Numerical Methods for Physicists

by

Volker Hohmann
Institute of Physics
University of Oldenburg, Germany

http://www.physik.uni-oldenburg.de
Version 1.0, April 2004

Contents

1 INTRODUCTION 5

1.1 Definition: Computer physics 5

2 MATLAB – FIRST STEPS 6

2.1 Using Matlab 6

2.2 Help 7

2.3 Variables 7

2.4 Operators 8

2.5 Loops and conditions 9

2.6 Generating matrices/vectors 10

2.7 Functions 10

2.8 Visualization 11

2.9 Input and output 12

2.10 Writing individual commands (“M-Files“) 12

2.11 Exercises 14

3 REPRESENTATION OF NUMBERS AND NUMERICAL ERRORS 15

3.1 Error measures 15

3.2 Representation of numbers and roundoff errors 15
3.2.1 Integers 16
3.2.2 Floating point numbers 17

3.3 Range errors 18

3.4 Truncation errors 18

3.5 Error propagation in arithmetic operations 18

3.6 Error propagation in iterated algorithms 19

3.7 Exercises 21

4 NUMERICAL DIFFERENTIATION AND INTEGRATION 22

4.1 Differentiation 22
4.1.1 Right-hand formula (”naive“ ansatz) 22
4.1.2 Centered formula 23
4.1.3 Richardson extrapolation 24

4.2 Integration 26
4.2.1 Trapezoidal rule 26
4.2.2 Recursive trapezoidal rule 27

4.2.3 Error analysis and Romberg integration 27
4.2.4 Gaussian integration 28

4.3 Exercises 29

5 ORDINARY DIFFERENTIAL EQUATIONS (ODE) 30

5.1 Mathematical formulation of the initial value problem 30

5.2 Simple methods 31
5.2.1 Euler's method 31
5.2.2 Euler-Cromer method 32
5.2.3 Midpoint method 32
5.2.4 Verlet method 33

5.3 Global and local truncation error 33

5.4 Runge-Kutta method of the 4th order 33
5.4.1 Adaptive step size selection 35

5.5 Application of various methods to solve initial value problems 36
5.5.1 Projectile motion 36
5.5.2 Physical pendulum 37
5.5.3 Planetary motion 39
5.5.4 Lorenz model 40

5.6 Boundary value problems 40

5.7 Exercises 41

6 SOLVING SYSTEMS OF EQUATIONS 43

6.1 Linear systems of equations 43
6.1.1 Elementary operations 44
6.1.2 Gaussian elimination with backsubstitution 44
6.1.3 Gauss-Jordan method 46
6.1.4 Singular value decomposition 47

6.2 Nonlinear systems of equations 50
6.2.1 Newton's method 50

6.3 Exercises 51

7 MODELING OF DATA 52

7.1 General mathematical formulation of the fitting problem 52

7.2 Establishing the merit function: Maximum-Likelihood Methods 53
7.2.1 Least-Squares method 53
7.2.2 Chi-Square method 54
7.2.3 Robust Fit methods 54

7.3 Minimization of functions 55
7.3.1 "General Least-Squares" method 55
7.3.2 Methods for nonlinear fitting: The "Simplex" method 57

7.4 Error estimation and confidence limits 59
7.4.1 Chi-square fitting: Confidence regions of parameters 59
7.4.2 The "Bootstrap" method for estimating the confidence regions 60

7.5 Goodness of fit **60**

7.6 Summary of methods **62**

7.7 Exercises **63**

8 ANALOG-DIGITAL TRANSFORM AND DISCRETE FOURIER TRANSFORM 65

8.1 Analog-Digital Transform (A/D Transform) **65**

8.2 Diskrete Fourier Transform (DFT) **66**
8.2.1 Real-valued time functions 68
8.2.2 Intensity spectrum 68

8.3 Fast Fourier Transform (FFT) **69**

8.4 Filtering and convolution theorem **70**
8.4.1 Convolution and cyclic convolution 71
8.4.2 Deconvolution 72

8.5 Exercises **73**

9 PARTIAL DIFFERENTIAL EQUATIONS 74

9.1 Classification of partial differential equations **74**
9.1.1 Parabolic equations 74
9.1.2 Hyperbolic equations 74
9.1.3 Elliptic equations 74

9.2 Initial value problems **75**
9.2.1 Discretization 75
9.2.2 Heat equation: FTCS scheme 76
9.2.3 Time-dependent Schrödinger equation: Implicit Scheme (Crank-Nicholson) 76
9.2.4 Advection equation: Lax-Wendroff scheme 78
9.2.5 von-Neumann stability analysis 81

9.3 Boundary value problems: Poisson and Laplace equations **83**
9.3.1 Laplace equation: Relaxation methods (Jacobi method) 83
9.3.2 Poisson equation 85

9.4 Exercises **89**

1 Introduction

This lecture does not deal with the physics of computers but is rather about how to solve physical problems by means of computers. Nowadays, computers play an important part as tools in experimental physics as well as in theoretical physics. Special fields of use are:

1. Data logging
2. Systems control
3. Data processing (calculation of functions, visualization)
4. Transformation of theoretical models on the computer

Data logging and systems control cannot be done by hand and eye any longer and the aid of fast computers is required for most experiments. Data processing enables large quantities of data to be analyzed and replaces generations of ”human computers“, who were forced to calculate functional values from tables and to draw diagrams by hand. Using computers for modeling is certainly the most complex way of employing computers in physics, which is always necessary when analytical (mathematical) solutions do not exist. The three-body problem, the nonlinear (physical) pendulum as well as solving the Schrödinger equation for complex atoms are such examples. In these cases numerical methods are applied on a computer which seek the (approximate) solution by calculating numbers according to simple recurrent rules (algorithms)[1]. The validity of such numerical solutions and the applied numerical methods must always be examined very closely and it is advisable to be suspicious about the results obtained in this way. This can be summarized in one sentence: It is insight not numbers that we are interested in.

The objective of the lecture is to understand the fundamentals of these applications, especially points 3 and 4, by means of simple examples and to acquire practical skills which can be applied or extended to concrete problems later on. Even if the tasks are sometimes very complex: In the end it is always a matter of ’merely’ calculating numbers according to simple rules.

1.1 Definition: Computer physics

The term ”computer physics” is to be defined on a mathematical basis:

1. Mathematics (e.g. analysis)
- is concerned with relatively abstract terms and uses symbols rather than numbers (numbers being also symbols which are, however, described as an abstract quantity, e.g. group of natural numbers)
- Solutions are always analytical, i.e. determined with arbitrary precision in principle; exceptions from this rule are dealt with theoretically (e.g. singularities)
- If analytical solutions are not known, proofs of existence or non-existence are investigated

2. Numerical mathematics
- deals with the calculation of numbers according to simple recurrent rules (algorithms)
- deals with questions of principle regarding the computability of numbers (convergence orders)
- Validation of algorithms on the basis of error estimates

3. Numerical mathematics on a computer
- Numerical mathematics + (unknown) loss of precision (e.g. by finite computational accuracy)

4. Computer physics
- Application of 3. to physical problems
- Additional possibility of validating algorithms on the basis of physical boundary conditions

[1] Of course, the computer also helps to find analytical solutions; however, the respective computer algebra programs (e.g. Maple or Mathematica) are based on the knowledge of the programmers, i.e. they do not find solutions which are unknown in principle to the programmers.

2 Matlab – first steps

Matlab is a special program for numerical mathematics and is used throughout this course. It is easy to use and allows us to rapidly enter the world of Numerics. Matlab is a very good complement to the known programs of symbolic (analytical) mathematics (Maple, Mathematica) and is applied when no consistent analytical solutions can be found for a problem. The fundamental data structure of Matlab are two-dimensional matrices (hence the name: MATrix LABoratory). Matlab can carry out mathematical calculations in a syntax similar to that of C and FORTRAN, respectively. It comprises all fundamental mathematical functions as well as special functions (Bessel etc.). The essential algorithms known from numerical mathematics (solution of linear systems of equations, eigenvalues/eigenvectors, differential equations etc.) have been implemented. Additionally, the program includes options for graphically representing functions and data sets. It can generate two-dimensional xy plots, plots in polar coordinates as well as two-dimensional representations. So-called toolboxes with further functions are available for special applications (image processing, signal processing), which are not contained in the standard package.

The following table shows the advantages and disadvantages of Matlab compared to programming languages such as C or Fortran:

Advantages	**Disadvantages**
fast program development	possibly slower execution as compared to optimized C programs
very good graphic options	
many important algorithms implemented	
portable programming (Linux, Windows)	

Matlab version 6.x including a toolbox for signal processing is available in the CIP room of the Physics Department and other public computer rooms on the campus. Since Matlab is licensed, it may not be transferred to other computers! A login for the Windows network of the computing center of the university is required for using Matlab (for details see: http://www.physik.uni-oldenburg.de/docs/cip/start.html)

In the following the fundamental structure of Matlab as well as the most important commands are explained. Further below you will find some exercises, which will help you to practice Matlab.

2.1 Using Matlab

Upon selection of Matlab there appears a command line editor into which commands can be written, which are executed by the program when the Return key (↵) has been pressed. Command lines executed before can be recalled using the arrow keys (↑↓). Writing an initial and pressing the arrow keys produces only those previous commands starting with that letter (or group of letters). The command line can be edited as known from text processing programs (marking symbols/words with the mouse, replacing and inserting symbols/words etc.).

Matlab searches for the commands written in the command line in the directories stated in the path. With the command "addpath" the path can be extended, if, for example, your personal directory with your own commands is to be added to the path.

The command diary *file name* enables Matlab to remember all commands as well as the "answers" given by Matlab in the stated file. Thus, the sequence of commands and results can be traced back afterwards.

Matlab stores the names and values of all variables generated during program execution in the so-called "workspace". With the commands save and load the workspace can be stored into a file and loaded (reconstructed again later on to continue the task. With the command clear the variables in the workspace can be deleted. The commands can be restricted to variable names by attaching variable names. For example, the command save test.dat x y would only store the variables x and y in the file test.dat. The commands who and whos list the variables available in the workspace, whos yielding more detailed information.

The graphs appear in additional windows on the screen. With the command figure a new graphic window is generated, so that several graphs can be produced.

2.2 Help

In order to get help for a specific Matlab command write "help *command*". Help offers a list of possible subjects without further arguments. If the name of a command is not known, it is recommended to use the command lookfor *keyword*. The keyword needs to be an English word or phrase. Its proper choice is crucial for the succesful application of the lookfor-command.

Access to the entire documentation is obtained by the command helpdesk. The documentation then appears in the HTML format. The manual "Getting started" which includes a detailed introduction to Matlab is of special interest to beginners.

Furthermore, "Examples and Demos" can be started from the help menu. There are examples arranged according to subjects. Generally, the Matlab commands to be used are shown, such that the examples can be used as a template for individual applications.

2.3 Variables

Names for variables may be assigned freely. Capitals and lower case letters are to be considered. Variables are introduced by assignment and do not have to be declared. The type of variable (scalar, vector, matrix) is determined by Matlab according to the assignment. Examples:

```
a = pi                  % the value Pi=3.1415... is assigned to a
b = [1 2]               % the (row) vector (1, 2) is assigned to b
c = [1 ; 2]             % the (column) vector (1, 2) is assigned to c
d = [[1 2 3];[4 5 6i]]  % a 2X3 matrix is assigned to d
```

(**Hints**: The text following the symbol % is considered a comment.
The variable pi is predefined.
The variable $i = \sqrt{-1}$ is predefined for defining complex values).

The value of a variable can be displayed at any time by writing the name of the variable without any additions in the command line. Examples:

```
a               % input of variable
a = 3.14        % answer by MATLAB (scalar)

b               % input of variable
b = 1   2       % answer by MATLAB (line vector)

c               % input of variable
c = 1           % answer by MATLAB (column vector)
    2

d               % input of variable
d = 1 2 3       % answer by MATLAB (2X3 matrix)
    4 5 6i
```

Elements or sectors of matrices and vectors can also be selected and assigned to variables, respectively. Examples:

```
d(1,3)              % select the first tow, third column
d = 3               % answer by MATLAB

d(1,2:3)            % select the first, row, elements 2 to 3
d = 2 3             % answer by MATLAB

z = d(2,1:3)        % select row 2, columns 1 to 3  to  z
z = 4 5 6i          % answer by MATLAB

d(2,2) = 3          % assign the value 3 to the matrix d (row 2, column 2)
d = 1 2 3           % answer by MATLAB (2X3 matrix)
    4 3 6i

d(1,:)          % address the first row
  = 1 2 3       % answer by MATLAB

d(1,1:2:3)      % address the first row, every second element)
  = 1  3        % answer by MATLAB

d(1,[3 1])      % address the first row, third and first elements
  = 3  1        % answer by MATLAB
```

(**Hints:** Each operation causes Matlab to document its work on the screen. This can be stopped by typing a semicolon (;) after a command. Indexing of elements of vectors/matrices always starts with index 1.) The term 'end' is a place holder for the highest index (last entry in a vector). E.g., a(1,end) always indexes the last column in row 1.

2.4 Operators

The apostrophe or inverted comma operator ' (**Attention**: not to be confused with double quotation marks!) performs a complex conjugation:

```
y = c';         % transform c into a row vector
y               % show result
y = 1 2         % answer by Matlab

y = d';         % calculate complex conjugated matrix
y               % show result
y = 1  4        % answer by MATLAB (3X2 matrix)
    2  5
    3 -6i
```

However, the operator .' performs a transposition:

```
y = d.';        % calculate transposed matrix
y               % show result
y = 1  4        % answer by MATLAB (3X2 matrix)
    2  5
    3  6i
```

The known operators +,-,*,/ function in Matlab just as in a pocket calculator. Applying these operators to matrices and vectors, however, may lead to confusions, since Matlab analyses the variables to the left and right of the operators and selects the operators 'intuitively', i.e. it selects scalar or matrix operations depending on the dimension of the variables. Examples:

```
y = a * a;      % multiply two scalars
y               % show result
```

```
y = 9.8696      % answer by Matlab

y = b * c;      % multiply row by column vector
y               % result is a scalar product
y = 5

y = b .* c'     % multiply two row vectors component by component
y = 1 4         % result is a row vector again


y = d .^ d      % exponentiating component by component
y = d ^ 3       % exponentiating matrix
```

Similarly matrices can be multiplied by each other or by vectors, added etc. If an operation is to be done component by component, a dot has always to be put before the operator. In any operation Matlab is very exact about the dimension of matrices and vectors and gives the error message "matrix dimensions must agree" in case of discrepancies.

2.5 Loops and conditions

Repetitive operations are performed in loops. The following loop calculates a row vector and a column vector:

```
for i=1:5               % start of loop
        p(i) = i^2;     % entries of a row vector
        q(i,1) = i^3;   % entries of a column vector
end                     % end of loop
```

Several loops can be interleaved. The index increment can be selected by the colon operator:

```
for i=1:2:5             % start of loop over uneven indices
        q(i) = i;       % entries for uneven indices
        q(i+1) = -i;    % entries for even indices
end                     % end of loop
```

When the command break is given within a loop, the loop is aborted. There are also loops which are carried out as long as a certain condition is fulfilled:

```
x=100;
while( x > 1)           % start of loop
        x = x/2;
end                     % end of loop
```

(**Hint**: Loops are relatively slow in Matlab. If ever possible, they should be replaced by vector operations).

The following operators are available for establishing relations:

Operator	Function
==	even
~=	uneven
>	greater
>=	greater than or equal to
<	smaller
<=	smaller than or equal to
&	logical “and“
|	logical “or“
~	logical “not“

The conditional execution of commands is done with so-called if-constructions.

```
if( x == 0 | x == 1 )              % start of if-construction
        status = 1;
elseif( x < 0 & x ~= -1 )          % Attention: elseif must be written in one word
        status = -1;
else
        status = 0;
end                                % end of if-construction
```

2.6 Generating matrices/vectors

The colon operator (:) serves to address parts of matrices/vectors (cf. section *Variables*), but also to generate matrices/vectors. Examples:

```
x = [1:10];                % generate row vector with integers between 1 and 10
x                          % show result
x = 1 2 3 4 5 6 7 8 9 10 % answer by Matlab

x = [1:3 ; 1 3 4];         % generate (2X3) matrix
x                          % show result
x = 1 2 3                  % answer by Matlab
    1 3 4
```

A step size can also be selected:

```
x = [0:pi/4:pi];                    % generate row vector with numbers from 0 to Pi,
                                    % step size Pi/4.
x                                   % show result
x = 0 0.7854 1.5708 2.3562 3.1416   % answer by Matlab
```

The following functions are available for generating matrices/vectors:

```
zeros   - generates matrices/vectors and fills them up with zeros
ones    - generates matricen/vectors and fills them up with ones
rand    - generates matrices/vectors and fills them up with evenly distributed random
numbers in the interval (0,1)
randn   - generates matrices/vectors and fills them up with Gaussian random numbers
          (mean 0, variance 1);
```

The required numbers of lines and columns each are given as arguments. Example:

```
x = randn(2,3);  % generate (2X3) matrix with random numbers (Gaussian)
```

2.7 Functions

Matlab offers all possible mathematical and numerical functions. All functions are applied to the entire vector/matrix. For example, the following command generates a vector containing the values of the sinusoidal function between 0 and 2 Pi with a resolution of Pi/10:

```
x = sin([0:pi/10:2*pi]);
```

The following table shows fundamental and special mathematical functions:

Function	Description
sqrt(x)	square root
sin(x)	sine
asin(x)	arc sine
sinh(x)	hyperbolic sine

asinh(x)	hyperbolic arc sine
cos(x)	cosine
tan(x)	tangent
exp(x)	exponential function
expm(X)	matrix exponential function
log(x)	natural logarithm
logm(x)	matrix logarithm
log10(x)	decimal logarithm
rem(i,n)	modulo function
round(x)	rounding towards nearest integer
floor(x)	rounding towards minus infinity
ceil(x)	rounding towards plus infinity
inv(X)	calculate the inverse of a matrix
fft(x)	fast Fourier transform
abs(x)	absolute value
real(x)	real part
imag(x)	imaginary part
angle(x)	phase angle
sum	sum of vector elements
prod	product of vector elements
max	search of maximum
min	search of minimum
mean	mean
std	standard deviation
cov	(co-)variance
median	median
bessel	Bessel functions
erf	Gaussian error function
gamma	gamma function
size	calculate quantity of a matrix/vector
hist	histogram
sort	sorting algorithm

2.8 Visualization

Various visualization options are available to represent data graphically. The simplest one are xy-plots which represent data points (x,y) in a cartesian coordinate system:

```
x = [0:pi/10:2*pi];
y = sin(x);
plot(x,y);
```

These commands generate a period of the sinusoidal function as a graph. The following table shows further fundamental commands available for visualization:

Function	Description
plot(x,y)	xy plot, linear axes
loglog(x,y)	xy plot, double logarithmic representation
semilogx(x,y)	xy plot, logarithmic x-axis
semilogy(x,y)	xy plot, logarithmic y-axis
title('text')	set image title
xlabel('text')	set lable of x-axis
ylabel('text')	set lable of y-axis
axis	set value range of axes
polar(theta,rho)	polar plot
contour(z)	2D representation of contour of matrix z
image(z)	2D color representation of matrix z

imagesc(z)	like image, but with scaling of data
pcolor(z)	similar to image(z)
colormap	set color scale in case of color representation
plot3(x,y,z)	3D representation of vectors
contour3(z)	3D representation of contour of matrix z
mesh(z)	3D mesh plot of matrix z
surf(z)	3D mesh plot of matrix z with color representation

In order to practice 2D and 3D representations a text matrix can be generated with the command peaks. For example, the command imagesc(peaks) plots a 2-D color representation of the matrix generated by peaks.

