } HRA < 0$$
$$Azimuth = 360° - \psi(\delta, \varphi, \alpha, HRA) \qquad \text{, for } LST > 12 \text{ or } HRA > 0$$

In fact this would only be correct for a certain day in solar time $0 < LST < 24$. As we want to observe the azimuth on a local day we have to add another case:

$$Azimuth = -\psi(\delta, \varphi, \alpha, HRA) \qquad \text{, for } LST < 0 \text{ and } HRA < 0$$

Thus we conclude that we need to treat different parts of the calculation in distinct ways in order to obtain plausible results. The different calculations are called *cases* and are executed when certain *conditions* are fulfilled which have to be checked beforehand. These conditions evaluate either to TRUE or FALSE and underly the so called Boolean logic.

Boolean logic:

In boolean logic a variable can only have one of two possible values which in MATLAB have the keywords `true` or `false`.

```
a=true;
b=false;
```

Just as with numbers we are able to calculate with logical values (refer to the math course for more details). Table 22.1 gives the operators for the most important logical operators. As mentioned before an operation can be considerd a function with two arguments and only for the most common operations a special operator is provided. Less common logical operations might not have a special operator and can alternatively be performed by calling the corresponding function.

Operation	Operator	Function
AND	`a & b`	`and(a,b)`
OR	`a | b`	`or(a,b)`
NOT	`~a`	`not(a)`
XOR		`xor(a,b)`

Table 22.1: Logical operators

Notice, that these operators may also be applied on arrays and matrices.

Conditional expressions:

In order to decide if a condition is fulfilled or not we require a logical value. This value can be given directly as a keyword or by evaluating the value of a variable.

Most often we use compare two values to decide on how to continue the calculations. These comparisons evaluate to a boolean value and can therefore also be used in conditional expressions.

Comparisons are functions on ordered sets M: $M^2 \rightarrow \{\texttt{true}, \texttt{false}\}$ with $a, b \in M$. Comparisons are also so common that MATLAB provides special comparators (see Table 22.2) to make coding faster and more intuitive.

Operator	Meaning
`a == b`	is equal to
`a ~= b`	is NOT equal to
`a < b`	is less than
`a <= b`	is less or equal than
`a > b`	is greater than
`a >= b`	is greater or equal than

Table 22.2: Logical comparators

Notice that these expressions are only evaluated when in parantheses: `c=(1<0);`

MATLAB provides additional functions for more complicated checks which return a logical value. Many of them start with 'is':

Example:

```
isinf(1/0)  %checks whether a value is infinity or not
```

If-Else structures

In order to execute code only in the case a certain condition is fulfilled every programming language provides a special case differentiation structure. In MATLAB (and most of the other programming languages) it is called an `if`-structure and written as:

```
if(⟨check condition⟩)
  ⟨do something⟩ %execute if check condition evaluates to TRUE
end
```

Sometimes it is useful to represent the flow of your program graphically in so called flow charts. See Fig. 22.3a for such a diagram of the `if` structure to understand visually what is happening.

The program runs into the `if` statement. In the case the statement in the parantheses evaluates to the logical value `true` the statements below until the keyword `end` are executed. If the condition evaluates to `false` the statements are skipped. The statements after the keyword `end` are then again executed for both cases.

The check condition may be a logical value, the return value of a function, a comparison operation or a logical combination of all of the mentioned:

Example: `if(a>11.1 & a<14.4)`

While this `if` structure is theoretically sufficient for all our programming needs often we will be in a situation where we do not want to simply continue the program after the `end` in the case the check condition evaluates to `false`. Instead we would like to execute alternative statements as illustrated in the flow chart Fig. 22.3b before the two program streams combine after the `end` keyword.

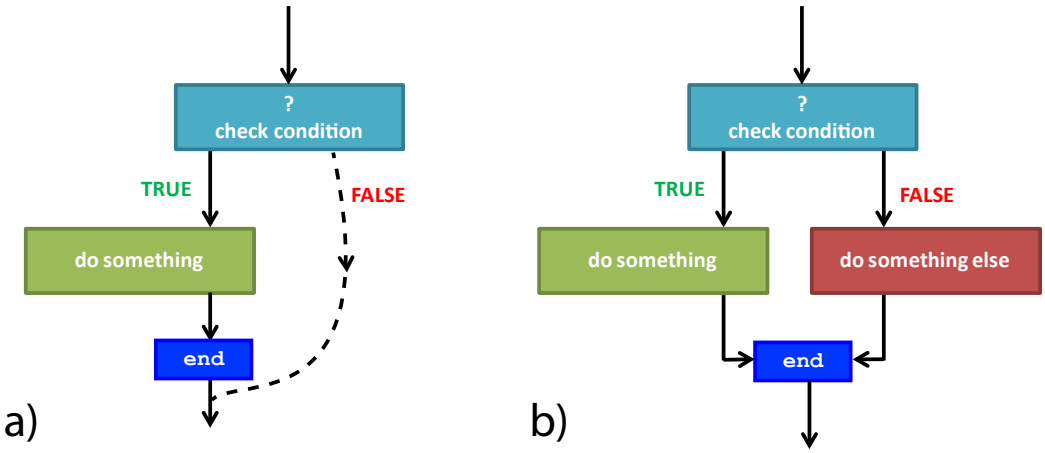

Figure 22.3: Flow chart of a) `if` structure, b) `if-else` structure (Source: own representation)

For this situation most programming languages provide the more intuitive `if-else` structure written in MATLAB as:

```
if(⟨check condition⟩)
  ⟨do something⟩ %execute only if check condition evaluates to TRUE
else
  ⟨do something else⟩ %execute only if check condition evaluates to FALSE
end
```

Often it is necessary to refine the case differentiation within one (or both) of the `true`/`false` streams. Therefore it is possible to nest `if-else` statements:

```
if(⟨check condition 1⟩ )
  ⟨do something⟩
  if(⟨check condition 2⟩ )
    ⟨do something more⟩
  end
else
  ⟨do something else⟩
end
```

While not necessary it is highly recommended that the nested statements are intended for better readability.

When the logical operators given in Table 22.1 are used in `if` statements (or loops) these operators will feature by default a so called *short circuit* behaviour. This means that when the logical expressions are evaluated and for instance in the case of the AND the first condition is already `false` then it is already clear that the full statement will be `false` and the second expression is not even evaluated anymore. This is implemented by default as it makes the execution of the program faster.

Sometimes you might want to use this feature outside of `if` statements or loops. The respective operator for this is `&&` for the short circuit AND and `||` for the short circuit OR. It is recommended that in order to be aware of the short circuit behaviour in `if` statements and loops you always explicitly use the short circuit operators.

22.4 Loops

Often it is necessary to repeat a certain fraction of the programming code multiple times before continuing. One example might be counting through all the elements of a vector.

`while` - Loops

This is accomplished via loops which in principle work as illustrated in the flow chart of Fig. 22.4 and are implemented in MATLAB via:

```
while(⟨check condition⟩) %loop opening
  ⟨do something⟩ %statements to perform
end
```

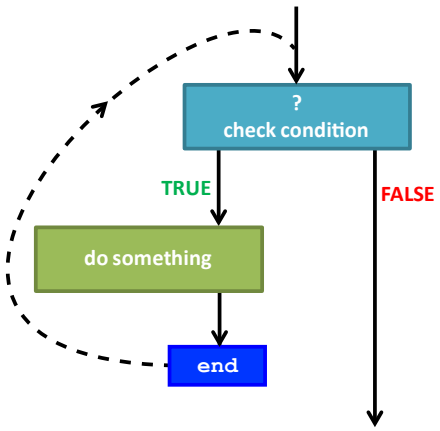

Figure 22.4: Flow chart of a while loop (Source: own representation)

The program runs into a loop opening statement (`while`) which similarly to the `if` statement checks if a certain (loop continue) condition is fulfilled. In the case the condition evaluates to `true` a number of statements is performed until the loop is closed with the keyword `end`. Instead of continuing with the program as with the `if` structure the program now automatically returns just before the loop opening statement to check the loop continue condition again. In the case the condition evaluates to `false` the loop stops and the program is continued below the end keyword.

This has the consequence that somewhere within the loop statements the condition which is checked has to be modified in such a way that after some time it evaluates to `false` otherwise the loop will never be stopped and the program be caught in an infinite loop. This is to be avoided as it can crash your computer.

The execution of statements within one repetition of the loop can be prematurely aborted with the keyword `continue`. When the program encounters this keyword the program jumps immediately to the position just before the loop opening to check the loop condition again.

The discussed loop structure is called a `while` loop and in principle is sufficient for all programming tasks. Yet, similar to before there exist certain variations to loops which occur so frequently that there exist special keywords which are more intuitive and more easy to program.

Similar to `if` statements loops can be nested which is for example necessary to process data tables of dimension larger than 1 (e.g. matrices).

`for` - Loops

While loops are very practical especially when it is not known how often exactly the loop has to be repeated. On the other hand when dealing with vectors and matrices we usually know exactly how often we have to perform the loop.

In this case we can use a counter variable which is incremented or decremented in each step by a fixed amount until the number of repetitions is achieved. Because programmers are lazy they have introduced a shortened form for this special loop scenario called `for`-loops which are implemented in MATLAB with:

```
for ⟨index⟩=⟨initial value⟩:⟨increment⟩:⟨finalvalue⟩
  ⟨do something⟩
end
```

The following example compares the `while` to the `for` loop implementation:

```
i=1;
while(i<10)
  ⟨do something⟩
  i=i+0.5;
end
```

```
for i=1:0.5:10
  ⟨do something⟩
end
```

In fact MATLAB automatically does something like this when calculating with arrays & matrices.

Example 22.2 — Sun Position over day with improved plot.

Let's employ loops and case differentiation to solve our problem with the sun position plot.

After the Elevation and Azimuth have been calculated we loop through each point in time (using a for loop). Within each loop repetition we check the conditions of HRA and LST and determine on a calculation method as specified above. The result is displayed in Fig. 22.5

```
...
Declination = asind(sind(EarthTilt)*sind(B));
Elevation = asind(sind(Declination)*sind(Latitude)+cosd(
   Declination)*cosd(Latitude)*cosd(HRA));
Azimuth = acosd((sind(Declination)*cosd(Latitude)-cosd(
   Declination)*sind(Latitude)*cosd(HRA))./cosd(Elevation));

%% Correcting the plot by going through the array element-wise
    and adjusting the values based on certain conditions
for i=1:length(TimeArray)
    if ((HRA(i) < 0) | (LST(i) < 12))
        Azimuth(i)=Azimuth(i);
    else
        Azimuth(i)=360-Azimuth(i);
    end

    if ((HRA(i) < 0) & (LST(i) < 0))
         Azimuth(i)=-Azimuth(i);
    end
end

% Plot Azimuth and Elevation in same figure
plot(TimeArray, Elevation, TimeArray, Azimuth)

figure
polar(deg2rad(Azimuth), (90-Elevation));
set (gca, 'color', 'none'); %set transparent background
view([-90 -90]);
```

It's now again your turn:

Exercise 22.3 — Winterlab Solar Cell Simulation.

Plot the $I(U)$ characteristic of the photovoltaic device from Exercise 22.2 for different illumination intensities G and temperatures T in two separate figures.

For each figure include the plots for the following measurement conditions:

1. Illumination: $0, 200, 400, 600, 800, 1000\ W/m^2$

2. Temperature: $20, 30, 40, 50, 60°\mathrm{C}$

Calculate the efficiency $\eta = \frac{P_{mpp}}{G}$ for each measurement series and plot it versus the varying parameter where P_{mpp} is the maximum power point for each paricular measurement.

Remember that Power=Voltage $\times$ Current and that there exists a function in MATLAB to find the maximum in an array.

HINT: Let's ignore uncertainties due to simulation resolution for the moment.

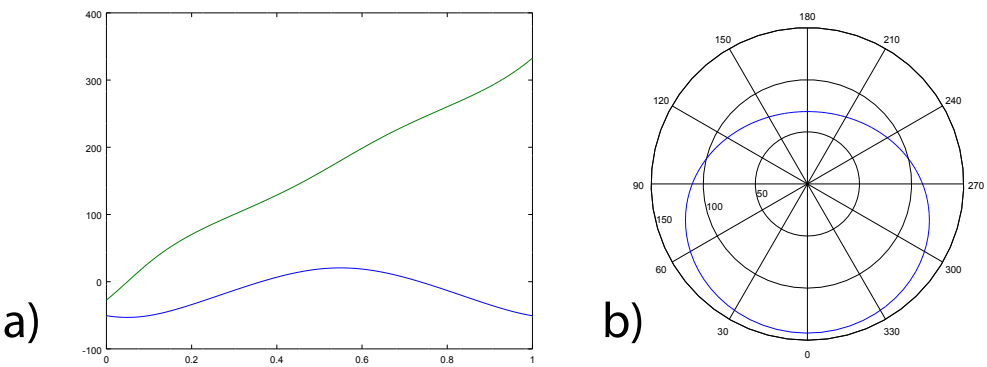

Figure 22.5: a) The azimuth is now a monotonous increasing function as expected, b) Now the sun comes full 'circle' in the polar plot as well (Source: own representation)

22.5 Function files

Now that we are acquainted with the fundamental programming structures of case differentiation and loops in principle we can accomplish any programming task. Thus we are ready to write our own functions.

In the case you are wondering why it is necessary to write functions at all consider the following.

- In the case certain pieces of code have to be executed multiple times at different parts of your program (and loops are not suited) you can reuse the code easily.
- Furthermore functions hide large sections of your code from view making code management and navigation more easy.
- Third you can modularize your code into functional units where each module acts like a black box. It does not matter how exactly the function is implemented only that it does its job. Thereby you can share code development onto different people more easily.