2.9 Input and output

The command input reads an input from the keyboard:

```
x = input('write value of x: ')
```

This command displays the text on the screen and waits for an input from the keyboard. The input is assigned to the variable x.

The command disp displays text and values of variables on the screen:

```
disp('the value of x is: ')
disp(x)
```

These commands display the text as well as the value of variables x.

The command fprintf is available for a formatted printout as known from the programming language C. Additionally, the commands fopen, fread a well as fwrite and fclose are available for reading or writing in files (Hint: not to be confused with the commands load and save, which store the workspace in a special Matlab data format).

2.10 Writing individual commands ("M-Files")

Individual programs and functions are easy to write in Matlab. For this purpose a so-called **script** is written, which is a text file comprising Matlab commands. Matlab interprets the file name as the name of the command; the extension "m" of the file is fixed. Therefore, the files are called M-Files. How such M-Files have to be arranged is explained by an example in the following. Further information can be obtained by the Matlab Help function. Write help script or help function.

(**Hint**: The directory in which the files are stored must be in the search path (command path, addpath) or the files must be in the actual directory (command cd) so that Matlab can find the individual M-Files.)

A script is produced by writing the commands into the text file line by line. A usual text editor (e.g. notepad) can be used for this, but also the special Matlab text editor (call the editor by selecting Open or New in the menu File). The script is started by writing the file name in the command line. Matlab then executes all commands and keeps the variables generated upon execution of the commands within the workspace.

(**Hint**: When an M-File is changed, Matlab perhaps does not read it in anew. The old version is executed (flaw in the network installation of Matlab, e.g. in the CIP room). The command clear all forces the program to read in a new version which, however, produces the side-effect that all variables in the workspace are deleted.

The script can be transformed into a function, which behaves like a special Matlab function. For this certain formalities are to be considered, which are explained by an example in the following:

```
function [mean,stdev] = stat(x)
        %
        % [mean,stdev] = stat(x)
        %
        % This function calculates the mean and standard deviation of a data set
        %
        % Transfer parameter:
        %    x – data set as vector
        %
        % Return values:
        %         mean    - mean
        %         stdev    - standard deviation
        %

        n = length(x);
        mean = sum(x) / n;
        stdev = sqrt(sum((x - mean).^2)/n);
```

The first line begins with the keyword function and declares the name of the function (here: stat), the arguments in parentheses (here: x) as well as the return values (here: mean and stdev). The following list of comments is displayed by Matlab, if the help function is called for this command (help stat). Then the actual function starts which can consist of Matlab commands. It is important that the return values are calculated from the arguments. For this purpose subsidiary variables can be generated, too (here: n). The function controls its own workspace, i.e. the function only knows the values of the arguments and only the return values are returned to the calling program. All subsidiary variables are local and are deleted upon termination of the function. Now the function can be called from the command line as follows:

```
noise = randn(1,100);              % generate random numbers
[mnoise,snoise] = stat(noise);     % calculate statistics
disp('mean and standard deviation are:');
mnoise
snoise
```

2.11 Exercises

Remark: Save the solutions of all exercises in Script files ("M-Files").

I. Generating data

a. Generate a time vector from 0ms to 10 ms at a sampling frequency of 10 kHz.
b. Generate a sine with a frequency of 1 kHz over the time basis given by the time vector from a.
c. Plot the sine from b. over the time vector from a.
d. Like c., however, only every second and fourth sampling values are to be plotted.
e. Save the generated data in a data file (hint: use the command "save")
f. Replace the first example loop in the section *Loops and conditions* by vector operations.

II. Using matrices

a. Define the matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

and the vectors $b = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$, $c = \begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$ and $d = \begin{pmatrix} 1 & 2 \end{pmatrix}$ as line vectors.

b. Calculate Ab^T ; bA ; b * c as scalar product and (!) component-wise.

c. Solve the linear system of equations $Ax=b^T$ (Hint: "help slash") and verify the solution by insertion.

d. Calculate $A_2\, d^T$, where $A_2 = \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix}$ (write A_2 as submatrix of A!).

III. 2D and 3D representation of matrices

a. Generate a two-dimensional Gaussian bell with a variance of 8 on a (25x25) matrix.
b. Plot the matrix in a 2D color representation with contours.
c. like b, however, in a 3D color representation.
d. **Optional task**: How can a. be solved without a FOR loop?

The examples in the menu ‚Help->Examples and Demos' are very instructive and show many tricks and knacks of Matlab programming (the source texts for all examples are included!). It is recommended to look through these examples.

3 Representation of numbers and numerical errors

Types of errors:

1. Errors in the input data (e.g. finite precision of measuring devices)
2. Roundoff errors
   - because of finite precision of the number representation on computers
   - because of finite precision of arithmetic/mathematical calculations on the computer
3. Range errors
4. Truncations errors
5. Error propagation (instable solutions in iterated algorithms)

Type 1 will not be treated here, because the measurement process is not a matter of Numerics. Before treating types 2-5 in detail, common error measures will be introduced.

3.1 Error measures

Definition: x^* be an approximation to x ($x,x^* \in$ A, A set of numbers, e.g. real numbers). Then

$$\Delta x = |x - x^*|$$
$$r(x) = \frac{|x - x^*|}{|x|} = \frac{\Delta x}{|x|}$$

are the **absolute error** and the **absolute relative error** of x^*.

Mostly, Δx is not known, but an **error bound** ε can be estimated, which indicates the maximum possible absolute error:

$$x^* - \varepsilon \le x \le x^* + \varepsilon \Leftrightarrow x = x^* \pm \varepsilon$$

In most cases the ”real“ or “true” number is not known, but only the approximation x*. Then the relative error is approximated as well:

$$r(x) \approx \frac{\Delta x}{|x^*|} \le \frac{\varepsilon}{|x^*|}$$

Example: If the measured value is 0.0123 ± 0.0005, then $\Delta x \le \varepsilon = 0.0005$. The measured value has two significant digits. The relative precision is $r(g) \le 0.0005/0.0123 \approx 0.04$.

3.2 Representation of numbers and roundoff errors

Digital computers cannot represent infinitely many different numbers, because only a finite number of information units (mostly stated in bits[2]) are available for representing a number. The set A of numbers that can be represented is called the set of machine numbers and generally is a subset of the real numbers. A depends on the exact kind of number representation on the computer and implicitly involves a **rounding operation** *rd(x)* stating how real numbers are approximated by machine numbers. It can be defined generally:

[2] 1 Bit corresponds to a ‘yes-no’ decision (0 or 1).

$$rd: \Re \to A,$$
$$rd(x) \in A: \|x - rd(x)\| \le \|x - g\| \forall g \in A$$

The absolute and relative approximation errors caused by the finite number representation are obtained by setting $x^* = \mathrm{rd}(x)$ in the formulas given above. In general, the result of an arithmetic operation is not an element of A, even if all operands are elements of A. The result must again be subjected to the rounding operation *rd(x)* in order to represent it. Thus, arithmetic operations can lead to additional roundoff errors, even if the result of the operation was calculated with infinite precision:

$$x, y \in A \Rightarrow rd(x+y) \in A \text{, but } (x+y) \text{ not necessarily} \in A$$

Digital computers work with different kinds of number representations and thus with different machine numbers. Special common representations of numbers as used in most digital computers are shown below.

3.2.1 Integers

Integers are mostly represented by the **two's complement**, representing each machine number *z* (z being in decimal format) by a set of bits as follows:

$$z \hat{=} \{\alpha_0, \alpha_1, ..., \alpha_{N-1}\} \quad , \quad \alpha_n \in \{0,1\}$$
with
$$z = -\alpha_{N-1} * 2^{N-1} + \sum_{n=0}^{N-2} \alpha_n 2^n \text{ with } \alpha_n \in \{0,1\}$$

An information unit of 1 bit („0" or „1") is required for each coefficient α_n , such that N bits are necessary to represent the number (common quantities for N being 8,16,32, and 64). Hence, the bit representation of several numbers for e.g. N=8 is:

Number	**Bit representation**							
+1	0	0	0	0	0	0	0	1
+2	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0
-1	1	1	1	1	1	1	1	1
+117	0	1	1	1	0	1	0	1
-118	1	0	0	0	1	0	1	0
Significance	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Key parameters of the two's complement are:
Range of numbers: $-2^{N-1} \le z \le 2^{N-1} - 1$

Error bound: $\Delta z \le \varepsilon = 0.5$

Relative error: $r(z) \approx \frac{\Delta x}{|z^*|} \le \frac{0.5}{|z^*|}$

The representation of integers has a constant absolute error and hence a relative error depending on the value of the number to be represented.

Remark: The name "two's complement" is explained by the fact that the transition from a positive to the negative number of equal absolute value results from performing a bit by bit complement in the bit representation (each "1" becomes "0" and vice versa) and then adding 1.

Remark: The advantage of the two's complement over other conceivable representations of integers is the fact that the addition of two numbers is done by simply adding bit by bit in the bit representation. It is not necessary to differentiate between the cases of positive and negative numbers. Example (note the carry-over):

+2	0	0	0	0	0	0	1	0
-1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	1

Remark: Due to the significance of each single bit in the bit representation multiplying by 2 means simply shifting all bits to the left by one position ("left shift") and accordingly, dividing by 2 means a "right shift".

3.2.2 Floating point numbers

A machine number g (g being in decimal format) is represented in the **floating point binary format** as follows:

$$g = (-1)^s * m * B^e \ , \ 1/B \le m < 1$$

with:

s	sign bit ("0" or "1"
m	mantissa
e	exponent
B	basis (fixed value)

Remark: The side condition 1/B≤ m< 1 makes the representation unique. This is called the normalized representation. For example, the number 0.5 with the basis B=10 can be represented as $0.5*10^0$ or $0.05*10^1$. The first representation is fixed by the side condition.

The 32bit **IEEE Floating-Point Format** is used on most digital computers. B=2 is used as a basis and the exponent *e* is represented as an 8 bit two's complement. A 23bit representation with the following bit significances is used for the mantissa m:

Bit	m_0	m_1	m_2	...	m_{21}	m_{22}
Significance	2^{-1}	2^{-2}	2^{-3}	...	2^{-22}	2^{-23}

Hint: The normalization condition causes the bit m_0 of the mantissa m to be always 1. Therefore, it is generally not saved at all.

Example: The number 3 can be represented as follows:

$$3 = \left(2^{-1} + 2^{-2}\right) * 2^2$$

Thus, the bit representation of the number 3 in the IEEE Floating-Point Format is:

s	e_0	e_1	E_2	e_3	e_4	e_5	e_6	e_7	m_0	m_1	m_2	...	m_{21}	m_{22}
0	0	0	0	0	0	0	1	0	1	1	0	...	0	0

Key parameters of this representatuion are:

Range of numbers (approx.): $-10^{38} \le g \le 10^{38}$

Be $g=m*B^e \in A$ a machine number:

Error bound: $\Delta g \le \varepsilon = \frac{1}{2} * 2^{-M} * B^e$, M : Number of bits of the mantissa

Relative error: $r(g) \approx \frac{\Delta g}{|g*|} \le \frac{\varepsilon}{m * B^e} \le \frac{\varepsilon}{1/2 * B^e} = 2^{-M}$

The floating point representation has a constant relative error. The error bound is called **machine precision**.

Hint: The machine precision is often defined as the smallest number ε > 0, which still yields a result >1.0 upon addition (1+ε).

Hint: Besides the 32bit IEEE Floating-Point Format (”float“), a corresponding 64bit IEEE Floating-Point Format (”double“) is available in most programming languages, which possesses more bits for exponent and mantissa and thus a higher precision and a larger range of numbers. Matlab internally works with the 64bit-format.

3.3 Range errors

The different representations of numbers cover a limited range of numbers (see above). This range can easily be exceeded by arithmetic operations. An important example is the calculation of the factorial *n!*, which exceeds the range of numbers of the 64bit floating point representation already for *n*>170. The range error caused by exceeding the range of numbers is bigger than the roundoff error by orders of magnitudes and must be avoided by choosing the appropriate representation of numbers and by checking the orders of magnitude of the numbers examined (**Example:** Calculation of the faculty via application of the Stirling formula and logarithmic representation).

Furthermore, it is possible that certain physical constants exceed or fall below the range of representable numbers (cf. Exercises).

3.4 Truncation errors

Errors inherent to the algorithm, even if no rounding errors occur. Occur in iterated algorithms,when the iteration is stopped after a certain number of iterations (**Examples:** Calculation of the exponential function by finite sums and approximation of the integral by rectangles of finite width).

3.5 Error propagation in arithmetic operations

Addition:

$$
\begin{aligned}
x &= a + b\,;\, a, b \ge 0\,, \\
x_{\max} &= a + \Delta a + b + \Delta b \\
x_{\min} &= a - \Delta a + b - \Delta b \\
\Delta x &= \Delta a + \Delta b \\
r(x) &= \frac{\Delta x}{x} = \frac{\Delta a + \Delta b}{|a + b|}
\end{aligned}
$$

This means that the absolute errors of the input values to the addition are added.

Subtraction:

$$
\begin{aligned}
x &= a-b\,;\, a,b \geq 0\,,\\
x_{\max} &= a+\Delta a-b+\Delta b\\
x_{\min} &= a-\Delta a-b-\Delta b\\
\Delta x &= \Delta a+\Delta b\\
r(x) &= \frac{\Delta x}{x} = \frac{\Delta a+\Delta b}{|a-b|}
\end{aligned}
$$

When two approximately equal numbers are subtracted, the relative error becomes very big! This is called **loss of significance, catastrophic roundoff error** or **subtractive cancellation**.

Example: Loss of significance in calculating the square formulas (→ Exercises).
Example: Look at the difference of the numbers p=3.1415926536 and q=3.1415957341, which are about equal and are both significant on 11 decimals. The difference p-q=-0.0000030805 has only five significant digits left, although the operation itself did not cause new errors.

Multiplication:

$$
\begin{aligned}
x &= a*b\,;\, a,b \geq 0\,,\\
x_{\max} &= (a+\Delta a)*(b+\Delta b) \approx a*b+\Delta a*b+a*\Delta b\,,\ \text{da } \Delta a*\Delta b \text{ much smaller than } \Delta a*b\\
x_{\min} &= (a-\Delta a)*(b-\Delta b) \approx a*b-\Delta a*b-a*\Delta b\\
\Delta x &\approx \Delta a*b+a*\Delta b\\
r(x) &= \frac{\Delta x}{x} \approx \frac{\Delta a*b+a*\Delta b}{|a*b|} = \frac{\Delta a}{|a|}+\frac{\Delta b}{|b|}
\end{aligned}
$$

Division:

$$
\begin{aligned}
x &= a/b\,;\, a,b \geq 0\,,\\
x_{\max} &= (a+\Delta a)/(b-\Delta b) = \frac{(a+\Delta a)*(b+\Delta b)}{(b-\Delta b)*(b+\Delta b)} \approx a/b+\frac{\Delta a*b+a*\Delta b}{b^2}\\
x_{\min} &= (a-\Delta a)/(b+\Delta b) = \frac{(a-\Delta a)*(b-\Delta b)}{(b+\Delta b)*(b-\Delta b)} \approx a/b-\frac{\Delta a*b+a*\Delta b}{b^2}\\
\Delta x &\approx \frac{\Delta a*b+a*\Delta b}{b^2}\\
r(x) &= \frac{\Delta x}{x} = \frac{\Delta x}{|a/b|} \approx \frac{\Delta a*b+a*\Delta b}{b^2*|a/b|} = \frac{\Delta a}{|a|}+\frac{\Delta b}{|b|}
\end{aligned}
$$

The absolute errors of the operands add up in addition and subtraction, while the relative errors add up in multiplication and division.

3.6 Error propagation in iterated algorithms

Many algorithms contain iterated reproductions, in which a set of calculations is repeated with the data calculated in the preceding step as long as a certain break condition is reached. The errors produced in one step propagate from step to step. Depending on the problem, the total error may increase linearly, increase exponentially or decrease exponentially (**error damping**). The behavior of an algorithm can be deduced from

the operations used in each iteration and the rules of error propagation in the fundamental arithmetic operation explained above.

Definition: *E(n)* be the total error following *n* iteration steps and ε the machine precision. If $|E(n)| \approx n\varepsilon$, the error propagation is called **linear**. If $|E(n)| \approx K^n\varepsilon$, the error propagation is **exponential**, which leads to **error damping** for $K<1$.

Example: Iterated reproductions lead to a strong dependence on the initial values in case of exponential error propagation and thus represent a path into deterministic chaos. An example for this is the **logistic parabola**.

3.7 Exercises

Remark: Save the solutions of all tasks in Script files ("M-Files").

I Roundoff errors

1. Determine the machine precision of Matlab and estimate the number of bits used to represent the mantissa. (**Hint**: Start at eps=1 and divide eps by 2 until (1+eps) is not larger than 1 any longer).
2. Calculate the formula x=(10+h)-10 for $h=B^{-n}$ for n=0...21 and B=10. Calculate the absolute relative error between x and h and plot it as a function of n. Repeat the same for B = 2. Why do we find distinct differences here?

II Range errors

1. How can the range error be avoided in calculations with physical constants (atom physics, quantum mechanics)?
2. Take eV (electron volt) as a unit for energy and the mass of the electron as a unit for mass. Convert 1kg, 1m, and 1 s into these units.

III Truncation errors

Taylor's series of the exponential function reads:

$$e^x = \lim_{N\to\infty} S(x,N) \text{ with } S(x,N) = \sum_{i=0}^{N} \frac{x^i}{i!}$$

Calculate the subtotals S(x,N) up to N=60 and plot the absolute relative error relating to the true value (Matlab function exp()) as a function of N. Test the program for x=10,2,-2 and -10. Why does the error increase for negative numbers?

IV Error propagation

1. The square equation $ax^2 + bx + c = 0$ has the solutions:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ and } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

a. Under which conditions do we find a loss of decimal places in the numerator?
b. Prove that the zeros can also be calculated according to the following equivalent formula:

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ and } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}$$

c. How can the loss of decimal places be avoided with b.?
d. Calculate the zeros for *a*=1, *b*=-10000.0001, and *c*=1 as well as for *a*=1, *b*=-100000.00001, and *c*=1 with both formulas. Compare the results with the correct solution and verify them by inserting into the equation (Hint: Use the command sprintf, otherwise Matlab will not represent enough decimal places (e.g. sprintf('%5.10g',x1))).