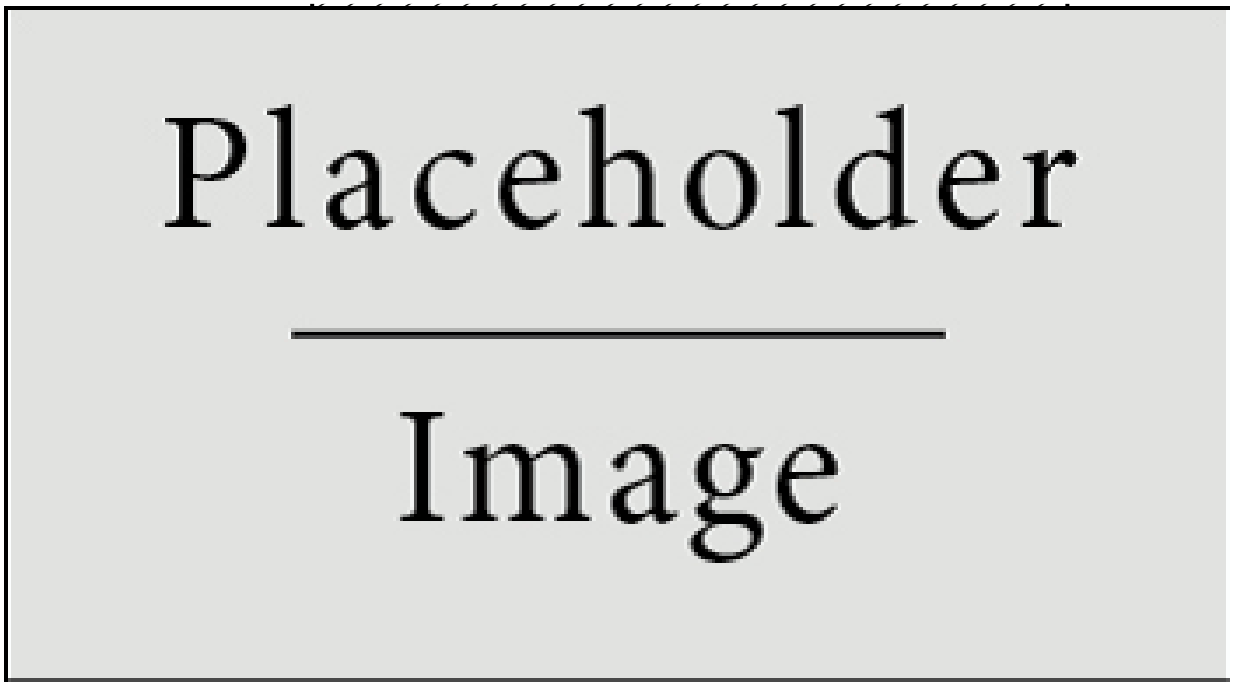

Figure 22.6: Functions as black boxes which can receive input, process it somehow and provide output (Source: own representation)

In MATLAB each function has to be implemented in its own separate .m file which is quite unusual in the world of programming languages. To differientiate function files from script files notice that script files cannot receive input arguments or provide output values to other parts of the program.

MATLAB requires that the **first** non-comment line of this function file has a very specific header of the style:

```
function [return1 return2] = functionname(param1, param2)
```

Via the keyword `function` MATLAB knows that the file is not merely a script file.

Then the return value is named; in the case you want to return multiple values this can be accomplished by returning an array of return values (mind the brackets). Return values need to be assigned a value with the exact same name within the code. No special return statements are necessary.

The function name has to obey the same naming rules as variables and has to have the exact same name as the file.

In the parantheses a list of expected arguments is given. The names of these arguments automatically become variables which you may access from within the code. The term argument and parameter can be used interchangably. It is not required that a function expects any arguments.

Return values and parameters may have any kind of data type and do not necessarily have to be all of the same type.

Notice that you cannot execute a function on its own as it is missing values for the arguments. Therefore you need to call the function from a script or the command line to test its functionality (e.g: `functionname(2,1);`). Obviously the function file needs to be visible to MATLAB just as discussed in the previous session for script files.

Use breakpoints to check the state of your workspace at different points in your function to investigate the performance during run-time.

You may introduce variables within your function which are not later returned. These variables are in general not valid outside the scope of this function meaning that they cannot be accessed from a calling script. Similarly you cannot access variables from the calling script other than the passed arguments. The arguments and the return values therefore form the interface of the function to the outside world. This is the reason the first line is so important.

Consider the following examples illustrating functions with different numbers of arguments and return values.

Example 22.3 — One input, one output.

In this example we modify the airmass script from Example 21.1 into a function that returns the radiation intensity dependent of the incident angle.

The radiation intensity calculated with the airmass value is given according to Ref [**?**] by the empirical formula $G = AM0 \times 1.1 \times 0.7^{AM^{0.678}}$ with the solar constant $AM0$ representing the intensity of the solar radiation just outside of the atmosphere.

sim13_airmass_power.m

```
function G = sim13_airmass_power(AngleFromZenith)

AM0=1353; %kW/m2 solar constant
AM=1/cosd(AngleFromZenith);
G=AM0*1.1*0.7^(AM^0.678);
```

Example 22.4 — Multiple inputs, one output.

This example calculates a factor which gives the reduction of the radiation density on a module in the case it is not facing the sun directly. This purely geometric consideration results in the equation found on Ref. [**?**]:

sim14_tilt.m

```
function tiltfactor = sim14_tilt(Elevation, Azimuth,
   ModuleTilt, ModuleAzimuth)

tiltfactor = cosd(Elevation)*sind(ModuleTilt)*cosd(
   ModuleAzimuth-Azimuth)+sind(Elevation)*cosd(ModuleTilt);
```

Example 22.5 — Multiple inputs and outputs.

Let's transform the script to calculate the sun position at a specific point in time into a function. Notice that most of the initial parameters are now arguments to be passed to the function.

sim15_sunpos.m

```
function [Elevation, Azimuth] = sim15_sunpos(DayOfYear,
   TimeZone, SummerTime, LocalTime, Longitude, Latitude)

%Initial Parameters
EarthTilt=23.45; %degrees

LT=LocalTime*24; %Local time, in hours
LSTM=15*(TimeZone+SummerTime);
B=(360/365)*(DayOfYear-81); %Position on the orbit
EoT = 9.87*sind(2*B)-7.53*cosd(B)-1.5*sind(B); %minutes
TC=4*(Longitude-LSTM)+EoT; %in minutes
LST=LT+TC/60;
HRA=15*(LST-12); %Hour angle, 0 at noon

Declination = asind(sind(EarthTilt)*sind(B));
Elevation = asind(sind(Declination)*sind(Latitude)+cosd(
   Declination)*cosd(Latitude)*cosd(HRA));
Azimuth=acosd(((sind(Declination)*cosd(Latitude)-cosd(
   Declination)*sind(Latitude)*cosd(HRA))/cosd(Elevation)));
... %case differentiation
```

Exercise 22.4 — Solar cell $I(U)$ analyzer.

Write a function which takes an $I(U)$ characteristic of a solar cell as argument and returns a vector of the following characteristic parameters:

- maximum Power P_{mpp} (remember that Power = Voltage x Current)
- Voltage at maximum power point U_{mpp}
- Current at maximum power point I_{mpp}
- Efficiency $\eta = \frac{P_{mpp}}{G}$
- Short circuit current $I_{sc} = I(0)$ (equal to the photocurrent)
- Open circuit voltage U_{oc} (remember: $0 = I(U_{oc})$)
- Fill factor= $\frac{P_{mpp}}{I_{sc} \times U_{oc}}$

Advanced option:

Use linear interpolation to get more exact values for the maximum power point and open circuit voltage! Remember that the maximum of a function is the x-intercept of its derivative. The derivative can be calculated easily using: $\frac{\Delta I}{\Delta U}$

HINT: Employ the function from Exercise 22.1 in this task.

Part III

Solid State Physics

32 — Semiconductors

After processing this chapter you will

- be able to calculate charge carrier densities
- understand the law of mass action
- be able to calculate drift and diffusion currents
- understand how a light bias or voltage bias influences the semiconductor properties

In solar cells we want to generate an electrical current. Therefore it is necessary to discuss the electrical conductivity in semiconductors in more detail. We will see that the density of states and the Fermi distribution which we have introduced in chapter 31 will help us in this. In order to enhance the electrical properties of semiconductors it is common to introduce additional charge carriers in a process called doping. At the end of this chapter we discuss the electrical behavior of semiconductors under various environmental conditions.

32.1 Conductivity

The conductivity σ is a measure for the ability of a material to conduct a current J in a given electric field F.

$$\sigma = \frac{J}{F} \tag{32.1}$$

The electron conductivity σ_n depends on the electron density n and the electron mobility μ_n via

$$\sigma_n = qn\mu_n \tag{32.2}$$

where q is the elementary charge of the electron. In semiconductors we can formulate an analogous expression for the hole conductivity using the hole density p such that the general expression for the conductivity is:

$$\sigma = q\mu_n n + q\mu_p p \tag{32.3}$$

The mobilities can be expressed in terms of diffusion constants D_n and D_p according to the Einstein relations:

$$\mu_n = \frac{qD_n}{kT} \tag{32.4}$$

$$\mu_p = \frac{qD_p}{kT} \tag{32.5}$$

So in order to describe the conductivity of a semiconductor we require the charge carrier densities n for electrons and p for holes.