4 Numerical differentiation and integration

Numerical methods of differentiation and integration are applied when analytical solutions are not known. There are many examples of functions that are not integrable analytically, e.g. the Gaussian error function

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2} dy ,$$

hence, the integral over the Gaussian distribution. Contrary to integration, differentiation in the analytical way is virtually always feasible.Consequently, numerical differentiation itself is rarely applied directly. However, it is the basis for developing algorithms to solve ordinary and partial differential equations and is therefore described in detail below.

The most important numerical methods of differentiation and integration are explained in the following. The procedures of developing algorithms and of estimating numerical errors are generally applicable and will be found again in a complex form in later chapters.

4.1 Differentiation

The principle of numerical differentiation is to approximate the function to be differentiated by a polynomial, which can then be derivated according to the known rules for polynomials. The value of the derivative of the polynomial is used as an estimated value for the derivative of the function. In general the approximation and the estimated value for the derivative are the more precise, the higher the order of the polynomial. But this is not a law and there are exceptions, especially when the function to be derivated cannot be well approximated by polynomials of high orders.

4.1.1 Right-hand formula (”naive“ ansatz)

We know from analysis that the derivative is calculated according to the following equation:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

The limiting process cannot be done numerically, however, selecting an appropriate small h yields an approximation of the derivative according to the following equation:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \varepsilon_a \quad (h > 0),$$

ε_a being the (unknown) truncation error resulting from the omission of the limiting process (finite h). The truncation error occurs even if the quotient has been calculated with arbitrary precision. Numerical calculation of the quotient leads to the following additional sources of errors:

1. Round-off errors in calculating x+h.
2. Round-off errors in calculating the numerator (Attention: Loss of significance is possible.)

In order to avoid the first kind of errors h should be chosen such that $x+h$ is a number that can be exactly represented[3]. The second rounding error cannot be avoided. If the function f is calculated to the machine precision ε_m, the absolute error in the calculation of the derivative is as follows:

$$\Delta(f'(x)) = \frac{2\varepsilon_m}{h} + \varepsilon_a \quad (h > 0)$$

For calculating the truncation error we expand the function into a **Taylor's series** of the first order:

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(\zeta) \quad , \; x \le z \le x+h$$

ζ is a value within the observed interval according to Taylor's theorem. The equation above can be immediately derived by resolving for $f'(x)$:

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x)}{h} - \frac{1}{2}hf''(\zeta) \\ &= \frac{f(x+h) - f(x)}{h} + O(h) \end{aligned}$$

Hence, the truncation error is linear in h. This is called the **order** $O(h)$. The absolute error can then be stated as follows:

$$\Delta(f'(x)) = \frac{2\varepsilon_m}{h} + \frac{1}{2}Mh \quad , \quad M = \max_{x \le \zeta \le x+h}\left(\left|f''(\zeta)\right|\right)$$

The first addend increases with decreasing h, while the second decreases with decreasing h. Thus, there is an optimum h with minimal error, which depends on the maximum M of the second derivative in the observed interval. Since M is usually unknown, it appears to be rather pointless to choose h arbitrarily small because of the numerical errors. It is appropriate to start with a relatively large h and then reduce it until the estimation of the derivative does not change any longer referring to a predetermined precision.

4.1.2 Centered formula

It would be possible to improve the precision, if the truncation error decreased more than linearly with h, e.g. by the order $O(h^2)$ or higher. The object of developing such formulas is to use Taylor's series of higher orders and to incorporate the interval x-h as well:

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(\zeta_1) \quad , \quad x \le \zeta_1 \le x+h$$

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{6}h^3 f'''(\zeta_2) \quad , \; x\text{-}h \le \zeta_2 \le x+h$$

The two formulas are subtracted from each other and resolved for $f'(x)$. In doing so the second derivative is dropped (just like all further even orders would be dropped):

---

[3] First calculate the following temporary quantities: $\begin{aligned} \widetilde{x} &= x + h \\ \widetilde{h} &= \widetilde{x} - x \end{aligned}$. $x, \widetilde{x}$ and $\widetilde{h}$ are exactly representable numbers, as they were explicitly assigned to a variable in the workspace. Then calculate the formula with $x, \widetilde{x}$ and $\widetilde{h}$.

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \frac{1}{6}h^2 \frac{f'''(\zeta_1) + f'''(\zeta_2)}{2}$$
$$= \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

In this so-called **centered formula** the truncation error decreases with h^2, while the roundoff error is proportional to $1/h$ in calculating the quotient, as already shown for the right-hand formula. In general this leads to a smaller total error.
An estimation of the second derivative can be found in a similar way. For this purpose the Taylor expansion is again performed in the intervals $x+h$ and $x-h$ and then resolved for $f''(x)$ by adding the two series:

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + O(h^4)$$
$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{6}h^3 f'''(x) + O(h^4)$$
$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4)$$

$$\Rightarrow f''(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} + O(h^2)$$

Hence, the second derivative is calculated to the order $O(h^2)$ using the centered formula. However, the function f must be evaluated at three instead of two points.

4.1.3 Richardson extrapolation

Of course, formulas of higher order can be found in a way similar to that shown in the preceding section, but the formulas of higher orders can be calculated from the formulas of lower orders more easily. This procedure is called **extrapolation** and is described in the following. For this we observe the entire Taylor expansion of function f at the point x. An appropriate conversion yields the centered formula again, however, the truncation error is calculated in all powers of h:

$$f(x+h) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x)}{k!} h^k$$
$$f(x-h) = \sum_{k=0}^{\infty} (-1)^k \frac{f^{(k)}(x)}{k!} h^k \Rightarrow$$
$$f(x+h) - f(x-h) = 2\sum_{k=0}^{\infty} \frac{f^{(2k+1)}(x)}{(2k+1)!} h^{(2k+1)} \Rightarrow$$
$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \sum_{k=1}^{\infty} \alpha_k h^{2k} \quad , \quad \alpha_k = \frac{f^{(2k+1)}(x)}{(2k+1)!}$$

The series represents the (unknown) truncation error. The quotient is defined as $D_0(h)$ [4] in the left part of the last equation and is evaluated for the step sizes h and $2h$:

[4] $D_0(h)$ corresponds to the centered formula.

$$D_0(h) = f'(x) + \sum_{k=1}^{\infty} \alpha_k h^{2k}$$

$$D_0(2h) = f'(x) + \sum_{k=1}^{\infty} 4^k \alpha_k h^{2k}$$

The next higher power of h in the truncation error (k=1) can now be eliminated by appropriately combining $D_0(h)$ and $D_0(2h)$:

$$\begin{aligned} 4D_0(h) - D_0(2h) &= (4-1) f'(x) + 4\sum_{k=2}^{\infty} \alpha_k h^{2k} - \sum_{k=2}^{\infty} 4^k \alpha_k h^{2k} \\ &= 3f'(x) + \sum_{k=2}^{\infty} \tilde{\alpha}_k h^{2k} \end{aligned}$$

The next iteration to the approximation of $f'(x)$ is thus obtained as follows:

$$\begin{aligned} D_1(h) &\equiv \frac{4D_0(h) - D_0(2h)}{3} \\ &= D_0(h) + \frac{D_0(h) - D_0(2h)}{3} \\ &= f'(x) + \frac{1}{3}\sum_{k=2}^{\infty} \tilde{\alpha}_k h^{2k} \end{aligned}$$

Since the series for the truncation error starts only at k=2, $D_1(h)$ has a truncation error of the order $O(h^4)$ and is calculated from the two quantities $D_0(h)$ and $D_0(2h)$, which are only of the order $O(h^2)$ themselves. This can be continued in general in order to eliminate higher orders:

$$\begin{aligned} D_k(h) &= \frac{4^k D_{k-1}(h) - D_{k-1}(2h)}{4^k - 1} \\ &= D_{k-1}(h) + \frac{D_{k-1}(h) - D_{k-1}(2h)}{4^k - 1} \end{aligned}$$

This recursion formula is called **Richardson extrapolation** and can be used whenever the truncation error shows only even powers in h. The numerical differentiation is then obtained by calculating the following table:

$$\begin{array}{cccc} D_0(h) & . & . & . \\ D_0(h/2) & D_1(h/2) & . & . \\ D_0(h/4) & D_1(h/4) & D_2(h/4) & . \\ D_0(h/8) & D_1(h/8) & D_2(h/8) & D_3(h/8) \end{array}$$

The table can only be calculated row by row, except for the first column which is directly yielded according to the centered formula. The following columns then result from the extrapolation formula. The highest-order result is then found as the last value in each row (diagonal element). Calculation of the rows is stopped as soon as the row result does not change any longer (referring to a predetermined precision).

4.2 Integration

Consider the definite integral

$$\int_a^b f(x)dx$$

The essential technique for numerically calculating definite integrals is to approximate the function *f* in the observed interval [*a*,*b*] by a polynomial *P*, which is then integrated according to the simple rules of polynomial integration. The integral of the polynomial is then used as an estimate of the integral of the function:

$$f(x)\big|_{[a,b]} \approx P(x) \qquad , \qquad P(x) = \sum_{n=0}^{N} \alpha_n x^n$$

$$\int_a^b f(x)dx \approx \int_a^b P(x)dx$$

The higher the order *N* of the polynomial, the better the assessment of the integral value in general. However, this is only true for functions which can be approximated by polynomials of high orders (which is not always possible, e.g. in case of discontinuities). Thus, the problem is reduced to performing an appropriate polynomial approximation. Generally speaking, a polynomial of the *N*-th order can be determined from *N+1* functional values. For approximation of the function *f* by a polynomial of the *N*-th order it is sufficient to evaluate *f* at *N+1* points in the observed interval [*a*,*b*].

4.2.1 Trapezoidal rule

Let us assume that we divide the interval [*a*,*b*] into *N* intervals. For this *N*+1 points within the interval need to be defined:

$$x_0 = a, \quad x_N = b, \quad x_0 < x_1 < \ldots < x_{N-1} < x_N$$

The function *f* is evaluated at these points: $f_i \equiv f(x_i), \quad 0 < i < N$

The simplest method of integration is to linearly connect the functional values at these points piecewise. Thus, we obtain a trapezoid with the following area in each interval:

$$T_i = \frac{1}{2}(x_i - x_{i-1})(f_i + f_{i-1}), \quad 1 < i < N$$

The total area is then: $T = \sum_{i=1}^{N} T_i$

This formula is simplified, if the points are evenly distributed in the interval. Be $h = (b-a)/N$ the width of each interval, hence $x_i = a + ih$. The area of each interval is then $T_i = 1/2\, h(f_i + f_{i-1})$. Except the marginal points x_0 and x_{N1}, each f_i occurs twice in the formula for the total area:

$$T(h) = 1/2\, h(f_0 + f_N) + h\sum_{i=1}^{N-1} f_i \qquad \left(\equiv \sum_{i=0}^{N} w_i f_i \right)$$

According to this **trapezoidal rule** the integral can be approximated the better, the smaller *h* is (the occurring truncation error due to the finite interval is stated later on) (**Remark**: The estimate of the integral results as a weighted sum of the sampling points of the function).

4.2.2 Recursive trapezoidal rule

Now we want to iteratively calculate the aproximation of the integral according to the trapezoidal rule by systematically reducing the interval *h*. It is useful to bisect *h* each time proceeding from the complete interval (*h*=(*b*-*a*)):

$$h_n = \frac{1}{2^n}(b-a), \qquad n = 0,1,\ldots$$

The functional values f_i calculated in the preceding iteration step can be used again in this case. By simple consideration we obtain:

$$T(h_0) = \frac{1}{2} h_0 \big(f(a) + f(b)\big)$$

$$T(h_{n+1}) = \frac{1}{2} T(h_n) + h_{n+1} \sum_{i=1}^{2^n} f\big(a + (2i-1)h_{n+1}\big)$$

The series $T(h_n)$ converges towards the value of the integral.

4.2.3 Error analysis and Romberg integration

A procedure similar to that one demonstrated in section 4.1.3 shows that the truncation error of the trapezoidal rule contains only even powers of *h*:

$$T(h_n) = \int_a^b f(x) + \sum_{k=1}^{\infty} \alpha_k h_n{}^{2k}$$

Hence, the Richardson extrapolation for raising the order of errors is applicable again:

$$T_0(h_n) \equiv T(h_n) \qquad \text{(recursive trapezoidal rule)}$$

$$T_k(h_n) = \frac{4^k T_{k-1}(h_n) - T_{k-1}(h_{n-1})}{4^k - 1}$$

$$= T_{k-1}(h_n) + \frac{T_{k-1}(h_n) - T_{k-1}(h_{n-1})}{4^k - 1}$$

Numerical integration is then obtained by calculating the following table:

$$\begin{array}{llll} T_0(h_0) & . & . & . \\ T_0(h_1) & T_1(h_1) & . & . \\ T_0(h_2) & T_1(h_2) & T_2(h_2) & . \\ T_0(h_3) & T_1(h_3) & T_2(h_3) & T_3(h_3) \end{array}$$

The table can only be calculated row by row, except for the first column which directly results from the recursive trapezoidal rule. The columns result from the extrapolation formula. The highest-order result is then found as the last value in each row. Calculation of the rows is stopped as soon as the result of the row does not change any more (referring to a predetermined precision).

Calculation of the integral according to the recursive trapezoidal rule including additional Richardson extrapolation is called **Romberg integration**.

4.2.4 Gaussian integration

The trapezoidal rule includes the evaluation of the function f at points x_i, which are equidistantly distributed in the interval $[a,b]$. The estimated value of the integral then reads:

$$\int_a^b f(x)dx \approx T(h) = 1/2\, h\left(f_0 + f_N\right) + h\sum_{i=1}^{N-1} f_i, \qquad f_i = f\left(x_i\right)$$

Hence, it represents a weighted sum of values of the function f_i, the weights w_i being 0.5 in this case for the end points and 1 for the other points. The question arises of whether a nonuniform distribution of the values x_i yields a higher precision and if so, which weights are to be used for summing up. Here, just as in many other mathematical problems, Gauss provides us with an answer. He succeeded in showing that a certain selection of weights and sampling points x_i yields an approximation of f by a polynomial of the order $2N$, if the function is evaluated at N points only. Thereby the order doubles as compared to the trapezoidal rule. The values of w_i and x_i are found in tabular form for different N in mathematical tables of formulas. As mentioned above, a higher order of polynomials does not necessarily mean a higher precision. The former is only true for functions f that can be approximated by polynomials of high orders.

Extending the Gaussian integration allows for the integration of certain weighting functions into the integrands:

$$\int_a^b f(x)g(x)dx \approx \sum_{i=0}^{N} w_i f_i, \qquad f_i = f\left(x_i\right)$$

The values of w_i and x_i are found in tabular form for different weighting functions $g(x)$ and different N in mathematical tables of formulas. The following weighting functions are commonly in use:

Integration formula	Weighting function
Gauss-Legendre	$g(x) = 1$
Gauss-Chebychev	$g(x) = \left(1 - x^2\right)^{-1/2}$
Gauss-Hermite	$g(x) = e^{-x^2}$
Gauss-Laguerre	$g(x) = x^{\alpha} e^{-x}$
Gauss-Jacobi	$g(x) = (1-x)^{\alpha}(1+x)^{\beta}$

The advantage of these rules is that they are precise for integrands which can be written as a product of a polynomial and one of the weighting functions listed above.

4.3 Exercises[5]

1. Write a Matlab function deriv for numerically calculating the derivative of any arbitrary function *f* at a predetermined point *x* according to the right-hand and centered formulas. Bisect the interval *h* proceeding from h=1, until the value does not change any longer referring to a predetermined precision (e.g. 10^{-5}). (**Hint**: Write *f* as an independent Matlab function and call it to calculate the values of the function *f* within deriv).
2. Calculate the derivation of the following function numerically using deriv:

   a. $f(x)=sin(x)$, $x = 5/4*\pi$
   b. $f(x)=e^{-x}/x^2$, $x = 0.5$

   Plot the absolute error between the numerically calculated and the exact derivatives as a function of the interval *h*. Check whether the absolute error decreases with *h* (right-hand formula) and h^2 (centered formula), respectively (**Hint**: Use a double logarithmic representation).
3. (*) Improve the function deriv of Exercise 1. using Richardson extrapolation and calculate the derivatives 2.a and b anew. At which interval *h* is the precision equal to that yielded by the centered formula?

Use the Matlab function rombf for Romberg integration for the next exercises:

4. Determine π by calculating the integral over a quarter unit circle with a precision of 20 digits.

5. (*) In Fresnel's diffraction the Fresnel integrals play an important part:

   $$C(w) = \int_0^w \cos\left(1/2\pi \cdot x^2\right)dx \qquad \mathrm{S}(w) = \int_0^w \sin\left(1/2\pi \cdot x^2\right)dx$$

   Write a function for calculating *C(w)* and *S(w)*. Plot *S* against *C* for *w* = 0.0, 0.1, 0.2,..., 5.0. What does the plot represent?

[5] Exercises with (*) are optional.

5 Ordinary differential equations (ODE)

Differential equations (DE) are of special relevance in physics since Newton dropped the apple and described this event (and thus gravitation) by means of the differential calculus, which was new at that time. Ordinary differential equations (ODE) contain only derivatives for one quantity, e.g. the time *t*. Differential equations containing derivatives for several different quantities are called partial differential equations (PDE). Spring pendulum and planetary motion are examples of ODE. Schrödinger's equation and the wave equation are examples of PDE. Many DE cannot be solved analytically so that numerical methods are required again.

5.1 Mathematical formulation of the initial value problem

For deriving the general form of ODE we proceed from Newton's law $\underline{F} = m\underline{a}$, which states a relation between force and acceleration[6]. For a given force we can derive the equations of motion:

$$\underline{a} = \frac{d^2\underline{r}}{dt^2} = \frac{\underline{F}}{m}$$

This vectorial equation of the 2nd order (2nd derivation occurs) can be rewritten into a system of two equations of the 1st order by inserting a new intermediate variable. This is always feasible, but not plain in general, as there are several possibilities to define the variable. In this case it is advisable to use velocity as an intermediate variable:

$$\frac{d\underline{r}}{dt} = \underline{v}$$
$$\frac{d\underline{v}}{dt} = \frac{\underline{F}\left(\underline{r}(t), \underline{v}(t), t\right)}{m}$$

$$\underline{r}\left(t = t_0\right) = \underline{r}_0$$
$$\underline{v}\left(t = t_0\right) = \underline{v}_0$$

Now we have two equations of the first order. By integration of the two derivatives we obtain two integration constants, which are determined by selecting the initial conditions. The force can depend on position and velocity (and hence implicitly on time) and also explicitly on time. Let us assume that the problem is two-dimensional in the Cartesian coordinates *x* and *y*. Then we can convert the problem into a general form by introducing further variables:

$$\underline{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} \equiv \begin{pmatrix} r_x \\ r_y \\ v_x \\ v_y \end{pmatrix} \quad \text{and} \quad \underline{H} = \begin{pmatrix} H_1 \\ H_2 \\ H_3 \\ H_4 \end{pmatrix} \equiv \begin{pmatrix} v_x \\ v_y \\ F_x\left(\underline{z}\right)/m \\ F_y\left(\underline{z}\right)/m \end{pmatrix}$$

The explicit dependence on time of all quantities has been omitted. According to this definition the general form of an ODE as an **initial value problem** is then:

$$\frac{d\underline{z}}{dt} = \underline{H}\left(\underline{z}(t), t\right) \quad \text{with} \quad \underline{z}\left(t_0\right) = \underline{z}_0$$

[6] All vectorial quantities are underlined.