32.2 Charge carrier densities

In chapter 31 we have learned about energy states in solids and derived an analytical formula which describes the density of states with respect to energy. Furthermore we have described the hole and electron distribution with the Fermi-Dirac statistics. However, in order to describe the conductivity we need to know the charge carrier densities, i.e. how many charge carriers contribute to the current.

The density of states describes just how many states there are (per energy interval dE) which potentially can be occupied by a charge carrier. The Fermi-Dirac distribution just describes the probability by which an electron has a certain energy. Yet, by multiplying the density of states with the probability distribution we get the number of charge carriers per energy interval dE. In order to obtain the total number of charge carriers (i.e. the charge carrier density) in a band we integrate over the width of the energy band. This can be expressed mathematically by:

$$n = \int_{E_c}^{\infty} g_c(E) f_0(E, E_F, T) dE \tag{32.6}$$

$$p = \int_{-\infty}^{E_v} g_v(E)(1 - f_0(E, E_F, T)) dE \tag{32.7}$$

Note that as holes are the absence of electrons they are described by $1 - f_0(E, E_F, T)$ instead of the regular Fermi distribution. Figure 32.1a illustrates this concept.

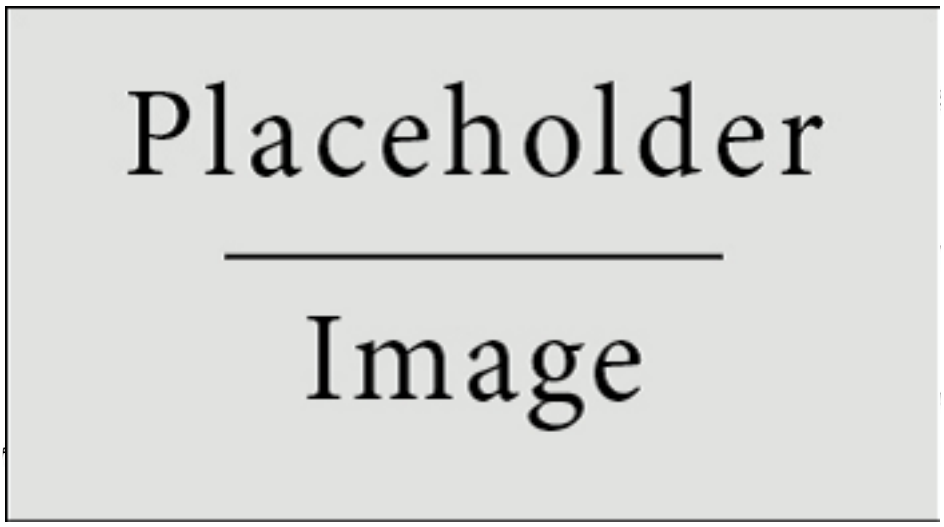

Figure 32.1: a) Calculation of the carrier densities from the density of states and the Fermi distribution. b) In n-type semiconductors the Fermi level shifts closer to the conduction band resulting in $n > p$. (Source: modified from [**?**])

When the temperature increases the 'step' in the Fermi distribution becomes wider. This means that more charge carriers can find a place in the conduction band effectively increasing the charge carrier density. More charge carriers means better conductivity which explains a characteristic property of semiconductors that they are better conductors at larger temperatures (compare this with metals where the conductivity decreases due to an increasing ohmic resistance).

32.3 Law of mass action

In order to describe the characteristics of the semiconductor analytically using the Fermi-Dirac distribution is quite cumbersome. Assuming that the Fermi level is far from both band edges we can simplify equation 31.13 with the Boltzmann-approximation. Considering the conduction band and $E \gg E_F$ the electon probability distribution simplifies to

$$f_0(E, E_F, T) \approx e^{E_F - E/kT} \tag{32.8}$$

and correspondingly for the valence band ($E \ll E_F$):

$$1 - f_0(E, E_F, T) \approx e^{E - E_F/kT} \tag{32.9}$$

Using these simplified expressions the integrals in Eqs. 32.6 and 32.7 can actually be evaluated analytically:

$$n = N_c e^{E_F - E_c/kT} \tag{32.10}$$

$$p = N_v e^{E_v - E_F/kT} \tag{32.11}$$

The coefficents N_c, N_p are called the effective conduction (resp. valence) band density of states and are given by these expressions:

$$N_c = 2(\frac{m_c^* kT}{2\pi\hbar^2})^{3/2} \tag{32.12}$$

$$N_v = 2(\frac{m_v^* kT}{2\pi\hbar^2})^{3/2} \tag{32.13}$$

Compare the density of states calculated in chapter 31 (Eqs. 31.26 and 31.27) with these effective densities of states. Note that Eqs. 32.12 and 32.13 are not dependent on the energy anymore.

Note the particular characteristic that the product of electron and hole densities is no longer dependent on the Fermi level but only on the energy band gap:

$$np = N_c N_v e^{-E_g/kT} \tag{32.14}$$

This is called the law of mass action. When no extra charge carriers are introduced the electron and hole densities are always the same as each hole is generated by the excitation of one electron (intrinsic semiconductor). Thus we can define the intrinsic carrier density n_i:

$$n_i^2 = np = N_c N_v e^{-E_g/kT} \tag{32.15}$$

The intrinsic carrier density is a function only of the semiconductor band gap and the temperature as such n_i is a characteristic material parameter.

Exercise 32.1 — Power conversion efficiency. Consider the two spectra (a) and (b) shown as function of photon energy in Fig 32.2. If you had a solar cell made from a semiconductor material with a band gap at E_g (black line) for which condition would you expect the higher power conversion efficiency of the device: spectrum (a) or spectrum (a+b)? Explain your answer.

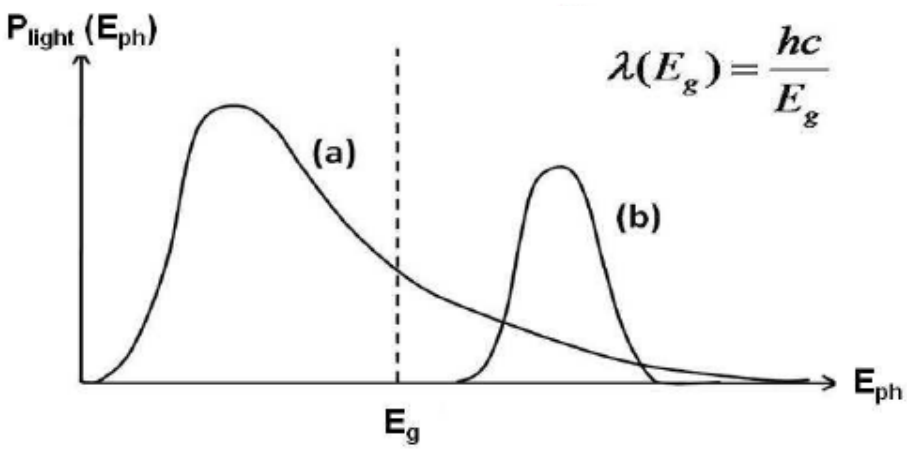

Figure 32.2: Spectra used for Exercise 32.1

Exercise 32.2 The expressions for the carrier densities can alternatively expressed using the intrinsic carrier density:

$$n = n_i e^{(E_F - E_i)/kT} \tag{32.16}$$

$$p = n_i e^{(E_i - E_F)/kT} \tag{32.17}$$

In these expressions we have introduced the intrinsic potential energy E_i.

1. Find an expression for E_i in terms of the effective masses of the charge carrier densities.
2. Where lies E_i with respect to the Fermi level in intrinsic semiconductors?
3. What does this teach you about the position of the Fermi level in intrinsic semiconductors?