In the above equation, $\underline{z}$ is the required solution vector (in the simplest case it is a scalar function of t) and $\underline{H}$, a linear or nonlinear vectorial function in the arguments $\underline{z}$ and t. $\underline{z}_0$ is the initial value, t mostly represents the time. The function $\underline{H}$ can be interpreted geometrically as a vector field, which predetermines a direction for $\underline{z}$ at each point of the space spanned by $\underline{z}$ and t.
A given problem should be converted into the general form shown above. Different formulations may arise depending on the intermediate variables chosen. Appropriate intermediate variables are not always easy to find, as numerical problems may arise according to the individual choice which are not always assessable in the beginning. In case of doubt several formulations have to be found and calculated.

5.2 Simple methods

Simple approaches directly resulting from the derived formulas for numerical differentiation are shown in the following. The different formulations are very easy to calculate and they appear to differ relatively little. However, experiments show that they are optimally applicable to different problems. In general, it is not clear beforehand which method is suitable in which case, as the numerical errors are difficult to assess. Only intuition and experience can help here. The idea behind all of these methods is to discretize the derivatives by appropriate approximations and thus to calculate the values of the variables within a time step from the values of the variables in the preceding time step.

5.2.1 Euler's method

Let us proceed from the simple equations of motion again:

$$\frac{d\underline{r}}{dt} = \underline{v}$$
$$\frac{d\underline{v}}{dt} = \frac{\underline{F}\left(\underline{r}(t), \underline{v}(t), t\right)}{m} = \underline{a}\left(\underline{r}(t), \underline{v}(t), t\right)$$

Approximating the derivative according to the right-hand formula using a time step τ, we obtain:

$$\frac{\underline{r}(t+\tau) - \underline{r}(t)}{\tau} + O(\tau) = \underline{v}(t)$$
$$\frac{\underline{v}(t+\tau) - \underline{v}(t)}{\tau} + O(\tau) = \underline{a}\left(\underline{r}(t), \underline{v}(t), t\right)$$

or

$$\underline{r}(t+\tau) = \underline{r}(t) + \tau\underline{v}(t) + O\left(\tau^2\right)$$
$$\underline{v}(t+\tau) = \underline{v}(t) + \tau\underline{a}\left(\underline{r}(t), \underline{v}(t), t\right) + O\left(\tau^2\right)$$

Note that the order of the truncation error increases after multiplication by τ. Introducing the following notation simplifies the equations:

$$f_n \equiv f(n\tau) \quad ; \quad n = 0,1,2,\ldots$$

Hence it follows:

$$\underline{r}_{n+1} = \underline{r}_n + \tau\underline{v}_n + O\left(\tau^2\right)$$
$$\underline{v}_{n+1} = \underline{v}_n + \tau\underline{a}_n + O\left(\tau^2\right)$$

For the one-dimensional case this **Euler step** can be interpreted geometrically (Figure 1). Proceeding from the current state r_n the gradient of the derivative field v is determined at point r_n. With this gradient the new value r_{n+1} is determined by linear continuation to the next sampling point.

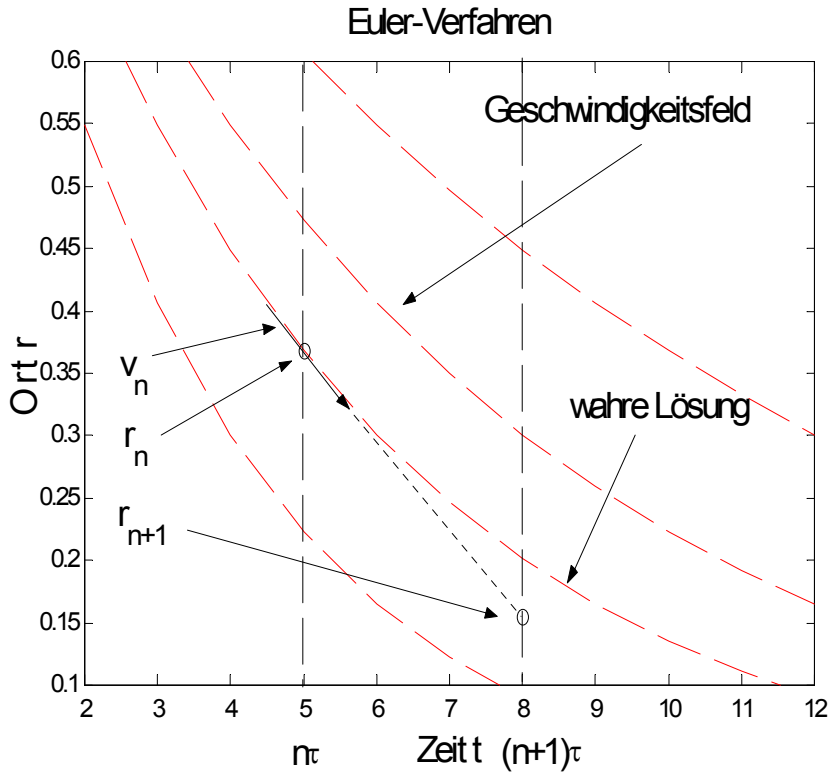


Figure 1: Geometrical interpretation of Euler's method (one-dimensional) on the basis of a simple system of solutions (exponential functions).

In order to calculate the trajectory as a series of discrete sampling points $n\tau$, we obtain the following algorithm:

1. Predefine the initial conditions $\underline{r}_0$ and $\underline{v}_0$.
2. Select a time step τ.
3. Calculate the acceleration from the current values of $\underline{r}$ and $\underline{v}$.
4. Apply Euler's method for calculating the new values of $\underline{r}$ and $\underline{v}$.
5. Return to step 3.

5.2.2 Euler-Cromer method

The following equations are a simple modification of Euler's method (not derived here):

$$\underline{v}_{n+1} = \underline{v}_n + \tau \underline{a}_n + O\left(\tau^2\right)$$
$$\underline{r}_{n+1} = \underline{r}_n + \tau \underline{v}_{n+1} + O\left(\tau^2\right)$$

The small alteration lies in the fact that the velocity newly calculated in the observed time step is used for calculating the new position vector. Note that the equations can only be calculated in the stated succession for this reason. This method has the same truncation error as Euler's method, but it works much better in some cases.

5.2.3 Midpoint method

The midpoint method is a mixture of Euler's method and the Euler-Cromer method:

$$\underline{v}_{n+1} = \underline{v}_n + \tau \underline{a}_n + O\left(\tau^2\right)$$
$$\underline{r}_{n+1} = \underline{r}_n + \tau \frac{\underline{v}_{n+1} + \underline{v}_n}{2} + O\left(\tau^3\right)$$

Here, the mean of the two velocity values are taken which raises the order of the position equation by 1. Inserting the first into the second equation yields:

$$\underline{r}_{n+1} = \underline{r}_n + \underline{v}_n \tau + \frac{1}{2}\underline{a}_n \tau^2$$

This equation is exact if the acceleration $\underline{a}$ is constant. This makes the method interesting in case of problems with a steady and slowly changing acceleration.

5.2.4 Verlet method

The Verlet method has the order 4 in the position equation and thus is especially suitable for problems in which this variable is of particular relevance. Moreover, the position equation can also be solved alone, if the force or acceleration depends on the position exclusively and the velocity is of no interest. For deriving the Verlet method we calculate with the first and second derivatives of the position and approximate them according to the centered formula for the first and second derivative, respectively:

$$\begin{aligned} \frac{d\underline{r}}{dt} &= \underline{v} \\ \frac{d^2\underline{r}}{dt^2} &= \underline{a} \end{aligned} \quad \Rightarrow \quad \begin{aligned} \frac{\underline{r}_{n+1} - \underline{r}_{n-1}}{2\tau} + O\left(\tau^2\right) &= \underline{v}_n \\ \frac{\underline{r}_{n+1} + \underline{r}_{n-1} - 2\underline{r}_n}{\tau^2} + O\left(\tau^2\right) &= \underline{a}_n \end{aligned}$$

A conversion yields:

$$\begin{aligned} \underline{v}_n &= \frac{\underline{r}_{n+1} - \underline{r}_{n-1}}{2\tau} + O\left(\tau^2\right) \\ \underline{r}_{n+1} &= 2\underline{r}_n - \underline{r}_{n-1} + \tau^2 \underline{a}_n + O\left(\tau^4\right) \end{aligned}$$

The disadvantage of this method is that it is not "self-starting", i.e., knowing the initial values $\underline{r}_0$ and $\underline{v}_0$ is not sufficient to iterate the trajectory. On the contrary, $\underline{r}_0$ and $\underline{r}_1$ must be known, which generally is not the case. In order to start the iteration, $\underline{r}_1$ can be calculated according to Euler's method and then be further iterated using the Verlet method.
Remark: If the acceleration does not depend on the velocity, the position equation can be iterated alone without calculating the velocity.

5.3 Global and local truncation error

Depending on the method applied, a truncation error occurs at each time step, e.g. of order $O(\tau^2)$ for Euler's method. This error is called **local error** in the following, as it occurs per time step. Considering a time interval of $T = N\tau$ for calculating the trajectory (hence N time steps), we maximally obtain the following **global error** in the observed variable at the end of the time interval:

$$\begin{aligned} \text{global error} &\propto N \times \text{local error} \\ &= NO\left(\tau^n\right) = \left(T/\tau\right)O\left(\tau^n\right) = TO\left(\tau^{n-1}\right) \end{aligned}$$

For example, Euler's method has a local truncation error of the order $O(\tau^2)$, but a global truncation error of only $O(\tau)$. This equation yields only an estimate of the global error, because it is not clear, whether the local errors accumulate or cancel (i.e. interfer constructively or destructively). Obviously this depends on whether the vector field of the derivatives in the observed range of variables has points of inflection and if so, how many of them and also on the kind of approximation (thus on the method applied). This missing knowledge of the error accumulation/prpagation implies that it is generally impossible to indicate, whether a certain method is suitable for a certain problem.

5.4 Runge-Kutta method of the 4th order

To get solution formulas of a higher order we have to leave the empirical path of finding simple solutions. For this purpose we proceed from the general form of the initial value problem:

$$\frac{d\underline{z}}{dt} = \underline{H}\left(\underline{z}(t), t\right) \quad \text{with} \quad \underline{z}\left(t_0\right) = \underline{z}_0$$

The simple approaches estimate the multi-dimensional derivative vector $\underline{H}$ from the current state $\underline{z}$ at time t once and then calculate the state at the next sampling time $t+\tau$. For the one-dimensional case

$$\frac{dz}{dt} = H\left(z(t), t\right)$$

this can be interpreted geometrically as shown above. It is imaginable to calculate several possible values of H in the following way:

$$H_1 = H\left(z, t\right)$$
$$H_2 = H\left(z + \tfrac{1}{2}\tau\ H_1, t + \tfrac{1}{2}\tau\right)$$
$$H_3 = H\left(z + \tfrac{1}{2}\tau\ H_2, t + \tfrac{1}{2}\tau\right)$$
$$H_4 = H\left(z + \tau\ H_3, t + \tau\right)$$

The values H_i can only be calculated in the given succession. For this, an Euler step is performed twice at half the step size. In the second step, the derivative assessed in the first half Euler step is used. Additionally, a full Euler step is performed using the gradient assessed in the second half step. The values H_i can be interpreted geometrically (as shown in Figure 1 for the simple Euler step) (cf. Fig. 2).

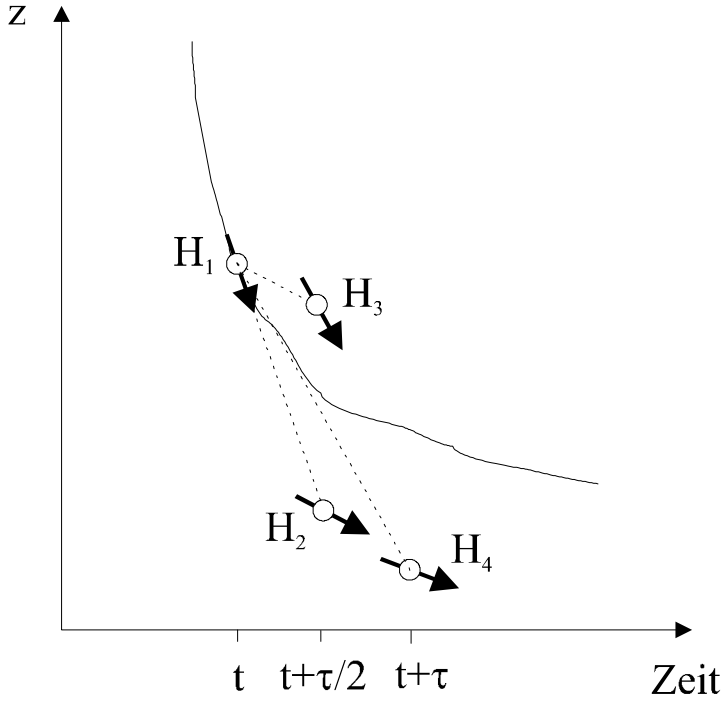


Fig. 2: Calculation of the estimates of the derivatives for the Runge-Kutta method of the 4th order (cf. text).

The question arises of how the state at the next sampling time (t+τ) is calculated from these four assessments of the derivative. It seems to be appropriate to take the weighted mean of the respective derivatives and then to perform an Euler step with this mean gradient. By means of the Taylor expansion (cf. textbooks) we can prove that the following weights are optimal, i.e., they yield a maximum order:

$$z(t+\tau) = z(t) + \tau \sum_{i=1}^{4} w_i H_i + O\left(\tau^5\right)$$

$$w_i = \left\{\tfrac{1}{6}, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{6}\right\}$$

Hence, the sum of weights is 1 as required. The gradients (H_2 and H_3) are weighted double. Altogether this procedure can be described vectorially for the general form of the initial value problem as follows:

$$\underline{z}(t+\tau) = \underline{z}(t) + \tfrac{1}{6}\tau \left(\underline{H}_1 + 2\underline{H}_2 + 2\underline{H}_3 + \underline{H}_4\right) + O\left(\tau^5\right)$$

$$\underline{H}_1 = \underline{H}(\underline{z}, t)$$
$$\underline{H}_2 = \underline{H}\left(\underline{z} + \tfrac{1}{2}\tau\ \ \underline{H}_1, t + \tfrac{1}{2}\tau\right)$$
$$\underline{H}_3 = \underline{H}\left(\underline{z} + \tfrac{1}{2}\tau\ \ \underline{H}_2, t + \tfrac{1}{2}\tau\right)$$
$$\underline{H}_4 = \underline{H}\left(\underline{z} + \tau\ \ \underline{H}_3, t + \tau\right)$$

The Runge-Kutta method described here is of the 4th order in each variable (i.e. in each component of $\underline{z}$) and therefore has a local truncation error of the order τ^5. Of course, formulas of even higher orders could be found in the same way, but they are more complex, i.e. require more evaluations of the function $\underline{H}$ per time step. Runge-Kutta of the 4th order has turned out to be the best compromise between precision and complexity. All textbooks agree that it represents the „workhorse“ for solving the ODE, especially when it is combined with an adaptive step size selection with a time step τ varying according to the assessed local truncation error (cf. next section). Furthermore, the so-called **predictor-corrector methods** enable the local errors and the step sizes to be controlled more precisely and also enable a correction term to be calculated for a higher precision. These methods will not be described in this context. Please refer to relevant textbooks.

5.4.1 Adaptive step size selection

The step size should be selected such that it is a fraction of the system-inherent time scales (e.g. 1/50th of the oscillation period of the system)[7]. Nevertheless, the system can reach states with strong differences in the absolute value of the derivative field in the course of time development. In case of large derivatives (i.e. large changes of the state of a system), a small time step has to be used in order to keep the local error below a predetermined bound. Within ranges of slowly changing states of a system, larger step sizes are sufficient. The number of states to be calculated can be minimized by adaptive step size selection. The additional time required for calculating the step sizes is less than the saved calculation expenditure in most cases.
The object of the adaptive calculation of step sizes is to assess the local truncation error for each time step anew and to select the step size such that it keeps below a predetermined bound. For this purpose the new state of a system is calculated twice, once via one step with the current step size and once via two steps with half the step size:

$$\underline{z}(t) \rightarrow \quad \rightarrow \quad \rightarrow \underline{z}_1(t+\tau)$$
$$\underline{z}(t) \rightarrow \underline{z}\left(t + \tfrac{1}{2}\tau\right) \rightarrow \underline{z}_2(t+\tau)$$

The local truncation error is assessed as the difference of both values:

$$\Delta\underline{z} \approx \left|\underline{z}_2 - \underline{z}_1\right|$$

Then the relative error can be assessed as follows:

[7] Systems with strongly differing time scales are problematic. The shortest time scale always determines the step size so that short step sizes have to be used, even if the long time scales predominate. These so-called **stiff ODE** are often resolved with implicit schemes (cf. textbooks).

$$r(\underline{z}) = \frac{\Delta \underline{z}}{|\underline{z}|} \approx \frac{|\underline{z}_2 - \underline{z}_1|}{\left|\frac{1}{2}(\underline{z}_2 + \underline{z}_1)\right|}$$

Since the local truncation error is proportional to the fifth power of the step size τ (Runge-Kutta 4th order), the current values of step size and truncation error can be related to the values to be expected upon selection of a new step size τ_{new}:

$$\left(\frac{\tau}{\tau_{new}}\right)^5 = \frac{\Delta \underline{z}}{\Delta} = \frac{r(\underline{z})}{r}$$

This equation can be written equally for the relative errors, because the scaling factors are cancelled.With a predetermined relative error bound r (e.g. 10^{-3}) the new step size can thus be calculated, which keeps within the predetermined error bound according to the relative error assessed before. Therefore, the equation is resolved for τ_{new}:

$$\tau_{new} = S\tau \left(\frac{r}{r(\underline{z})}\right)^{1/5}$$

An additional factor of safety $S \cong 0.9$ is incorporated such that the change will not become too large. Moreover, the new step size should not shift to higher or lower values by more than a predetermined factor (factor about 4 and ¼, respectively). Such a limitation is necessary, because the assessment and the quotient in the equation above may become singular.
The step size is repeatedly reduced per time step according to the given formula (including limitation), until the truncation error assessed with the new step size becomes smaller than the given truncation error. Vice versa, the step size is increased once only per time step. Thus, we reduce the danger that the time step becomes too large due to inaccurate assessment of the truncation error, which may extremely impair the precision. Assessing the time step too small will lead to a higher expenditure of work, but will not reduce the precision.