32.4 Doping

In order to improve the conductivity of semiconductors impurity atoms are implanted in a controlled manner. These impurities alter local bonds (see Fig. 32.3) and thus the density of states. If a specific impurity creates an energy state in the bandgap above the intrinsic potential energy the Fermi level shift closer towards the conduction band (see Fig. 32.4b). Similarly for the opposite case when the introduced energy state is below E_i and the Fermi level shifts closer to the valence band. This process is called **doping** and we distinguish between n-type doping ($E_F > E_i$) and p-type doping ($E_F < E_i$). The doped semiconductor is now called an extrinsic semiconductor.

Figure 32.3a illustrates a pure intrinsic semiconductor where all electron bonds are saturated. The corresponding band diagram is shown in Fig. 32.4a with the Fermi level around the center of the energy gap.

In Fig. 32.3b we replace one atom of the bulk with an extrinsic atom which possesses an additional electron which is not incorporated in any of the bonds. Because the new atom donates an additional electron it is called donor. This type of doping is called n-type as the additional charge carrier is negative. The additional electron creates an additional energy state close to the conduction band and thereby raises the Fermi level (see Fig. 32.4b).

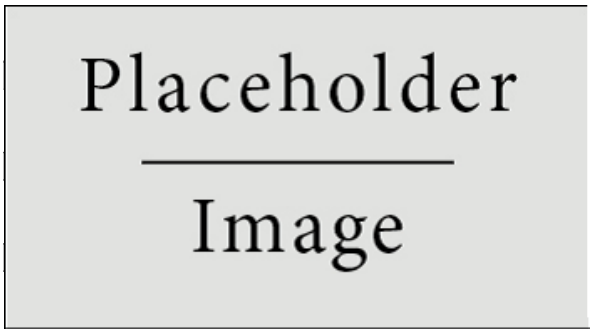

Figure 32.3: a) Intrisic Si lattice b) Incorporation of a donor atom adding one electron c) Incorporation of an acceptor atom removing an electron from another bond (Source: modified from [**?**])

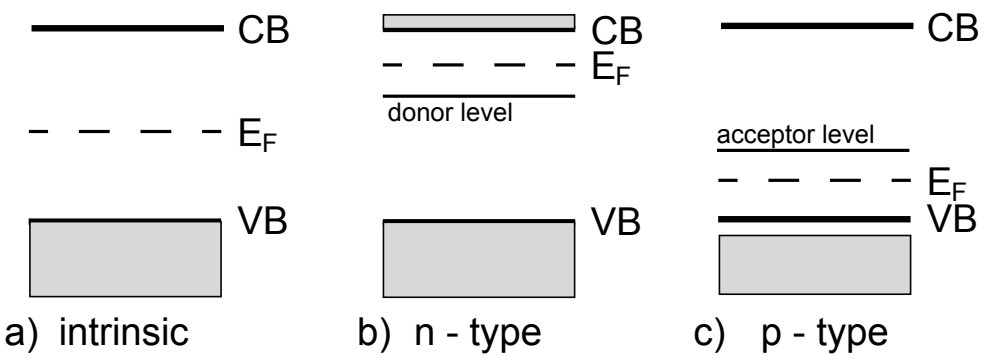

Figure 32.4: Shift of the Fermi level from the a) intrinsic level by b) n-type and c) p-type doping. In b) the free electrons form a high energy band while all the electron states in the low energy band are full. In c) there are no free electrons in the conduction band while the holes create free electron states in the valence band.

From this donor state the electron is easily excited (donor is ionised) such that at room temperature it can be assumed that all electrons from the donor states are in the conduction band. The contribution of the electrons from the donor states dominates and increases the conductivity significantly. Thus the electron density can be approximated by the dopant density $N_d \gg n_i$.

$$n \approx N_d \tag{32.18}$$

When the concentration of one carrier type is increased by doping, the concentration of the other species decreases, i.e. n_i remains constant. That means that the law of mass action $n_i^2 = np$ still holds in doped semiconductors:

$$p = n_i^2/N_d \tag{32.19}$$

Consequently the electron density is increased compared to the hole density: $n > p$ (see Fig. 32.1b). As the conduction in n-type semiconductors is dominated by electrons they are called majority carriers and the holes minority carriers.

Figure 32.3c illustrates the opposite case. Here an atom is introduced which lacks one electron to saturate all bonds. As this dopant accepts an additional electron it is called an acceptor. In the hole picture the acceptor introduces a free hole into the material which is positively charged. Therefore this type of doping is called p-type. The introduced energy state is close to the valence band decreasing the Fermi level (see Fig. 32.4c). At room temperature all acceptor states are filled with electrons from the valence band. The increased density of holes in the valence band can be approximated by the density of acceptor dopants $N_a \gg n_i$:

$$p \approx N_a \tag{32.20}$$

$$n = n_i^2/N_a \tag{32.21}$$

In the case of p-type doping the holes are the majority carriers and the electrons the minority carriers.

Exercise 32.3 — Intrinsic carrier density. Germanium has an effective density of states (DOS) $N_C = 10^{19}\mathrm{cm}^{-3}$ for the conduction band and a band gap E_g = 0.66 eV. The intrinsic carrier density at 300 K is $1.8 \times 10^{13}\mathrm{cm}^{-3}$.

1. What is the effective DOS for the valence band, N_V ?
2. If the material is n-doped to give an electron density of $n = 10^{18}\mathrm{cm}^{-3}$, what is the hole density?
3. What is the intrinsic carrier density at 100 K? (You can assume that the effective DOS do not change with temperature.)

Exercise 32.4 — Fermi level in doped semiconductors. Crystalline silicon has effective DOS of $N_V = 3 \times 10^{19}\mathrm{cm}^{-3}$ for the conduction band and $N_C = 3 \times 10^{19}\mathrm{cm}^{-3}$ for the valence band, and a band gap $E_g = 1.1$ eV. A silicon crystal is doped with $10^{17}\mathrm{cm}^{-3}$ boron (B) atoms.

1. What is the position of the Fermi-level relative to the valence band maximum, E_V, and conduction band maximum, E_C, at 300 K?
2. If the acceptor state energy, E_D, is 0.05 eV above the valence band maximum use the Fermi-Dirac distribution and the Fermi level calculated in the previous task to calculate the fraction of dopant atoms that are ionized.
3. Using the same approximations, calculate the Fermi-level and fraction of ionized dopants at 77 K. Is the assumption of complete ionization still valid?

32.5 Bias, quasi fermi levels

Until now we have considered the semiconductor to be in equilibrium. For our purposes the two most important sources disturbing the equilibrium are:

1. Application of electrical bias resulting in injection of charge carriers
2. Illlumination resulting in great changes of charge carrier distributions due to photoexcitation

In the above situations *both* carrier concentrations (i.e. the charge carrier densities) are increased above the equillibrium level n_0 and p_0. If we wanted to describe this with the Fermi distribution it should shift both closer to the conduction and closer to the valence band. This is a contradiction and therefore the Fermi-Dirac statistics and thus the assumption $n_i^2 = np$ are no longer valid. As the Fermi-Dirac statistics no longer holds we also lose the advantage of using the Boltzmann approximation to describe the semiconductor and thereby our ability to describe the semiconductor analytically. Thus, as often in physics, we assume that the disturbance is not too large and occurs on a large timescale (compared to relaxation processes within the bands).

Due to this assumption the electron and hole densities each relax into their own separate quasi thermal equilibriums each of which can be described using the Fermi-Dirac statistics. In other words, the electrons of the conduction band are distributed according to a distribution function $f_c(k, r)$ with the electron Fermi-level E_{F_n} and temperature T_n while the holes in the valence band are distributed according to a distribution function $f_v(k, r)$ with a hole Fermi level E_{F_p} and temperature T_p:

$$f_c(k, r) \approx f_0(E, E_{F_n}, T_n) \tag{32.22}$$

$$f_v(k, r) \approx f_0(E, E_{F_p}, T_p) \tag{32.23}$$

E_{F_n} and E_{F_p} are called quasi Fermi levels. Generally the quasi Fermi levels are split ($E_{F_n} \neq E_{F_p}$) (see Fig. 32.5).