5.5 Application of various methods to solve initial value problems

The presented methods can be applied to different simple physical systems. Some of them are described in the following. The question of how these different methods behave when solving physical problems is investigated (partly by means of exercises).

5.5.1 Projectile motion

The trajectory of a ball is studied in two dimensions $\begin{pmatrix} x \\ y \end{pmatrix}$. We assume a gravitational force in negative y-direction and a frictional force proportional to the square of the velocity and directed against the current direction of motion. The equations of motion then read:

$$\frac{d\underline{r}}{dt} = \underline{v}$$

$$\frac{d\underline{v}}{dt} = \frac{\underline{F}(\underline{r}(t), \underline{v}(t), t)}{m}$$

$$= \frac{\underline{F}_a(\underline{v}) + \underline{F}_g}{m}$$

with

$$\underline{F}_a(\underline{v}) = -\frac{1}{2} c_w \rho \; A |\underline{v}| \underline{v}$$

$$\underline{F}_g = -mg \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

with:

$c_w \cong 0.35$ (drag coefficient)

$\rho \cong 1.2 \frac{kg}{m^3}$ (density of air)

$A\left[m^2\right]$ and $m\left[kg\right]$ as the cross-sectional area and mass of the ball

$g\left[\frac{kg \cdot m}{s^2}\right]$ gravitational acceleration.

If the frictional force is negligible, the analytical solutions are known. If the ball is thrown with an original velocity v_0 at an anagle Θ to the horizontal line, the throwing range, maximal height, and flying time are:

$$x_{\max} = \frac{2v_0^2}{g} \sin\Theta \cos\Theta$$

$$y_{\max} = \frac{v_0^2}{2g} \sin^2\Theta$$

$$T = \frac{2v_0}{g} \sin\Theta$$

This limiting case can be used to verify the numerical solution. Moreover, a physically reasonable step size can be selected as a fraction of the expected flying time. The projectile motion can be integrated using Euler's method (cf. Exercises).

5.5.2 Physical pendulum

We look at a pendulum, which is modeled as an ideal mass point of the mass *m* and oscillates around an axis by means of a massless arm of the length *L*. The equation of motion for this system reads (see textbooks):

$$\frac{d\Theta}{dt} = \omega$$

$$\frac{d\omega}{dt} = -\frac{g}{L} \sin\Theta$$

g being the gravitational acceleration. The angular acceleration is a nonlinear function of the elongation and there is no analytical solution. For small excursion angles Θ, however, the equation can be linearized:

$$\sin\Theta \approx \Theta \Rightarrow \frac{d\omega}{dt} = -\frac{g}{L}\Theta$$

For this mathematical pendulum we obtain the solution:

$$\Theta(t) = \exp\left(i2\pi\frac{t}{T_0}\right) \qquad , \qquad T_0 = 2\pi\sqrt{\frac{L}{g}}$$

If the ratio L/g is normalized to 1, the oscillation period T_0 equals 2π. The step size can then be adjusted as a physically useful fraction of the oscillation period. Using the law of conservation of energy it can be checked, whether the numerical solution is physically correct. For this purpose the total energy

$$E_{ges} = \frac{1}{2}mL^2\omega^2 - mgL\cos\Theta$$

is numerically calculated for each time step and plotted as a function of time (see Exercises).

Nonlinear effects are found at large elongation angles as a deflection of the oscillation form in relation to the sine function. In this case the oscillation period also depends on the maximum excursion angle θ_m (for derivation see next subsection):

$$\begin{aligned} T_0 &= 4\sqrt{\frac{L}{g}} \cdot K\left(\sin\left(\frac{1}{2}\theta_m\right)\right) \\ &\approx 2\pi\sqrt{\frac{L}{g}}\left(1+\frac{1}{16}\theta_m^2+\ldots\right) \end{aligned}$$

For smaller θ_m the formula turns into the above-mentioned formula for the linear case. K is the complete elliptic integral of the 1st category:

$$K(x) = \int_0^{\pi/2} \frac{1}{\sqrt{1-x^2\sin^2 z}}dz$$

If a frictional force proportional to the angular velocity as well as a harmonic excitation are introduced in addition to the nonlinear term of the restoring force, a path into **deterministic chaos** can be demonstrated by this simple system:

$$\begin{aligned} \frac{d\Theta}{dt} &= \omega \\ \frac{d\omega}{dt} &= -\frac{g}{L}\sin\Theta - \alpha\omega + A_0\sin(\omega_0 t) \end{aligned}$$

If the excitation amplitude is sufficient for the pendulum to overturn, this leads to a chaotic motion depending on damping and excitation. For further analysis, the trajectory of motion can be observed in the **phase space**, i.e. the space spanned by Θ and ω. If only points of the trajectory are plotted, which belong to a certain phase of excitation (e.g. zero phase), this leads to the so-called **Poincaré section**. If the motion is periodic there are only a few entries in the Poincaré section, the number of points indicating the multiplicity of the oscillation period relative to the excitation period. The path into chaos can then be understood with the help of the so-called period duplication (bifurcation): Plotting the Poincaré points (only Θ components) as a histogram as a function of the excitation amplitude, we obtain windows with periodic motions. With increasing excitation amplitude, the multiplicity of the period doubles. A multitude of such period duplications then leads to chaotic motions. This is an example of deterministic chaos: The underlying equation is well-defined and deterministic. Nevertheless, the motion is not predictable, because it extremely depends on the initial conditions. Additionally, the truncation and roundoff errors accumulate when pursuing the numerical calculation, leading to unpredictable global errors.

5.5.2.1 Derivation of the formula for the oscillation period

The undamped physical pendulum oscillates periodically. The maximum excursion be θ_m. At the stationary point, i.e. at $\theta = \theta_m$, the angular velocity ω equals 0. The total energy at this point then is

$E_{ges}\Big|_{\Theta=\Theta_m} = -mgL\cos\Theta_m$.

Due to the constancy of the total energy we obtain for all time points

$$\frac{1}{2}mL^2\omega^2 - mgL\cos\Theta = -mgL\cos\Theta_m$$

$$\Rightarrow \qquad \omega^2 = \frac{2g}{L}\left(\cos\Theta - \cos\Theta_m\right)$$

$$\Rightarrow \qquad \frac{d\Theta}{dt} = \sqrt{\frac{2g}{L}\left(\cos\Theta - \cos\Theta_m\right)}$$

$$dt = \sqrt{\frac{L}{2g}}\frac{d\Theta}{\sqrt{\cos\Theta - \cos\Theta_m}}$$

In the time period from $t = 0$ to $t = T_0/4$ (1/4 of the required oscillation period) the pendulum oscillates from $\theta = 0$ to $\theta = \theta_m$. Thus, both sides of the equation can be integrated:

$$\int_0^{T_0/4} dt = \sqrt{\frac{L}{2g}}\int_0^{\Theta_m}\frac{d\Theta}{\sqrt{\cos\Theta - \cos\Theta_m}}$$

$$\Rightarrow \qquad T_0/4 = 2\sqrt{\frac{L}{g}}\int_0^{\Theta_m}\frac{d\Theta}{\sqrt{\sin^2\left(\Theta_m/2\right) - \sin^2\left(\Theta/2\right)}}$$

with the replacement

$\cos 2\Theta = 1 - 2\sin^2\Theta$

An evaluation of the expression by means of the above-mentioned complete elliptic integral of the 1st category yields the above-mentioned formula for the oscillation period T_0.

5.5.3 Planetary motion

We observe the motion of a body in a central potential, e.g. a comet in the gravitation field of the sun. Neglecting all further forces (such as planets, solar wind etc.) the force affecting the body in the coordinate system of the sun is:

$$\underline{F}\left(\underline{r}\right) = -\frac{GmM}{\left|\underline{r}\right|^3}\underline{r}$$

m being the mass of the body, M the mass of the sun, and G the gravitation constant. It is useful to calculate in astronomical units. Then, $GM=4\pi^2$ AU^3/yr^2 with AU=1.496x10^{11}m being the average distance between sun and earth, and yr being one year. Furthermore, the mass m of the body can be set to 1. The equation of motion can now be integrated numerically. From Kepler's laws we know that the solutions are ellipses. Furthermore, total energy and angular momentum

$$E = \frac{1}{2}m\left|\underline{v}\right|^2 - \frac{GmM}{\left|\underline{r}\right|} \qquad , \qquad \underline{L} = \underline{r}\times\left(m\underline{v}\right)$$

are conserved. These physical boundary conditions allow for a verification of the numerical solution. Conservation of energy and angular momentum are also applicable to the verification of numerical solutions of many-body problems, which cannot be solved analytically anymore.
Applying the different methods, we find that Euler's method fails for (approximately circular) orbits, while the Euler-Cromer method is particularly suitable. In case of eccentric motion the Euler-Cromer method fails, too. In

this case, the adaptive Runge-Kutta method of the 4th order is the favourable method, which also applies to three- or many-body problems.

5.5.4 Lorenz model

The weather was previously assumed to be unpredictable only because of the great number of variables. Consequently, the development of efficient computers nourished hopes that they would enable long-term forecasts. This hope had to be abandoned. We know today that the weather is intrinsically unpredictable, because the underlying equations are nonlinear and their solution reacts extremely sensitively to changes of the initial conditions ("butterfly effect"). This can already be demonstrated by means of systems with few variables. For example, E. Lorenz investigated a model with three variables which describes the convection of a liquid between two parallel plates with a constant difference in temperature:

$$\frac{dx}{dt} = \sigma \left(y - x\right)$$
$$\frac{dy}{dt} = rx - y - xz$$
$$\frac{dz}{dt} = xy - bz$$

The derivation of the model is found in textbooks (e.g. H.G. Schuster: "Deterministic Chaos"). Please note that x represents approximately the circular flow rate, y the temperature gradient between rising and dropping liquid volumes, and z the deviation of the course of temperature from the equilibrium (thermal conduction only). The parameters σ and b depend on the properties of the liquid as well as on the geometry of the plates. Typical values are $\sigma = 10$ and $b = 8/3$. The parameter r is proportional to the imposed temperature gradient.
In order to investigate the sensitivity of the system to the initial conditions, it is the obvious thing to integrate the equations with two slightly different initial conditions (Runge-Kutta with adaptive step size) and to plot the time courses of the variables for comparison (cf. Exercises). Moreover, it is an obvious choice again to represent the trajectory in the phase space. It moves on an aperiodic orbit which is relatively well localized in the phase space, i.e. on an a**ttractor** (cf. Exercises).

5.6 Boundary value problems

So far, only the initial value problem has been treated: From a completely known initial state the temporal development is iterated. Of course, as many initial values as there are integration constants have to be specified. However, they do not need to be determined at a fixed time t_0. This leads to the so-called **boundary value problems**, in which part of the initial state at t_0 and part of the final state after a time T are given. For example, the locations at times t=0 and t=T may be predetermined for a projectile motion, but the velocity may be unknown. Then the above-mentioned formulas for calculating the temporal development cannot be directly applied any longer. Instead, assumptions about the complete initial state would have to be made, it would have to be iterated according to the above-mentioned formulas and varied until all boundary conditions were fulfilled. Of course, this means that the trajectory has to be calculated many times, until the right one is found. **Relaxation methods** are better suited to resolve boundary value problems. They are explained when treating partial differential equations (see below).

5.7 Exercises

1. Write a function ball, that numerically calculates the trajectory of a volleyball in two dimensions. Apply Euler's method to this problem. The initial conditions of the problem $\vec{x}(0)$ and $\vec{v}(0)$ should be stated as parameters. The trajectory should be represented graphically and the time of flight and throwing range should be calculated. Hint: A moving ball is decelerated by air friction as follows:

   $\vec{a}_{friction} = -0.5 c_w \rho \frac{A}{m} \vec{v}|\vec{v}|$ with

   $c_w \cong 0.35$ (drag coefficient)

   $\rho \cong 1.2 \frac{kg}{m^3}$ (density of air)

   $A\left[m^2\right]$ and $m\left[kg\right]$ as the cross-sectional area and the mass of the ball.

   Gravitation should not be neglected, either.

2. Calculate the oscillatory behaviour of a physical pendulum using Euler's and Verlet's methods. The function pendulum should expect the initial conditions as well as step size and kind of solution (Euler / Verlet). The elongation angle and the total energy in dependence on time are to be plotted.
   Recall the equations of motion:

   $$\frac{d\omega}{dt} = -\frac{g}{L} \cdot \sin(\Theta)$$

   $$\frac{d\Theta}{dt} = \omega$$

   The total energy of the system is:

   $$E = \frac{1}{2} m L^2 \omega^2 - m g L \cos(\Theta)$$

   Hint:

   Set $\frac{g}{L} = 1$ to simplify matters and observe the standardized energy $\frac{E}{m \cdot L^2}$.

   Examine the cases $\left[\Theta_0 = 10^\circ; \tau = 0{,}1\right]$ and $\left[\Theta_0 = 10^\circ; \tau = 0{,}05\right]$ using Euler's method and $\left[\Theta_0 = 10^\circ; \tau = 0{,}1\right]$ and $\left[\Theta_0 = 170^\circ; \tau = 0{,}1\right]$ using Verlet's method!

   What can we say about the suitability of the methods for this problem ?

3. Observe the harmonically driven, damped physical pendulum:

   $$\frac{d\Theta}{dt} = \omega$$

   $$\frac{d\omega}{dt} = -\frac{g}{L} \sin\Theta - \alpha\omega + A_0 \sin\left(\omega_0 t\right)$$

   a.) Integrate the system using Verlet's method. Set $\omega_0 = \sqrt{\frac{g}{L}} \equiv 1$ and calculate about 80 periods (t=0 - 80*2π). Write a plot program drawing the trajectory in the phase space (ω over Θ) as well as the elongation Θ and the total energy as a function of time.

   b.) Generate the following oscillatory behaviour by an appropriate selection of parameters:

   (i) Oscillation of a damped pendulum without excitation (Hint: Start with $A_0 = 0$ and vary the damping α until the pendulum dies off after about 5-10 periods from a starting position of Θ =90 Grad).

   (ii) Limit cycle (Hint: low damping and low excitation, oscillation without overturn).

(iii) Beating (Hint: no (!) damping and low excitation, oscillation without overturn).
(iv) Chaotic motion (Hint: Increase excitation up to and beyond an overturn).

Explain the different types of oscillations and document the sets of parameters found.

c.) Proceeding from b.), generate a Poincaré section by plotting only those points in the phase space that belong to the zero phase of excitation:

$$A = A_0 \sin\left(\omega_0 t^*\right) \quad , \quad \omega_0 t^* = n2\pi \quad , \quad n \in N \, .$$

Reject those points that are taken during the build-up time (about 20 periods, i.e. until t=40*π). What can we tell about the four cases of b.)?

d.) Plot the Poincaré points (Θ-components) as a histogram as a function of the excitation amplitude ("bifurcation diagram"). Generate a bifurcation diagram with excitation amplitudes between approx. 0 and 5 rad/s with a step size of 0.5 rad/s. Use the value of damping from exercise 1.b.i). Identify the interesting range of transition from periodic to chaotic motion and calculate this area with a solution of von 0.1rad/s anew. Describe the result.

4. Observe the body in the central potential. It is affected by a gravitational force:

$$\underline{F}\left(\underline{r}\right) = -\frac{GmM}{\left|\underline{r}^3\right|}\underline{r}$$

a.) Program the adaptive Runge-Kutta method of the 4th order. For this write a function
[x, t, tau] = rka(x,t,tau,err,derivsRK,param) for calculating a time step, with:
x = current value of the dependent variable
t = independent variable (usually time)
tau = current step size
err = desired local truncation error (appr. 10^-3)
derivsRK = right side of DE (name of function returning dx/dt
(calling format: derivsRK(t,x,param)).
param = extra parameter of derivsRK
x = new value of dependent variable
t = new value of independent variable
tau = recommended value for tau for next call of rka

b.) Integrate the system with Euler-Cromer and rka for a circular and an eccentric path. Plot each trajectory in the position space for 10 periods as well as the course of the total energy. How do the two methods differ?

c.) Draw a log-log plot of the step size determined for each step by rka as a function of the related radial distance from the origin for the eccentric path. Which law can be derived? (Hint: Kepler's laws).

5. Observe the Lorenz model

$$\frac{dx}{dt} = \sigma \left(y - x\right)$$
$$\frac{dy}{dt} = rx - y - xz$$
$$\frac{dz}{dt} = xy - bz$$

Integrate the system with rka. Plot x and z as a function of t as well as the trajectory in the phase space (zx- and yx-projection). Use the parameters σ=10, b=8/3 and r=28 as well as the initial conditions:
a.) x=1, y=1, z=20
b.) x=1, y=1, z=20.01

How do the paths belonging to the different initial conditions differ?

6 Solving systems of equations

The preceding section dealt with the solution of ordinary DE

$$\frac{d\underline{z}}{dt} = \underline{H}\left(\underline{z}(t), t\right) \quad \text{with the initial state} \quad \underline{z}\left(t_0\right) = \underline{z}_0$$

Starting from the initial state, the temporal development of the system can be determined as a series of states $\underline{z}\left(n\tau\right)$, e.g. by means of the Runge-Kutta method. So far, only **autonomous systems** have been treated in which the force and thus $\underline{H}$ does not explicitly depend on time, i.e. $\underline{H}\left(\underline{z}\left(t\right), t\right) = \underline{H}\left(\underline{z}\left(t\right)\right)$. For this class of systems there often exists a class of initial conditions $\underline{z}_0^*$ for which $\underline{z}\left(n\tau\right) \equiv \underline{z}_0^*$. These states in the *N*-dimensional phase space (state space) are called **stationary**. The following equivalence is easy to realize:

$$\underline{z}_0^* \text{ stationary state} \quad \Leftrightarrow \quad \underline{H}\left(\underline{z}^*\right) \equiv \underline{0} \quad ,$$

because the latter condition means $\frac{d\underline{z}}{dt} \equiv \underline{0}$. The vector equation $\underline{H}\left(\underline{z}^*\right) = \underline{0}$ contains *N* equations with *N* unknown quantities. Therefore, the roots of this system of equations have to be found in order to find the stationary states. This problem is divided into two classes. If $\underline{H}$ is a linear function in $\underline{z}$ or can be linearized in the surrounding of expected stationary states, methods of linear algebra can be used to solve the problem. If $\underline{H}$ is nonlinear (e.g. the physical pendulum comprises the nonlinear cosine-function), Newton's gradient method can be applied. Both classes of equations are explained in the following. Additionally, there are special methods in case $\underline{H}$ is a polynomial in $\underline{z}$, which will not be treated in this context (please refer to textbooks and to the function **roots** in Matlab, which finds zeros of polynomials).