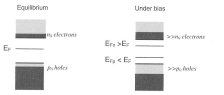

Figure 32.5: Upon disturbing the equilibrium the different charge carriers densities relax to separate quasi equilibrium distributions which can be seen as a splitting of quasi Fermi levels (Source: [**?**])

With these expressions for the respective charge carrier distribution functions we find the analytical solutions for the charge carrier densities:

$$n = N_c e^{-(E_c - E_{F_n})/kT_n} \tag{32.24}$$

$$p = N_v e^{-(E_{F_p} - E_v)/kT_p} \tag{32.25}$$

Similar to the law of mass action the product is given by

$$np = n_i^2 e^{\Delta\mu/kT} \tag{32.26}$$

with the expression $\Delta\mu = E_{F_n} - E_{F_p}$ for the difference in the quasi Fermi levels. Note that in general E_{F_n}, E_{F_p} and thus $\Delta\mu$ are a function of position.

According to an analytical approach which uses the Boltzmann Transport equation (Ref. [**?**]) the charge carrier currents are given by:

$$J_n(r) = \mu_n n \nabla E_{F_n} \tag{32.27}$$

$$J_p(r) = \mu_p p \nabla E_{F_p} \tag{32.28}$$

This has the important consequence that charge carrier currents are driven by the gradient of the respective quasi Fermi levels.

32.6 Energy levels

For future discussions it is important to understand the differences between the various terms describing differences on the energy scale of a semiconductor. Figure 32.6 illustrates the various quantities which will be used in the following.

The energy level labeled 0 is just a reference energy level. In principle this could be any energy level. Often the vacuum potential is chosen as reference energy level.

$e\phi$ designates the energy level of the electrostatic potential which relates to the electric field according to: $F = -\nabla\phi$

E_C and E_V are the conductive resp. the valence band edges as usual. $E_{F,C}$ and $E_{F,V}$ designate in this illustration the quasi Fermi levels.

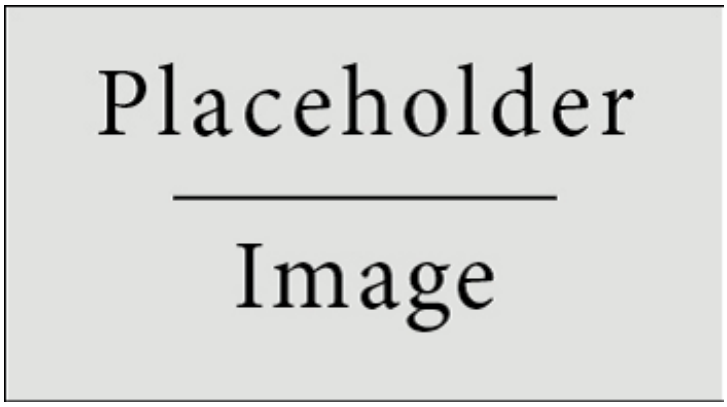

Figure 32.6: Illustration of the potential differences used for describing the semiconductor characteristics (Source: [**?**])

The chemical potential μ_e and μ_h, i.e. the amount of energy needed to remove an particle from the semiconductor, has a concentration dependent and a concentration independent contribution. The concentration independent contribution $\mu_{e,0}$ resp. $\mu_{h,0}$ which is determined by the base material and is a material constant is identified with the electron resp. hole affinities χ_n and χ_p. The concentration dependent contribution $kT\ln(n_e/N_C)$ resp. $kT\ln(n_h/N_V)$ are the difference between the respective band edge to the quasi Fermi level.

The Fermi levels can now be identified with the electrochemical potentials η_e and η_h, i.e. the sum of the chemical and the electrostatic potential.

32.7 Origins of quasi Fermi level gradients

As the current arises from the gradient of the quasi Fermi levels it is worthwhile to investigate the formation of this gradient a little further. By reformulating equations 32.24 and 32.25 as expressions for E_{F_n} resp. E_{F_p} and differentiating we find an expression for the gradient of the quasi Fermi level :

$$\nabla E_{F_n} = (\nabla E_c - kT\nabla \ln N_c) + \frac{kT}{n}\nabla n \tag{32.29}$$

$$\nabla E_{F_p} = (\nabla E_v - kT\nabla \ln N_v) + \frac{kT}{p}\nabla p \tag{32.30}$$

The gradient of the conduction band edge however can be expressed (according to section 32.6) as electrostatic field F and gradient in the electron affinity (which is a property of the material composition)

$$\nabla E_c = qF - \nabla\chi \tag{32.31}$$

For the gradient of the valence band edge additionally the gradient of the bandgap has to be considered.

$$\nabla E_v = qF - \nabla\chi - \nabla E_g \tag{32.32}$$

Therefore, the gradient of the quasi Fermi levels and thus a current can originate from five different origins:

1. electrostatic potential
2. gradient of the electron affinity
3. gradient of the band gap
4. gradient of the effective density of states
5. gradient of the charge carrier density

Putting all this into Eqs. 32.27 and 32.28 and applying the Einstein relations 32.4 and 32.5 results in the following expressions for the electron and hole currents:

$$J_n(r) = qD_n\nabla n + \mu_n n(qF - \nabla\chi - kT\nabla \ln N_c) \tag{32.33}$$

$$J_p(r) = -qD_p\nabla p + \mu_p p(qF - \nabla\chi - \nabla E_g + kT\nabla \ln N_v) \tag{32.34}$$

Let us take a moment to discuss these expressions a little further.

- The current in semiconductors can have multiple origins.
- The electric field is only one of many possible origins of the electric current.
- The gradient in the material properties bandgap, electron affinity and effective density of states create an effective electric field which add to the electrostatic field.
- The term dependent on the gradient of the charge carrier concentrations (left) could be seen as contribution due to diffusion.
- The term dependent on the the charge carrier concentration (right) could be seen as drift current due to the sum of electrostatic and effective field.
- In homogeneous materials no effective electric field exists and only the electrostatic field contributes to the drift current.

$$J_n(r) = qD_n \nabla n + q\mu_n F n \tag{32.35}$$

$$J_p(r) = -qD_p \nabla p + q\mu_p F p \tag{32.36}$$

It is very important to note that the picture of separate drift and diffusion currents is misleading. The separation is purely mathematically and no real drift and diffusion currents exist. Only the net current should be considered. However, this drift/diffusion picture is more intuitive and widely used in literature about photovoltaics, yet, one should always keep in mind that in reality the current originates only from the net gradient of the quasi Fermi levels. Thus consider the following two examples as special extreme cases:

Example 32.1 — No concentration gradient. The net current equals the contribution of the drift currents.

$$J = J_n + J_p = q(\mu_n n + \mu_p p)F \tag{32.37}$$

Figure 32.7 shows the drift of the charge carriers. The negatively charged electrons drift down the gradient in the conduction band, while the positive holes drift upwards the gradient of the valence band.

Placeholder

Image

Figure 32.7: The negatively charged electrons drift down the gradient in the conduction band, while the positive holes drift upwards the gradient of the valence band. (Source: [**?**])

Example 32.2 — No electric field. The net current equals the contribution of the diffusion currents.

$$J_n = q(D_n \nabla n - D_p \nabla p) \tag{32.38}$$

Figure 32.8 shows how the concentration profiles of a strongly assymetric charge carrier distribution levels out due to diffusion.

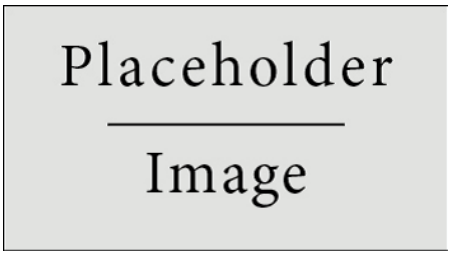

Figure 32.8: The concentration profiles of a strongly asymetric charge carrier distribution levels out due to diffusion. (Source: [**?**])

Exercise 32.5 — Conductivity in silicon. Let us close the loop back to the beginning of this chapter. The mobility of the electrons in the conduction band of silicon ($E_g = 1.1$ eV) is about $\mu_n = 1350 cm^2/Vs$ while the mobility of the holes in the valence band is about $\mu_p = 480 cm^2/Vs$.

1. As n_i is a material parameter you should find it in the literature or the internet. Please research the value and give its value (including the source).
2. How large is the conductivity in intrinsic Si?
3. (Solve this one after you have read the whole chapter) The silicon is now doped with $10^{16}\mathrm{cm}^{-3}$ donors. The Fermi level at room temperature is close to the conduction band edge such that we assume full ionization of the donors. How large is the conductivity of the doped silicon?

Exercise 32.6 — Assumptions. Name the most important assumptions in the derivation of the charge carrier currents.

Part IV

Electrical Power Systems

41 - DC Circuit Concepts

Learning objectives

After processing this chapter

- you know the basic quantities and their units used in electrical engineering
- you know the formulas describing the mathematical relations in between them
- you can apply fundamental calculation methods to simple DC networks
- you understand the behaviour of the electric components R, L and C

Preface

The abbreviation DC stands for "direct current" that kind of electrical energy everybody knows from a simple battery. DC current plays only a minor role in nowadays energy supply which is dominated by AC systems. Nonetheless the basic principles of electrical relations can be explained best in DC circuits why we will start our lecture with this topic.

41.1 Voltage - Current - Power

The major quantities used for electrical circuits in order to describe its behaviour are listed in the following table:

Quantity	Symbol	Unit	Abbreviation
Voltage	U (sometimes V)	volt	V
Current	I	ampere	A
Power	P	watt	W
Energy	W	watt-hour	Wh

An electric circuit connects mainly two-terminal components by series and parallel arrangements. So-called active components are voltage or current sources (e.g. a battery or a generator). Typical passive components are resistors, inductors and capacitors as listed subsequently.

Quantity	Symbol	Unit	Abbreviation
resistance	R	Ohm	Ω (=V/A)
inductance	L	Henry	A (=Vs/A)
capacitance	C	Farad	F (=As/V)

The next figure shows a simple DC circuit diagram with a battery as constant voltage source and one passive element (R) connected to it. The resistor R can be replaced by L or C as well.

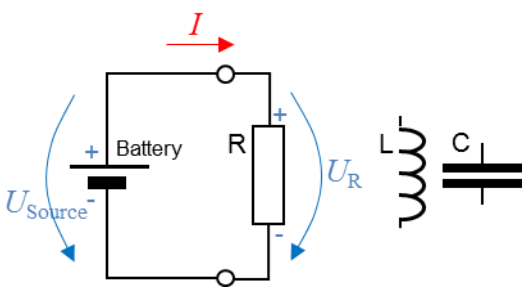

Figure 41.1: Basic DC circuit diagram

All quantities described so far correlate with one another. For example, the battery forces a current flow from its positive pole via the resistor to the negative side. This current passing the resistor causes a voltage drop across it which can be measured by means of a voltmeter.

The basic relations valid for such a simple DC circuit are described by the following formulas:

Ohm's Law $$R = \frac{U}{I} \qquad (41.1)$$

The reciprocal value of the Ohm's resistance is called

conductance $$G = \frac{1}{R} = \frac{I}{U}$$ measured in S (Siemens) (sometimes also as "mho" – the reverse spelling of 'Ohm') (41.2)

electric power $$P = U * I = \frac{U^2}{R} = I^2 * R \qquad (41.3)$$

electric energy $$W = P * t$$ t – time (41.4)

41.2 Calculation in Electrical Circuits

The following rules are valid for both DC circuits as well as AC circuits.

Kirchhoff's Current Law

The current flowing into a node or branching point is equal to the sum of the individual currents leaving the node or branching point.

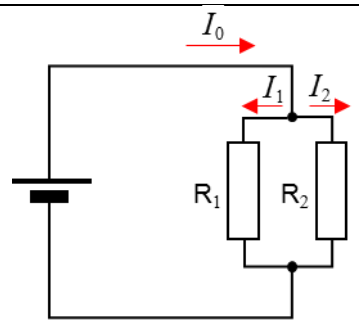

Figure 41.2: Kirchhoff's Current (1st) Law

$$I_0 = I_1 + I_2 \qquad (41.5)$$

Kirchhoff's Voltage Law

The algebraic sum of all the voltages around any closed path in a circuit equals zero.

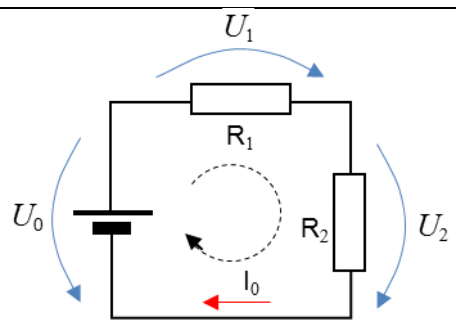

Figure 41.3: Kirchhoff's Voltage (2nd) Law

$$0 = U_1 + U_2 - U_0 \qquad (41.6)$$

$$U_0 = U_1 + U_2$$

Series Connection

The algebraic sum of series resistances represents the equivalent resistance.

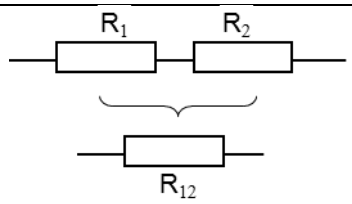

Figure 41.4: Series Connection

$$R_{12} = R_1 + R_2 \tag{41.7}$$

Parallel (or Shunt) Connection

The algebraic sum of parallel conductances represents the equivalent conductance.

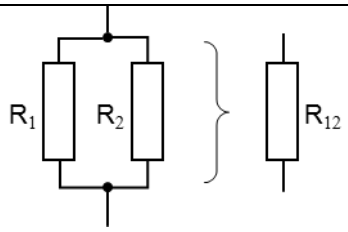

Figure 41.5: Parallel Connection

$$G_{12} = G_1 + G_2 \tag{41.8}$$

$$R_{12} = \frac{1}{G_{12}} = \frac{R_1 * R_2}{R_1 + R_2}$$

Example 1.2

Given is a DC circuit with the following values:

U_B=12V
R_1=5Ω
R_2=5 Ω
R_3=10 Ω.