6.1 Linear systems of equations

A linear system of equations generally consists of *M* equations with *N* unknown quantities:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N - b_1 &= 0 \\ &\vdots \\ a_{M1}x_1 + a_{M2}x_2 + \ldots + a_{MN}x_N - b_M &= 0 \end{aligned}$$

x_i being the unknown quantities to be determined and a_{ij} and b_i the (fixed) coefficients. This system can be written in a matrix form[8]:

$$\underline{\underline{A}}\,\underline{x} - \underline{b} = \underline{0} \quad \text{or} \quad \underline{\underline{A}}\,\underline{x} = \underline{b} \quad ,$$

with the definitions:

$$\underline{\underline{A}} = \begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MN} \end{pmatrix}, \quad \underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \quad \text{und} \quad \underline{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{pmatrix}$$

$\underline{\underline{A}}$ being the (*M*x*N*) matrix of the coefficients, $\underline{x}$, the (unknown) solution vector, and $\underline{b}$, the vector of the coefficients of the zeroth order. Example:

[8] Matrices are marked by a double underscore.

$$\begin{aligned} 2x_1 + x_2 - 4 &= 0 \\ 4x_1 - 2x_2 - 2 &= 0 \end{aligned} \quad or \quad \begin{pmatrix} 2 & 1 \\ 4 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

Regarding the dimensions of the problem (*M*x*N*) three cases are distinguished:

1. ***M>N***: The system is **overdetermined**, i.e. there are more equations than unknown quantities. In this case – if the equations are linearly independent – only an **approximate solution** is possible (see next chapter).
2. ***M<N***: The system is **undetermined**, i.e. there are less equations than unknown quantities. The solutions $\underline{x}$ lie within a **solution space** of the dimension *N* -*M*.
3. ***M=N***: The system has a **well-defined** solution, if all equations are linearly dependent (determinant of $\underline{\underline{A}}$ being not equal to zero). If equations are linearly dependent, case 2 applies.

In the following all three cases are treated. For case 3 the elementary Gaussian and Gauss-Jordan methods are applied, while the method of singular value decomposition is presented for cases 1 and 2. The latter is widely applied in linear algebra.

6.1.1 Elementary operations

Basic algorithms for solving systems of equations (Gaussian and Gauss-Jordan methods, respectively), use operations which do not alter the result (i.e. the solution vector $\underline{x}$). These operations are called **elementary operations**. The following elementary operations are feasible:

	Operations	**Meaning**
1	Exchange of rows in $\underline{\underline{A}}$ and $\underline{b}$	Exchange the sequence of two equations
2	Form linear combinations of rows in $\underline{\underline{A}}$ and $\underline{b}$	Form linear combinations of equations
3	Exchange columns in $\underline{\underline{A}}$	Exchange variables **(see remark)**

Remark: Operation 3 means that the sequence of components in the solution vector $\underline{x}$ is permuted. The permutations have to be made undone in order to obtain the original solution (exchanging, for example, the components position and velocity would be a fatal error). If several operations of type 3 are performed, they must be made undone in the reverse order. This requires a corresponding "bookkeeping", i.e. memorizing the operations of type 3 performed.

Example:

$$\begin{aligned} 2x_1 + x_2 &= 4 \\ 4x_1 - 2x_2 &= 2 \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} 4x_1 - 2x_2 &= 2 \\ 2x_1 + x_2 &= 4 \end{aligned} \quad \text{1st operation}$$

$$\Leftrightarrow \quad \begin{aligned} 2x_1 + x_2 &= 4 \\ 6x_1 - x_2 &= 6 \end{aligned} \quad \text{2nd operation (2nd row equals 1st row plus 2nd row)}$$

$$\Leftrightarrow \quad \begin{aligned} x_1 + 2x_2 &= 4 \\ -2x_1 + 4x_2 &= 2 \end{aligned} \quad \text{3rd operation (exchange } x_1, x_2 \text{ in the solution vector)}$$

All systems of equations on the right-hand side have the same solution (considering the exchange of the solution components in case 3).

6.1.2 Gaussian elimination with backsubstitution

This method is suitable for case 1 (*M*=*N*). The underlying idea is to first convert the system by means of elementary operations such that the matrix $\underline{\underline{A}}$ takes on an upper triangular shape:

$$\begin{pmatrix} a'_{11} & \cdots & \cdots & a'_{1N} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a'_{NN} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_N \end{pmatrix}$$

Remark: The components of $\underline{\underline{A}}$ and $\underline{b}$ are written here with apostrophe, since the numerical values have changed due to the elementary operations used for finding the triangular shape.

When the system is in the stated triangular form, the components of the solution can be iteratively determined by means of **backsubstitution**:

$$x_N = b'_N / a'_{NN} \quad \text{, to be calculated directly from the last row}$$

$$x_{N-1} = \frac{1}{a'_{N-1\,N-1}} \left(b'_{N-1} - a'_{N-1,N} x_N \right) \quad \text{, to be calculated from the second last row}$$

$$x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{j=i+1}^{N} a'_{ij} x_j \right) \;,\; i = N-1, N-2, \ldots, 1 \quad \text{, general form}$$

All components of the solution can be calculated from the first equation and by iteration of the last equation. Proceeding from x_N , x_{N-1} is calculated etc.

6.1.2.1 Generating the triangular shape and pivoting

The triangular shape required for the backsubstitution can be generated as follows:

1. Start out from the diagonal element a_{11}.
2. Subtract the a_{1j} / a_{11}-fold of the first row from each row j below the diagonal element. This results in zeros in the 1st column below the diagonal element.
3. Proceed to the next column and continue at with 1, however, starting out from the diagonal element of the actual column.

The problem inherent to this method is caused by the numerical errors occurring upon division by the respective diagonal element. The element by which we divide is called the **pivot** and the (empirically founded) method for minimizing the error is called **pivoting**. Pivoting means that the rows below and the columns to the right of the currently observed diagonal element (including the row and column with the current pivot) are exchanged such that the biggest element becomes the pivot (elementary operations 1 and 3). Partial pivoting means that only rows are exchanged. Here we save the necessary "bookkeeping" when exchanging columns. Partial pivoting is empirically found to be sufficient in general.

Remark: If the pivot (following pivoting) reaches the range of machine precision, the procedure is numerically instable. If the pivot equals zero, this may indicate that the system of equations is really singular (determinant = 0), or that the value 0 has accidentally resulted from numerical inaccuracies. Hence, the problem cannot be clearly analysed on the basis of the quantity of the pivot. In case of a small pivot we can only state that the solution is instable **or** that the system may be even singular. The solution should be verified by evaluating the equation using the estimated solution in that case.

6.1.2.2 Iterative improvement of the solution

The described Gaussian method comprises many calculation steps so that calculation errors may accumulate. Therefore, the numerically found solution will generally deviate from the true solution by an amount larger than the machine precision. In the following a method is described that allows for an iterative improvement of the solution up to the order of magnitude of the machine precision. The approximate solution for the system of equations

$$\underline{\underline{A}}\,\underline{x} = \underline{b}$$

found using the Gaussian method be $\underline{x}$‘ and $\underline{x}$ be the (unknown) true solution. By insertion the solution can be verified:

$$\underline{\underline{A}}\,\underline{x}' = \underline{b}'$$

We define the numerical errors as:

$$\delta\underline{x} = \underline{x}' - \underline{x}$$
$$\delta\underline{b} = \underline{b}' - \underline{b}$$

$\delta\underline{b}$ is known (and is, as mentioned above, generally larger than the machine precision), while $\delta\underline{x}$ is unknown, because the true solution $\underline{x}$ is unknown. $\delta\underline{x}$ can, however, be calculated proceeding from the approximate solution and utilizing linearity:

$$\begin{aligned} & \underline{\underline{A}}\,\underline{x}' = \underline{b}' \\ \Leftrightarrow \quad & \underline{\underline{A}}\left(\underline{x} + \delta\underline{x}\right) = \left(\underline{b} + \delta\underline{b}\right) \\ \Leftrightarrow \quad & \underline{\underline{A}}\,\underline{x} + A\,\delta\underline{x} = \underline{b} + \delta\underline{b} \\ \Leftrightarrow \quad & \underline{\underline{A}}\,\delta\underline{x} = \delta\underline{b} \end{aligned}$$

The linearity of the system is utilized in the third equation. Because $\underline{x}$ is assumed to be the true solution, the first addends on both sides of equation 3 can be dropped. Resolving the system of equations written in the fourth row (applying the Gaussian method again), we obtain an approximate solution $\delta\underline{x}$‘ for the true error $\delta\underline{x}$. The improved estimate of the solution is then:

$$\underline{x}'' = \underline{x}' - \delta\underline{x}'$$

This method can be iterated until the error $\delta\underline{b}$ reaches the order of magnitude of the machine precision. For this purpose the improved solution $\underline{x}$‘‘ is again inserted into the system of equations, a new $\delta\underline{b}$ is calculated, a new $\delta\underline{x}$ is determined etc. The drawback of the described method is that the system of equations has to be resolved several times. This expenditure of time is only justified in case a particularly precise solution is required.

6.1.3 Gauss-Jordan method

Gaussian elimination with backsubstitution is the most favourable method for calculating the solution in case 1 (M=N) regarding the expenditure of time. However, if the inverse matrix $\underline{\underline{A}}^{-1}$ is to be calculated in addition, an extended procedure known as the Gauss-Jordan method is more appropriate. For this purpose the following matrix equation is defined:

$$\underline{\underline{A}}\left[\underline{x} \cup \underline{\underline{Y}}\right] = \left[\underline{b} \cup \underline{\underline{1}}\right]$$

with:

$$\left[\underline{b} \cup \underline{\underline{1}}\right] = \begin{pmatrix} b_1 & 1 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ b_N & 0 & \cdots & 0 & 1 \end{pmatrix} \text{ and } \left[\underline{x} \cup \underline{\underline{Y}}\right] = \begin{pmatrix} x_1 & y_{11} & \cdots & y_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ x_N & y_{N1} & \cdots & y_{NN} \end{pmatrix}$$

We can explain this formulation as follows: N+1 systems of equations are established simultaneously, each column in [$\underline{b}\cup\underline{1}$] and [$\underline{x}\cup\underline{Y}$] representing an individual, independent system of equations. These systems can be resolved simultaneously by converting matrix $\underline{\underline{A}}$ into a unit matrix using elementary operations (**Remark**: Each elementary operation then influences each column of [$\underline{b}\cup\underline{1}$]) separately, as described above for the case of one column. The right side of the equation then represents the unknown $\underline{x}$ which solves

$$\underline{\underline{A}}\,\underline{x} = \underline{b}$$

and the unknown $\underline{\underline{Y}}$ which solves

$$\underline{\underline{A}}\,\underline{\underline{Y}} = \underline{\underline{1}} \ .$$

Thus, $\underline{\underline{Y}}$ is the inverse of $\underline{\underline{A}}$.

6.1.3.1 Generating the unit matrix

The unit matrix can be generated with the following scheme which strongly resembles the Gaussian method:

1. Go through matrix $\underline{\underline{A}}$ column-wise, starting out from column 1. i be the current column.
2. Take the diagonal element a_{ii} as pivot and perform pivoting.
3. Divide row i by the pivot. Hence, the diagonal element assumes the value 1.
4. Set the values of all other rows in column i to zero by subtracting the a_{ji}–fold of the i-th row from the j-th row (for all rows j except the i-th row itself).
5. Proceed to the next column i+1 and continue with 2.

The notes from 6.1.1.1 on the numerical stability hold also for this extended method.

6.1.4 Singular value decomposition

As indicated above, the Gaussian and Gauss-Jordan methods are not suitable for systems of equations which are almost singular because of the repeated division by the respective pivot. Moreover, it is not suitable for overdetermined and underdetermined systems. In the following a method is presented that is numerically very stable and is suitable for all classes of systems of equations. Furthermore, it enables a clear diagnosis of the system's numerical stability. In the literature this method is mostly called **SVD** (singular value decomposition). SVD is based on a theorem of linear algebra the proof of which is not given here (refer to textbooks of linear algebra). It says that each MxN matrix $\underline{\underline{A}}$ with $M \geq N$ can be decomposed as follows:

$$\underline{\underline{A}} = \underline{\underline{U}} \cdot \begin{pmatrix} w_1 & & \\ & \ddots & \\ & & w_N \end{pmatrix} \cdot \underline{\underline{V}}^T = \underline{\underline{U}} \cdot diag\left(w_j\right) \cdot \underline{\underline{V}}^T$$

with:

$\underline{\underline{U}}$: MxN-matrix

$\underline{\underline{V}}$: NxN-matrix

w_j : singular values

$\underline{\underline{U}}$ and $\underline{\underline{V}}$ being column-wise orthonormal, i.e.:

$$\underline{\underline{U}}^T \cdot \underline{\underline{U}} = \underline{\underline{V}}^T \cdot \underline{\underline{V}} = \underline{\underline{1}}$$

$$\Leftrightarrow \quad \sum_{i=1}^{M} U_{ik} U_{in} = \delta_{kn} \qquad 1 \le k \le N,\ 1 \le n \le N$$

$$\sum_{j=1}^{N} V_{jk} V_{jn} = \delta_{kn} \qquad 1 \le k \le N,\ 1 \le n \le N$$

We will not deal with the details of how the decomposition is calculated. It is important to know that stable and efficient algorithms are available for this purpose. In the following it is shown, how SVD can be used to calculate the solution of linear systems of equations in the different cases.

Remark: The command **SVD** is available in Matlab. Attention: There are several variants of singular value decomposition. The variant presented here is calculated by the SVD command, if 0 is given as additional parameter: [U,S,V]=svd(A,0)).

Remark: SVD is also feasible in the case of *M*<*N* (underdetermined system), however, *N-M* of the singular values are then equal to zero and the column-wise orthonormality of $\underline{\underline{U}}$ and $\underline{\underline{V}}$ does not apply to those columns containing the zeros in the matrix of the singular values.

6.1.4.1 SVD of a quadratic matrix (case M=N)

In the case of *M*=*N* all matrices involved are quadratic. Then, row-wise orthonormality also applies to $\underline{\underline{U}}$ and $\underline{\underline{V}}$, i.e.:

$\underline{\underline{U}} \cdot \underline{\underline{U}}^T = \underline{\underline{V}} \cdot \underline{\underline{V}}^T = \underline{\underline{1}}$ is valid.

$\underline{\underline{U}}$ and $\underline{\underline{V}}$ thus are orthogonal and

$$\underline{\underline{U}}^T = \underline{\underline{U}}^{-1}$$
$$\underline{\underline{V}}^T = \underline{\underline{V}}^{-1}$$

Then, the inverse of matrix $\underline{\underline{A}}$ is yielded as

$$\underline{\underline{A}}^{-1} = \underline{\underline{V}} \cdot \begin{pmatrix} 1/w_1 & & \\ & \ddots & \\ & & 1/w_N \end{pmatrix} \cdot U^T = \underline{\underline{V}} \cdot diag\left(1/w_j\right) \cdot U^T$$

and the solution of the system of equations is:

$$\underline{\underline{A}}\underline{x} = \underline{b} \Rightarrow \underline{x} = \underline{\underline{A}}^{-1}\underline{b} = \underline{\underline{V}} \cdot diag\left(1/w_j\right) \cdot \underline{\underline{U}}^T \cdot \underline{b}.$$

This method will only go wrong, if one or several of the singular values become zero or lie within the order of magnitude of machine precision. The more singular values get small the more instable the solution. In the first place SVD enables a clear diagnosis of the situation owing to the analysis of singular values. Two cases are to be distinguished (ε_m: machine prescision):

(i) The matrix is **ill-conditioned**, i.e. $\dfrac{\min_{j\in 1,N}\left(\left|w_j\right|\right)}{\max_{j\in 1,N}\left(\left|w_j\right|\right)} < \varepsilon_m$

(ii) The matrix is **singular**, i.e. $\exists j : w_j = 0 \quad , j \in 1, N$

6.1.4.1.1 Case (i): „ill-conditioned“

Formally, there is a unique solution in case (i). However, the Gaussian and Gauss-Jordan methods are numerically instable in this case, whereas SVD yields a generally more robust and more precise solution in the following way (without proof):

$$\underline{x} = \underline{\underline{V}} \cdot diag\left(1/w_j'\right) \cdot \underline{\underline{U}}^T \cdot \underline{b} \quad \text{, with}$$

$$1/w_j' = \begin{cases} 0 & |w_j| < \varepsilon_m \\ 1/w_j & sonst \end{cases}$$

i.e., we simply eliminate those singular values the value of which is not exactly known. This is illustrated by the fact that we throw out those equations and their combinations which more or less contain round-off errors only by setting the $1/w_j$ to zero. The solution $\underline{x}$ found in this way is generally better in the least-squares sense than the solutions yielded by other methods (Gauss elimination, Gauss-Jordan), i.e.

$$r^2 = \left|\underline{\underline{A}}\underline{x} - \underline{b}\right|^2$$

becomes smaller. The proof is not given here.

6.1.4.1.2 Case (ii): singular

The case (ii) is mathematically more complex. The inverse matrix $\underline{\underline{A}}^{-1}$ does not exist in this case, however, the system of equations

$$\underline{\underline{A}}\underline{x} = \underline{b}$$

can have a solution space as solution. The equation

$$\underline{x} = \underline{\underline{V}} \cdot diag\left(1/w_j'\right) \cdot \underline{\underline{U}}^T \cdot \underline{b} \quad \text{, with}$$

$$1/w_j' = \begin{cases} 0 & w_j = 0 \\ 1/w_j & otherwise \end{cases}$$

yields an “as optimal as possible“ special solution (without proof). “As optimal as possible” means that

$$r^2 = \left|\underline{\underline{A}}\underline{x} - \underline{b}\right|^2$$

becomes minimal and that $\underline{x}$ has the minimum length of all vectors within the solution space. The solution space L is yielded as the sum of the so-called nullspace and the special solution:

$$L = \left\{\underline{x} + \underline{x}' : \underline{x}' \in N\right\}$$

The **nullspace** N is defined as follows:

$$N = \left\{\underline{x} : \underline{\underline{A}}\underline{x} = \underline{0}\right\}$$

Thus, the nullspace contains all vectors mapped on zero.

Remark: Those columns of $\underline{\underline{V}}$ containing zeros in the related matrix of singular values ($diag(w_j)$) span the nullspace (without proof).

Remark: Those columns of $\underline{\underline{U}}$ containing zeros in the related matrix of singular values ($diag(w_j)$) span the **range** (without proof). The range is the set of all vectors $\underline{b}$, for which the system of equations $\underline{\underline{A}}\underline{x}=\underline{b}$ has at least one

exact solution. If $\underline{b}$ lies within the range of $\underline{\underline{A}}$, the above-mentioned special solution is exact. If b lies outside the range, the least-squares condition applies.

Remark: The case (ii) mathematically corresponds to the underdetermined system ($M<N$), which therefore can be resolved in the same way.