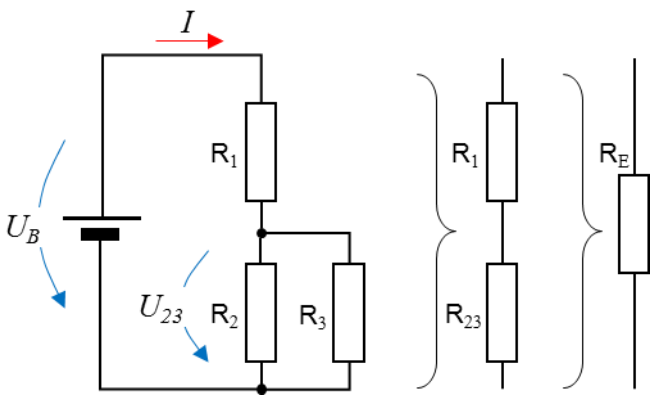

Figure 41.6: Example 1.2

Calculate:

a) the equivalent resistance R_E

First determine the equivalent resistance of the shunt (parallel) connection of R_2 and R_3 by means of their conductances.

$$G_{23} = G_2 + G_3 = \frac{1}{R_2} + \frac{1}{R_3} = \frac{1}{5} + \frac{1}{10} = 0.3S$$

$$R_{23} = \frac{1}{G_{23}} = 3.33\Omega$$

Adding resistance R_1 results in the required equivalent resistance (R_1 and R_{23} in series).

$$R_E = R_1 + R_{23} = 8.33\Omega$$

b) the power delivered by the source

$$P = U \cdot I$$

If replacing I by means of Ohm's Law solved for I you get

$$P = U \cdot I = U \cdot \frac{U}{R} = \frac{U^2}{R} = \frac{12^2}{8.33} = 17.28W$$

c) the voltage drop across R_3

The current flowing out of the battery passes both resistances in series (here R_1 and R_{23}) and causes a voltage drop at both depending of its size (R).

$$I = \frac{P}{U_S}\left(= \frac{U_S}{R_E}\right)$$ (many possibilities to determine the overall current!)

The voltage drop across R_{23} represents the requested voltage U_3

$$U_3 = U_{23} = R_{23} \cdot I = R_{23} \cdot \frac{P}{U_S} = 3.33 \cdot \frac{17.28}{12} = 4.8V$$

Exercise 1.1

Determine
a) the voltage measured across resistor R_1 and
b) the current flowing through R_3.

41.3 Behaviour of Electrical Components

The three elements R, L and C do not only differ in its unit as discussed above, they show a totally different behaviour during cut-in time when being connected to an electric voltage source. This peculiarity becomes more obvious in AC circuits where the voltage is permanently changing.

Resistor

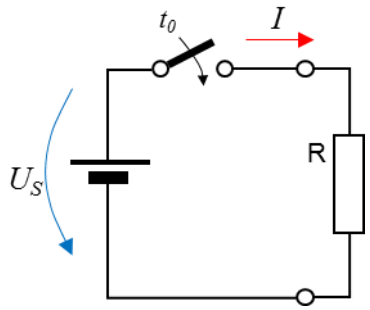

Figure 41.7:
Resistor in a DC circuit

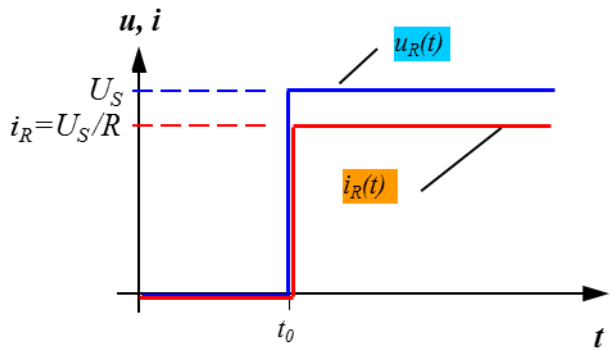

Figure 41.8:
Voltage and current at a resistor

The closing of the switch at time t_0 connects the source voltage with the resistor and forces a constant current flowing through it. The size of the current is calculated by means of Ohm' Law.

$$u_R(t) = U_S \tag{41.9}$$

$$i_R(t) = \frac{U_S}{R} \tag{41.10}$$

Capacitor

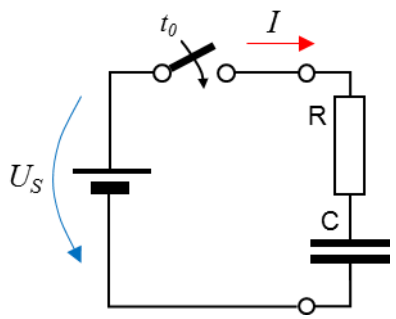

Figure 41.9:
Capacitor in a DC circuit

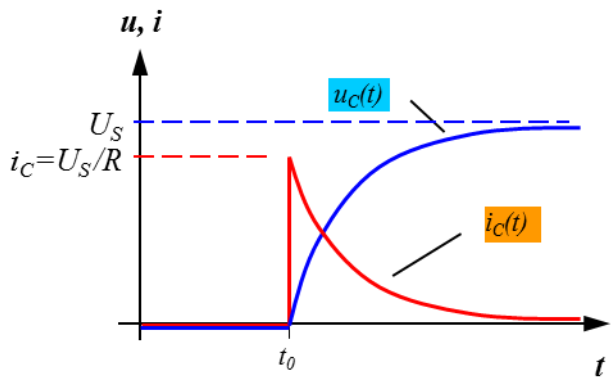

Figure 41.10:
Voltage and current at a capacitor

Let's regard the capacitor as a totally discharged battery. The voltage measured across the ends is zero, even though it is connected to the voltage source U_S. At t_0 a high charging current starts flowing limited only by the resistor R which is connected in series with the capacitance. The capacitor loads up gradually as can be seen by the voltage curve. The time constant of C and R determines the speed of this loading process.

$$\text{Time constant:} \quad \tau = C * R \tag{41.11}$$

$$u_C(t) = U_S * (1 - e^{-\frac{t}{\tau}}) \tag{41.12}$$

$$i_C(t) = \frac{U_S}{R} * e^{-\frac{t}{\tau}} \tag{41.13}$$

Inductor

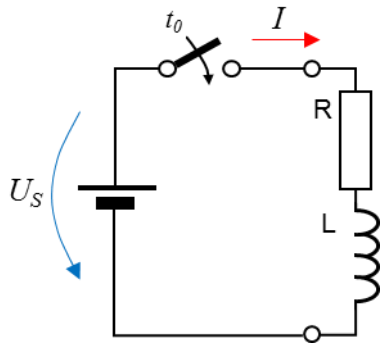

Figure 41.11:
Inductor in a DC circuit

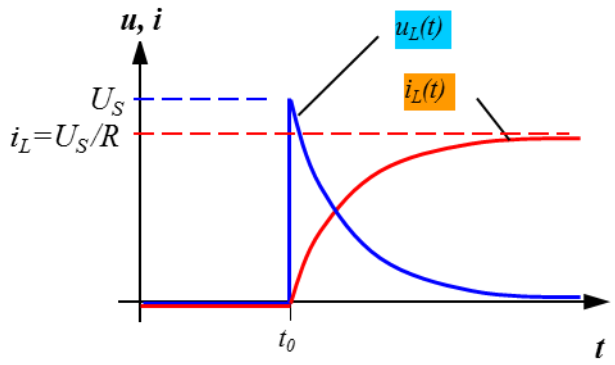

Figure 41.12:
Voltage and current at an inductor

An inductor behaves opposite to a capacitor. While the capacitance damps (the rising of) the voltage across it the inductor delays the current flow. The rising current creates a magnetic flux which in return induces a counter voltage in the coil causing that damping behaviour. After dying down of the cut-in rush the final current is only limited by the serial resistor. The time constant of L and R determines the speed of this settle process.

$$\text{Time constant:} \quad \tau = \frac{L}{R} \tag{41.14}$$

$$u_L(t) = U_S * e^{-\frac{t}{\tau}} \tag{41.15}$$

$$i_L(t) = \frac{U_S}{R} * (1 - e^{-\frac{t}{\tau}}) \tag{41.16}$$

Exercise 1.2

A given DC circuit consists of a battery (400V) and some power consumers summarized in the load resistance R_L (25Ω). This circuitry is protected by a fuse for a nominal current of I_{fuse}=20A.

A capacitor shall be charged from that battery by a shunt connection to the existing loads. The cut-in rush current would trip out the fuse if not being limited by an additional resistor R_C in series with C. Determine the size of R_C so that the nominal value of the fuse is not exceeded.