6.1.4.2 SVD for overdetermined systems (case M>N)

In case there are more equations than unknown quantities there is generally no exact solution to the system of equations. SVD, however, finds the "optimal" approximate solution again (without proof):

$$\underline{x} = \underline{\underline{V}} \cdot diag\left(1/w_j\right) \cdot \underline{\underline{U}}^T \cdot \underline{b}$$

$$\text{with}: \; r^2 = \left|\underline{\underline{A}}\underline{x} - \underline{b}\right|^2 \quad \text{being minimal}$$

Remark: Again, singular values smaller than the machine precision may exist here. Replace them by zeros again in the matrix of inverse singular values.

6.2 Nonlinear systems of equations

When a system of equations cannot be linearized, the algebraic methods of linear algebra described above cannot be applied. There is no choice but to apply iterative methods, of which Newton's gradient method is described below as a simple example.

6.2.1 Newton's method

The gradient method searches the zeros of a system of equations by taking steps in the direction of the gradient starting from the current estimate of the zero. Let us assume that $\underline{z}^*$ is the solution of the system of equations

$$\underline{H}(\underline{z}) = \underline{0} \quad \text{with} \quad \underline{H} = \left(h_1(\underline{z}) \quad \ldots \quad h_N(\underline{z})\right) \quad \text{and} \quad \underline{z} = \left(z_1 \quad \ldots \quad z_N\right)$$

(**Remark**: $\underline{H}$ and $\underline{z}$ are row vectors here). Let us further assume that $\underline{z}_1$ is the estimate of the solution, which deviates from the true solution by $\delta\underline{z}$, i.e.

$$\delta\underline{z} = \underline{z}_1 - \underline{z}^* .$$

With this definition and assuming $\underline{z}^*$ to be the true zero, a Taylor expansion of the 1st order yields:

$$\underline{0} \equiv \underline{H}\left(\underline{z}^*\right) = \underline{H}\left(\underline{z}_1 - \delta\underline{z}\right) = \underline{H}\left(\underline{z}_1\right) - \delta\underline{z} \cdot \underline{\underline{D}}\left(\underline{z}_1\right) + O\left(\delta\underline{z}^2\right)$$

with the Jacobi matrix $\underline{\underline{D}}$:

$$\underline{\underline{D}}(\underline{z}) = \left\{D_{ij}\right\} \quad , \quad D_{ij} = \frac{\partial h_j(\underline{z})}{\partial z_i}$$

Neglecting the terms of higher order we thus obtain:

$$\underline{0} = \underline{H}\left(\underline{z}_1\right) - \delta\underline{z} \cdot \underline{\underline{D}}\left(\underline{z}_1\right) + O\left(\delta\underline{z}^2\right)$$

$$\Rightarrow \quad \underline{H}\left(\underline{z}_1\right) \approx \delta\underline{z} \cdot \underline{\underline{D}}\left(\underline{z}_1\right)$$

$$\Rightarrow \quad \delta\underline{z} \approx \underline{H}\left(\underline{z}_1\right) \cdot \underline{\underline{D}}^{-1}\left(\underline{z}_1\right)$$

$$\Rightarrow \quad \underline{z}_2 = \underline{z}_1 - \delta\underline{z} = \underline{z}_1 - \underline{H}\left(\underline{z}_1\right) \cdot \underline{\underline{D}}^{-1}\left(\underline{z}_1\right)$$

$\underline{z}_2$ is a new estimate of $\underline{z}^*$, which can be iteratively inserted into the last equation again.

6.3 Exercises

1. The matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 7 & 3 & 4 \\ 8 & 9 & 15 \end{pmatrix}$ and the vector $b = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$ be given. Solve the linear system of equations $Ax = b$ and calculate the inverse matrix A^{-1}. Use the supplied Matlab script gaussj.m, which performs the Gauss-Jordan elimination procedure. Try to understand the program and compare the result with the installed Matlab functions inv (calculation of the inverse) and \ (calculation of the solution of the linear system of equations; type "help slash" for hints on use).

2. The matrices

$$A = \begin{pmatrix} 1 & -5 & 3 \\ 2 & -10 & 0 \\ 2 & 8 & 8 \end{pmatrix}, \quad B = \begin{pmatrix} 10 & -8 & -9 \\ 6 & 10 & 2 \\ 1 & 7 & 3 \end{pmatrix} \text{ and } C = \begin{pmatrix} 4 & -6 & -2 & 5 \\ -8 & 7 & 5 & 7 \\ -1 & 4 & 0 & -10 \\ -6 & -1 & -4 & 0 \end{pmatrix}$$

be given. Find the inverse matrices using the script gaussj.m. Why does this go wrong in all cases? In which cases should a solution exist (calculate the respective determinants) and how could we still obtain a result with this script in these cases? (Hint: "pivoting")

3. The Lorenz model

$$\frac{dx}{dt} = \sigma\,(y - x)$$
$$\frac{dy}{dt} = rx - y - xz$$
$$\frac{dz}{dt} = xy - bz$$

be given.

a) Show that it has the following stationary states:

$x^* = y^* = z^* = 0$ and $x^* = y^* = \pm\sqrt{b(r-1)}, z^* = r - 1, if \sigma \neq 0$

b) Search the stationary states for r=28, σ=10 and b=8/3 using Newton's method. Perform the calculation for several different initial estimates of the stationary states. How does the solution depend on it?

Hint: Use the supplied routine newtn, which implements Newton's method. Then write a routine [H,D]=fnewt(z,p), which calculates the function H as well as the Jacobi matrix D from the current state z=(x,y,z) and the set of parameters p=(r,σ,b).

7 Modeling of data

Modeling of data is of great relevance when validating physical theories and models. In most cases some parameters of the physical models are to be fitted to measured data and it is to be investigated to what degree the fitted model describes the data. This procedure is illustrated by means of simple examples in the following. In particular the techniques of fitting parameters and of assessing the goodness of fit will be described.

7.1 General mathematical formulation of the fitting problem

Let us assume N samples of data pairs (x_i,y_i) and a model to be fitted which predicts a functional relationship between the dependent quantity y and the independent quantity x:

$$y = f(x,\underline{a}) \quad \text{with} \quad \underline{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_M \end{pmatrix},$$

$\underline{a}$ being a vector consisting of M model parameters[9]. Of course, there should be clearly more data pairs than model parameters, i.e. $N >> M$.

The activity of a radioactive isotope be given as an example:

$$y = f(x,\underline{a}) = a_1 \cdot e^{-a_2 x} \quad \text{with} \quad \underline{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

y corresponding to the activity, x is the time and the parameters a_1 and a_2 describe the scaling of activity and decay rate. The model describes an exponential decrease in activity.

To simplify the expressions used later on we also write the data in vectors:

$$\underline{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} \quad , \quad \underline{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

The **general procedure** for fitting the model is as follows:

1. Define a ”merit function“ K (or: “cost function”) describing the deviation of the model from the data. The larger the deviation of the model from the data, the larger the merit function. This monotonous relationship applies to the least-squares merit function for example, which is defined as the sum over all data pairs of the squared deviations of the values of dependent variables assessed by the model from the actual data, i.e.:
   $$K(\underline{x},\underline{y},\underline{a}) = \sum_{i=1}^{N} \left| y_i - f(x_i,\underline{a}) \right|^2 .$$
2. Minimize the merit function for a given data set by varying the parameter $\underline{a}$.

The result of the fitting should then be:

1. Statement of the best set of parameters $\underline{a}_{\text{opt}}$ (best fit) corresponding to the minimum of K.
2. Estimation of the error of parameters, i.e. statement of a range in which the ‘true’ parameters are found with a given probability.

[9] The problem is formulated here without limitation for one independent and one dependent variable. In case of several independent variables, all of them enter the model function as arguments. Likewise, the model may predict several dependent variables so that the model function would be a vector-valued function.

3. Statistical statement about the quality of the model (”goodness of fit“), i.e. clarification of the question of whether the model is suited to describe the data sufficiently.

The last two points are often omitted, maybe, because they are time-consuming and difficult to understand. However, they belong to a systematic model analysis. In the following methods of model fitting are described. In each case the questions of which is the best merit function, how the merit function can be minimized, and how the fit is to be assessed will be clarified.

7.2 Establishing the merit function: Maximum-Likelihood Methods

The Maximum-Likelihood concept (ML method) allows to derive well-defined merit functions *K*, which are adapted to the respective fitting problem. The idea is to calculate the probability *W* that the observed data are generated by the model starting out from a statistical model of the measuring process. The negative logarithm of this probability is then used as the merit function *K*:

$$K\left(\underline{x},\underline{y},\underline{a}\right) = -\log\left(W\right)$$

$$W : \text{probability that the observed data are generated by the model}$$

The set of parameters that minimizes *K* is stated as the best fit $\underline{a}_{\text{opt}}$:

$$\underline{a}_{opt} = \underset{\underline{a}}{\operatorname{argmin}}\left(K\left(\underline{x},\underline{y},\underline{a}\right)\right) \qquad \left(= \underset{\underline{a}}{\operatorname{argmax}}\left(W\left(\underline{x},\underline{y},\underline{a}\right)\right)\right)$$

The minimization of *K* corresponds to the maximization of the probability *W* due to the monotony of the –log(*x*) function. Why is this procedure not called “Method of maximal probability” instead of “Maximum-Likelihood method“? The answer to this question is that the statement of the probability *W* refers to data and not to the model parameters: If the model is correct, there is only one “true” set of parameters and to state the probability of the set of parameters makes no sense. Therefore, the expression **likelihood** of a set of parameters is defined as follows:

The likelihood of a set of parameters $\underline{a}$ corresponds to the probability W that the observed data are generated by the model with this set of parameters

In the following, examples are given of how *W* and *K*, respectively, can be calculated from assumptions on the statistics of the measuring process. In particular, the statistical prerequisites of the measuring process are explained which render the least-squares merit function optimal in the sense of the ML concept. Starting from that fact, extensions of the least-squares method are stated based on slightly changed statistical assumptions on the measuring process (Chi-square method and robust fit methods).

Remark: The ML method does not detect systematic errors of the measurement process, but only models random errors. Systematic errors lead to a misfit of parameters.

7.2.1 Least-Squares method

The Least-Squares method is derived on the basis of the following assumptions:

1. The measurement errors are distributed as a Gaussian distribution with a fixed variance σ, i.e. they vary around the “true“ value predicted by the model $y = f\left(x,\underline{a}\right)$.
2. All measurement errors are statistically independent, i.e. the random error of one observation is independent of the random error of another observation.

In this case the probability that an observation with a certain deviation from the predicted value is generated is:

$$W_i = \exp\left(-\frac{1}{2}\left(\frac{y_i - f\left(x_i,\underline{a}\right)}{\sigma}\right)^2\right)\cdot\Delta y \ .$$

On the basis of the statistical independence of the observations the total probability that all observations are generated by the given model then is the product of the probabilities of each observation:

$$W = \prod_i W_i = \prod_i \left[\exp\left(-\frac{1}{2}\left(\frac{y_i - f\left(x_i, \underline{a}\right)}{\sigma} \right)^2 \right) \cdot \Delta y \right] .$$

Remark: The term Δy is a constant term which turns the probability **density** (1st term of equation) given by the Gaussian distribution into a probability. It is constant and can be taken as equal for all samples.

The merit function then results as

$$K\left(\underline{x}, \underline{y}, \underline{a}\right) = -\log\left(W\right) = \sum_{i=1}^{N} \frac{\left(y_i - f\left(x_i, \underline{a}\right)\right)^2}{2\sigma^2} - N \log\left(\Delta y\right)$$

Since the scaling of the merit function is arbitrary (no change of the position of the minimum), the constant factor $2\sigma^2$ and the constant 2nd addend $-N \cdot \log(\Delta y)$ can be omitted. As expected, we obtain the **merit function of the Least-Squares method** and the optimum set of parameters derived from it:

$$\boxed{\begin{aligned} K\left(\underline{x}, \underline{y}, \underline{a}\right) &= \sum_{i=1}^{N} \left(y_i - f\left(x_i, \underline{a}\right)\right)^2 \\ \underline{a}_{opt} &= \underset{\underline{a}}{\operatorname{argmin}}\left(K\left(\underline{x}, \underline{y}, \underline{a}\right)\right) \end{aligned}}$$

Remark: This derivation demonstrates the statistical prerequisites underlying the Least-Squares method which is often presented descriptively. They are often, but not always, fulfilled in measurements. Therefore, this method should never be applied unchecked!

7.2.2 Chi-Square method

The Least-Squares method is generalized by admitting different variances for the different observations. Of course, they must be known beforehand[10]. Thus, there are triplets (x_i, y_i, σ_i) for each sample. In this case, the variance cannot be extracted as a constant factor in the formula for the Least-Squares method given above. Hence, we obtain as the function to be minimized and optimum set of parameters:

$$\boxed{\begin{aligned} \chi^2\left(\underline{x}, \underline{y}, \underline{\sigma}, \underline{a}\right) &= \sum_{i=1}^{N} \left(\frac{y_i - f\left(x_i, \underline{a}\right)}{\sigma_i} \right)^2 \\ \underline{a}_{opt} &= \underset{\underline{a}}{\operatorname{argmin}}\left(\chi^2\left(\underline{x}, \underline{y}, \underline{\sigma}, \underline{a}\right)\right) \end{aligned}} ,$$

$\underline{\sigma}$ being a vector of the variances corresponding to the observations. The merit function is called the **Chi-Square**. Since the Chi-Square method includes the Least-Squares method as a special case with constant variance, we will deal with the Chi-Square exclusively in the following.

7.2.3 Robust Fit methods

The assumption that the random errors are normally distributed represents a rather "rigid" marginal condition. The likelihood that a sample deviates from the model value by more than 5σ is infinitely low in this case. Therefore, the Chi-Square method will fail, if there is one single "outlier" among the observations. The merit function increases significantly, because the likelihood for this event is considered to be very low based on the assumption of the Gaussian distribution. The minimizing procedure will attempt to decrease the costs, i.e. to include the outlier in the model. Thus, its weight becomes too large in the merit function and the fit method does not react to outliers robustly. If it is not sure, whether the Gaussian distribution of random errors can be taken as

[10] For example, the random error may be proportional to the quantity y upon reading the measuring instrument.

a basis, it should be considered which other distribution may be more suitable. Having found one, we can determine the likelihood and thus the merit function according to the above-described scheme.

E.g., if outliers have to be expected, we can use a distribution that does not approach zero as fast as the Gaussian distribution. Then the likelihood of large deviations from the mean is not too low any longer. The **Lorentz distribution** is the obvious choice for this purpose:

$$W_i = \frac{1}{1 + \frac{1}{2}\left(\frac{y_{i-} f\left(x_i, \underline{a}\right)}{\sigma} \right)^2} \cdot \Delta y$$

It resembles the Gaussian distribution within the range $\pm 2\sigma$, but its slopes are not so steep. As a merit function for the ML method based on the Lorentz distribution we obtain:

$$\begin{aligned} K\left(\underline{x}, \underline{y}, \underline{a}\right) &= -\log\left(W\right) = -\log\left(\prod_i W_i \right) \\ &\propto \sum_{i=1}^{N} \log\left(1 + \frac{1}{2}\left(\frac{y_i - f\left(x_i, \underline{a}\right)}{\sigma} \right)^2 \right) \end{aligned},$$

The constant addend $-N \cdot \log(\Delta y)$ has been omitted again. This merit function is significantly less sensitive to outliers than the Chi-Square method and the Least-Squares method, respectively (cf. Exercises), however, it is based on an assumption about the distribution of measuring errors (Lorentz distribution), which is only empirically founded. At any rate, it is better to use theoretical arguments for choosing a distribution. For example, the Poisson distribution can be assumed to be the theoretically correct distribution for the random error of radioactive decay measurements (see Exercises).

Remark: If possible, the assumption of a Gaussian distribution should not be rejected carelessly, because it has the advantage that steps 2. and 3. of the fitting process (statement of a parameter interval where the "true" parameters are probably found and statement of fit quality, i.e. suitability of the model) can be solved statistically more "neatly" (see section further below in this chapter).

7.3 Minimization of functions

The merit function being determined, the question arises of how to minimize it. Two cases are presented here, which allow the majority of practical cases to be solved:

1. The Chi-Square is used as merit function **and**
   a) the model $f\left(x, \underline{a}\right)$ is linear in the parameters $\underline{a}$ (not necessarily linear in the data!). In this case, the minimization problem can be directly solved without time-consuming iterative algorithms (in the literature this case is usually called **"general linear least-squares"**)
   b) the model $f\left(x, \underline{a}\right)$ is nonlinear in the parameters $\underline{a}$. Then only iterative methods can be used to minimize the function.
2. The Chi-Square is not used as a merit function, i.e. distributions of the random error other than the Gaussian distribution are employed. In this case there is nothing left but iterative methods independent of whether the model $f\left(x, \underline{a}\right)$ is linear in the parameters $\underline{a}$ or is not.

All cases are covered in the following.

7.3.1 "General Least-Squares" method

The model is assumed to be linear in the parameters $\underline{a}$ for this method:

$f(x,\underline{a}) = \sum_{k=1}^{M} a_k f_k(x)$,

$f_k(x)$ being an **arbitrary** function of the independent variable *x*. It can also be nonlinear, e.g. a power function

$f_k(x) = x^{k-1}$.

Then the model function is a polynomial in *x*

$f(x,\underline{a}) = a_1 + a_2 \cdot x + a_3 \cdot x^2 + \ldots + a_M \cdot x^{M-1}$.

With the general formulation of a linear function in the parameters, the Chi-Square is:

$$\chi^2(\underline{x},\underline{y},\underline{\sigma},\underline{a}) = \sum_{i=1}^{N} \left(\frac{y_i - \sum_{k=1}^{M} a_k f_k(x_i)}{\sigma_i} \right)^2 .$$

With the definitions

$$\text{"design matrix"}\ \underline{\underline{A}} = \left\{A_{ij}\right\} \quad \text{with} \quad A_{ij} = \frac{f_j(x_i)}{\sigma_i} \quad (N \times M) - \text{matrix}$$

$$\text{vector of normalized observations} \quad \underline{b} = \begin{pmatrix} y_1/\sigma_1 \\ \vdots \\ y_N/\sigma_N \end{pmatrix}$$

$$\text{parameter vector} \qquad \underline{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_M \end{pmatrix}$$

the Chi-Square can be rewritten as follows:

$\chi^2(\underline{x},\underline{y},\underline{\sigma},\underline{a}) = \left|\underline{\underline{A}} \cdot \underline{a} - \underline{b}\right|^2$.

The Chi-Square can thus be converted in this case such that it corresponds to the quadratic distance between the vectors $\underline{\underline{A}}\,\underline{a}$ and $\underline{b}$. The optimum parameter vector

$$\underline{a}_{opt} = \underset{\underline{a}}{\operatorname{argmin}}\left(\chi^2(\underline{x},\underline{y},\underline{\sigma},\underline{a})\right) = \underset{\underline{a}}{\operatorname{argmin}}\left(\left|\underline{\underline{A}} \cdot \underline{a} - \underline{b}\right|^2\right)$$

exactly corresponds to the Least-Squares solution of the system of equations $\underline{\underline{A}}\,\underline{a} = \underline{b}$, because this solution just minimizes the quadratic distance between $\underline{\underline{A}}\,\underline{a}$ and $\underline{b}$. The solution is known using SVD:

$$\boxed{\underline{a}_{opt} = \underline{\underline{V}} \cdot diag\left(1/w_j\right) \cdot \underline{\underline{U}}^T \cdot \underline{b}},$$

$\underline{\underline{A}} = \underline{\underline{U}} \cdot diag\left(w_j\right) \cdot \underline{\underline{V}}^T$ being the SVD of the matrix $\underline{\underline{A}}$.

Using SVD it is possible to solve the minimization problem in one step without iteration. Therefore, this method is particularly efficient and the absolute minimum is found. Additionally, any possible numerical problems can be solved by treating the singular values (cf. chapter on systems of equations: Setting the reciprocal value to zero, if the singular value is too small).

Remark: The so-called **normal equations** are often found in textbooks as the solution of the "General-Least-Squares". They result from setting the derivatives of the Chi-Square to zero with respect to the parameters.

$$
\begin{aligned}
& 0 \overset{!}{=} \frac{\partial \chi^2}{\partial a_k} && \forall k = 1, \ldots, M \\
\Rightarrow \quad & 0 \overset{!}{=} \sum_{i=1}^{M} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{N} a_j f_j(x_i) \right] f_k(x_i) && \forall k = 1, \ldots, M \\
\Leftrightarrow \quad & \left(\underline{\underline{A}}^T \, \underline{\underline{A}} \right) \cdot \underline{a} \overset{!}{=} \underline{\underline{A}}^T \cdot \underline{b}
\end{aligned}
$$

The equation in the third row describes the M normal equations (second row) in a matrix form, using the above-mentioned definitions of the design matrix, the parameter vector, and the vector of the normalized data. The optimum parameter vector results from matrix inversion:

$$\underline{a}_{opt} = \left(\underline{\underline{A}}^T \, \underline{\underline{A}} \right)^{-1} \cdot \underline{\underline{A}}^T \cdot \underline{b} \ .$$

By this method an extremum of the Chi-square function is found. It has to be verified that it is really a minimum, which means an additional calculation step. Altogether, a solution via SVD appears to be more suitable than a solution via normal equations, because the former directly yields a minimum and is numerically very robust.

7.3.2 Methods for nonlinear fitting: The "Simplex" method

If the model is nonlinear in the parameters and/or if the merit function is not the Chi-square (cases 1.b. and 2.), there is nothing left but iterative methods to find the minimum in "mountains" of the merit function. There exist several methods which search the minimum proceeding from an initial set of parameters. The simplest method is the Newtonian gradient method again. Its disadvantage is that the partial derivatives of the model function f have to be calculated (which is not always feasible) and that the method, depending on the initial values, certainly ends up in the nearest minimum, which may be a local minimum and not the absolute minimum. The **Simplex method** is presented below as one of the methods that neither require a partial derivative nor inevitably end up in the nearest local minimum.

Remark: In Matlab, the function **FMINSEARCH** implements the Simplex method.

Remark: The Simplex method and related methods can probably "jump across" the nearest local minimum. None of these iterative methods guarantees that the absolute minimum of the merit function is found!

The Simplex method works as follows:

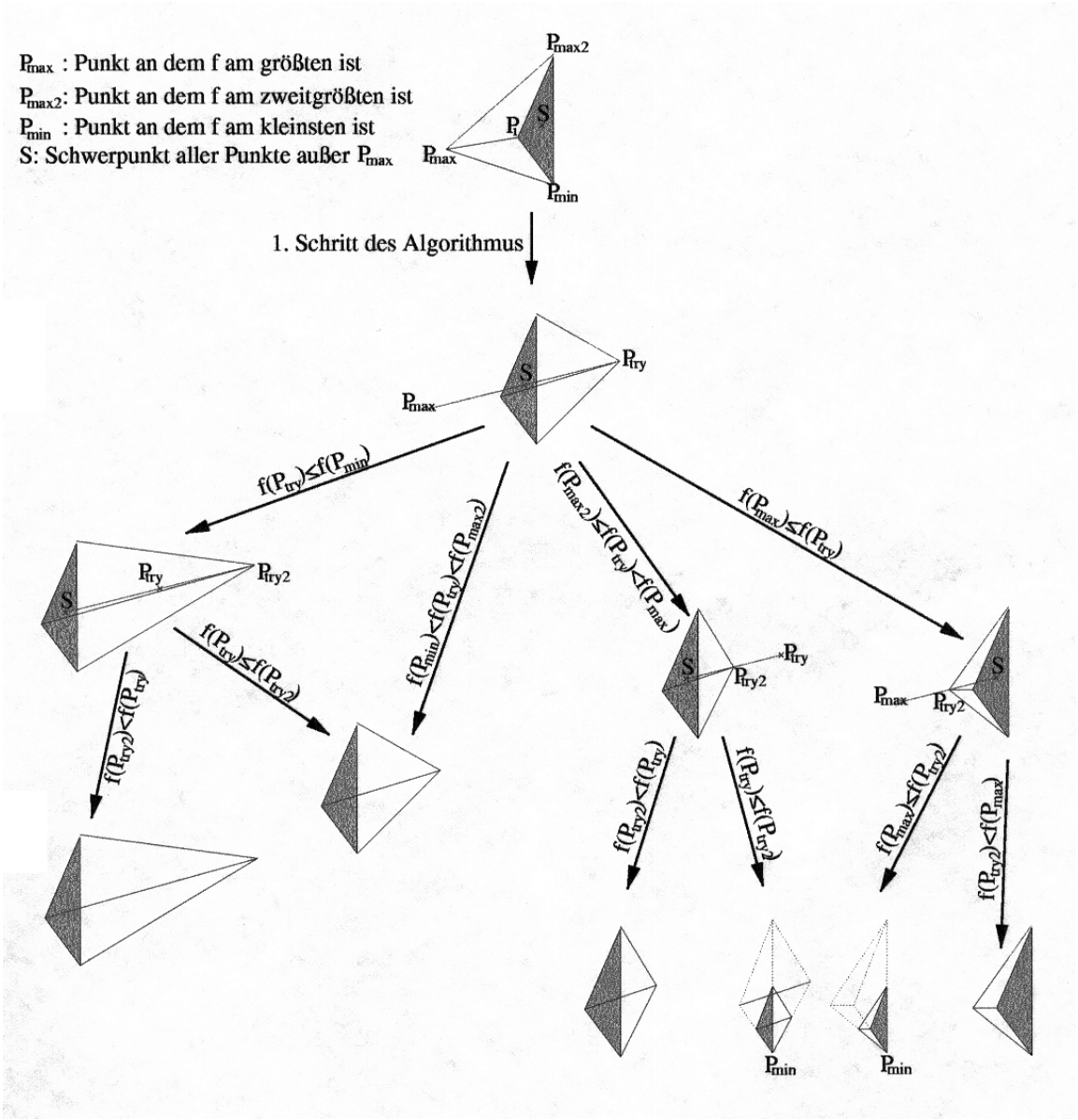
1. Generate a Simplex in the N-dimensional parameter space. A Simplex generally is a geometrical figure with (N+1) interconnected points (or vertices) in the N-dimensional space, e.g. a triangle in two dimensions. Distribute the vertices of the Simplex around an initial value of the parameters in the parameter space.
2. Calculate the merit function K on the vertices of the Simplex.
3. Depending on the result, apply one of the **elementary operations** described below, which cause a move of the Simplex in the parameter space.
4. Continue with 2., unless the termination criteria are fulfilled. The termination criteria are generally fulfilled, if the values of the merit function on the vertices of Simplex do not differ by more than a predetermined difference or if a given maximum number of iterations has been reached.

The following elementary operations can be performed:

1. Moving the vertex where the function is largest through the opposite face of the Simplex (reflection). Thus, the Simplex moves away from the maximum.

2. Like 1., however, with additional contraction of the Simplex in the dimension perpendicular to the reflection face.
3. Like 2., but expansion instead of contraction.
4. Contraction of the Simplex in one or more dimensions.
5. Expansion of the Simplex in one or more dimensions.

With each iteration the Simplex changes its position and size in the parameter space due to the elementary operations and thus moves “amoebia-like” in the direction of the minimum. The Matlab function FMINSEARCH uses empirical rules to choose the elementary operations depending on the distribution of the values of the merit function on the corners of the Simplex. The following figure illustrates the rules for the example of a tetrahedron (Simplex in three dimensions). Starting out from a current position of the Simplex, it is calculated what the Simplex looks like in the next step:



7.4 Error estimation and confidence limits

The optimal set of parameters $\underline{a}_{opt}$ which corresponds to the minimum of the merit function is determined by establishing and minimizing the merit function. Now the question of the error, or uncertainties, in a set of estimated parameters arises, i.e., how far does the estimated set of parameters $\underline{a}_{opt}$ deviate from the "true" set of parameters $\underline{a}_{true}$. For this purpose **confidence regions** are usually calculated stating a region within the parameter space, in which the true set of parameters is found with a predetermined probability. The limits of the confidence region are the confidence limits. For calculating the confidence regions, again different cases are to be distinguished (as for minimizing the merit function):

1. The Chi-square is used as a merit function **and**
   c) the model $f(x,\underline{a})$ is linear in the parameters $\underline{a}$ ("general linear least-squares"). In this case the confidence regions can be stated directly from the SVD solution.
   d) the model $f(x,\underline{a})$ is nonlinear in the parameters $\underline{a}$. Then the confidence regions must be calculated via repeated calculation of the Chi-square in the neighbourhood of the optimal set of parameters (iso-contours of the merit function).
2. The Chi-square is not used as merit function, i.e., distributions of the measurement error other than the Gaussian distribution are used. In this case only statistical methods can be applied to determine the confidence regions, which require the optimal set of parameters to be calculated from "synthetic"data sets (bootstrap method).

The different methods are explained in the following.

7.4.1 Chi-square fitting: Confidence regions of parameters

7.4.1.1 Model function $f(x,\underline{a})$ is nonlinear in the parameters

In this case the χ^2 function is evaluated on a grid within the neighbourhood of the optimum set of parameters and the iso-χ^2 curves are determined which correspond to a change of the χ^2 as compared to the minimum by different $\Delta\chi^2$. The following table shows $\Delta\chi^2$ values for several probabilities p of the true set of parameters to be found within the region delimited by the related iso-curve and for different numbers of parameters (without proof):

	M			
p / %	**1**	**2**	**3**	**4**
68.3	1.00	2.30	3.53	4.72
95.4	4.00	6.17	8.02	9.70
99	6.63	9.21	11.3	13.3

Example: The probability that the true set of parameters is found within the region delimited by the iso-curve with $\Delta\chi^2$ =6.17, i.e. the curve on which the value of χ^2 is by 6.17 larger than in the minimum, is 95.4% with two parameters (M=2). This region is called the **confidence region on the confidence level p = 95.4%**.

Remark: There are many statements about **confidence intervals** in the literature. They represent the projection of confidence regions on the axes of the parameter space. The $\Delta\chi^2$ values given in the table for M=1 are always valid for the confidence intervals (without proof). Example: A confidence interval lies on the 99% level, if it has been generated by projection of the confidence region with $\Delta\chi^2$ =6.63, independent of the dimension M of the parameter space.

Remark: The confidence regions are always ellipsoids in this case (see below).

7.4.1.2 "General linear least squares" case

In this case an analytical formula for the limits of the confidence regions can be given, i.e. it is not necessary to extract the iso-curves by calculating the Chi-square on a grid. The table given above is still valid for the confidence levels, however, there is an easier way to calculate the limits. If a variation $\delta\underline{a}$ of the parameter

vector proceeding from the optimum vector $\underline{a}_{opt}$ is given, the change of the Chi-square is calculated according to the following formula (without proof):

$$\Delta\chi^2 = \sum_{i=1}^{M}\left(w_i^2\left(\underline{\underline{V}}_{(i)}\cdot\delta\underline{a}\right)^2\right)$$

with

$\underline{\underline{V}}_{(i)}$: i - th column of the matrix $\underline{\underline{V}}$

$\delta\underline{a}$: variation of the parameter vector proceeding from the optimum set of parameters $\underline{a}_{opt}$

Thus, the columns of the matrix $\underline{\underline{V}}$ from the SVD of the design matrix $\underline{\underline{A}}$ form the axes of an ellipsoid representing the margins of the confidence regions (iso-χ^2 curves). The axis intercepts for the iso-curve at $\Delta\chi^2 = 1$ just represent the variance of the parameters (without proof):

$$\sigma^2(a_k) = \sum_{i=1}^{M}\left(\frac{V_{ki}}{w_i}\right)^2 \quad , \quad k = 1...M$$

These values represent the confidence interval of the parameter a_k on the 68.3% level.

7.4.2 The “Bootstrap“ method for estimating the confidence regions

In cases of non-normal distribution of measurement errors (merit function is not the Chi-square) the above-mentioned relation between confidence regions and confidence levels are not valid. In that case only empirical statistical methods can be applied to calculate the confidence regions, because analytical calculations are usually not feasible. A suitable empirical method is the "bootstrap“ method, which is explained in the following: From the available data set, new synthetic data sets are generated which have the same properties as the original data set in a statistical sense. Such a synthetic data set can be generated by drawing N data pairs (with replacement!) from the set of samples (N measured pairs). By drawing with replacement the data pairs of the synthetic data set are not identical with the measured data set and an arbitrary number of different synthetic data sets can be generated. Model fitting is then performed for each synthetic data set and the optimum set of parameters $\underline{a}_{opt}$ is determined which, of course, differs depending on the data set. Now the distribution of the determined sets of parameters is calculated and those regions are stated as confidence regions in which a certain percentage of sets of parameters are found.

Remark: The ”bootstrap“ method is only applicable, if the sequence of samples is not relevant and if the measurement errors of all samples have the same (non-normal) distribution. If this is not the case, time-consuming Monte-Carlo methods are to be applied, which are not treated here.

7.5 Goodness of fit

Having determined the optimum set of parameters and its confidence ranges, we have to investigate whether the model sufficiently describes the data (“goodness of fit“). The probability Q that the remaining deviations of samples from the model prediction result from random measurement errors is determined and it is investigated, whether the deviations are of a systematic nature. Systematic deviations mean that the model does not cover systematic effects in the measured data, i.e., its prediction capacity and quality are limited.

The goodness of fit is evaluated on the basis of the probability Q that the remaining deviations of measured values from the model prediction result from random measurement errors. The larger Q is, the larger is the goodness of fit.

If the Chi-square merit function is applied, results of mathematical statistics can be used in order to calculate Q. For this, we make use of the fact that a K-fold sum of squares of independent N(0,1)-distributed random variables (Gauss-distributed with mean 0 and variance 1) are $\chi 2_K$ –distributed (Chi-square-distributed with K degrees of freedom). The Chi-square does not exactly correspond to this condition, because the model function reduces the independence of random variables. Nevertheless, the Chi-square is assumed with good approximation to be $\chi^2_{(N-M)}$ –distributed with N samples and M model parameters. Q can be derived from this as follows (without proof):

$$Q = 1 - \frac{1}{\Gamma\left((N-M)/2\right)} \cdot \int_0^{\chi^2/2} y^{(N-M)/2-1} \exp(-y) dy$$
$$= 1 - gammainc\left(\chi^2/2, (N-M)/2\right)$$

The second addend represents the **incomplete Gamma function**. Q is directly calculated by inserting the observed Chi-square values at the minimum of the merit function as well as the number (N-M) of degrees of freedom into this formula.

Remark: The incomplete Gamma function is available in Matlab as the command **gammainc**.

On the basis of the value of Q the following statements on the model can be made:

Value of Q	**Conclusion**
small (Q~10^{-18})	The model is incorrect **or** the variance of random measurement errors has been estimated too small **or** the assumption of Gauss-distributed measuring errors is not correct.
Q~10^{-3}	Model OK
Q~1	Too good to be true. Maybe the variance of random measurement errors has been estimated too large.

Thus, it is obvious that the assumptions on the statistics of measurement errors must be correct for an "applicable" evaluation of the value of Q.

The value of the Chi-square can also be assessed with a **"rule-of-thumb"**. Assuming that the random deviation of data from the predicted model values is approximately 1σ on average, then each addend in the Chi-square function is approximately 1. The Chi-square thus takes approximately the value N for N addends (N samples). Simultaneously, the adjustment becomes easier with increasing number of parameters. This reduces the expected value of the Chi-square by about the number of parameters M. **The rule-of-thumb thus reads, that the value of the Chi-square should about equal the number of degrees of freedom *N-M*, if the remaining deviations of samples from the model prediction are of random nature and to make the model acceptable**. Hence, we obtain the following evaluation table:

Value of χ^2	**Conclusion**
χ^2>>N-M	The model is incorrect **or** the variance of random measurement errors has been estimated too small **or** the assumption of Gauss-distributed measuring errors is not correct.
χ^2~N-M	Model OK
χ^2<<N-M	Too good to be true. Maybe the variance of random measurement errors has been estimated too large.

Remark: For the other merit functions which are not based on a Gauss-distribution of measuring errors, Q and thus the goodness of fit are difficult to determine.

7.6 Summary of methods

The following table shows the different cases of model adjustment:

Case	Merit function	Minimization	Confidence ranges	Goodness of fit
Random measurement errors Gauss-distributed and model function linear in the parameters (”general least-squares“)	Chi-square	via SVD or normal equations	Calculation of iso-χ^2-curves via SVD	Chi-square distribution or rule-of-thumb
Measurement errors Gauss- distributed, but model function nonlinear in the parameters	Chi-square	Simplex method	Calculation of iso-χ^2-curves by sampling the parameter space	Chi-square distribution or rule-of-thumb
Measurement errors not Gauss-distributed	Merit function according to the Maximum-Likeli-hood principle based on the assumed distribution of measurement errors (e.g. Lorentz distribution (“robust“ fit) or Poisson distribution)	Simplex method	“Bootstrap“ method	Only feasible if the probability distribution of the values of the merit function can be derived, perhaps rule-of-thumb is applicable

7.7 Exercises

1. The supplied script linfit.m fits a straight line to test data. The χ^2-function is used as merit function and the Simplex method for function minimization (Matlab function FMINS).

   i. Try to understand the script linfit.m and alter the test data as well as the initial values of the parameters. Which influence do "outliers" have?
   ii. Alter the merit function in such a way that it is based on Lorentz-distributed measurement errors. How does the influence of outliers change?

2. In order to measure the half-life value of a radioactive isotope, the number of decays is counted every 10s for an interval of Δt=1s. The following values are measured:

Time / s	Aktivity (Number of decays)	Time / s	Aktivity (Number of decays)
10	24	60	5
20	17	70	4
30	11	80	2
40	10	90	3
50	6	100	1

   i. Estimate the half-life. Start from the model that the activity decreases exponentially:

   $$A(t) = A_0 e^{-\alpha t}$$

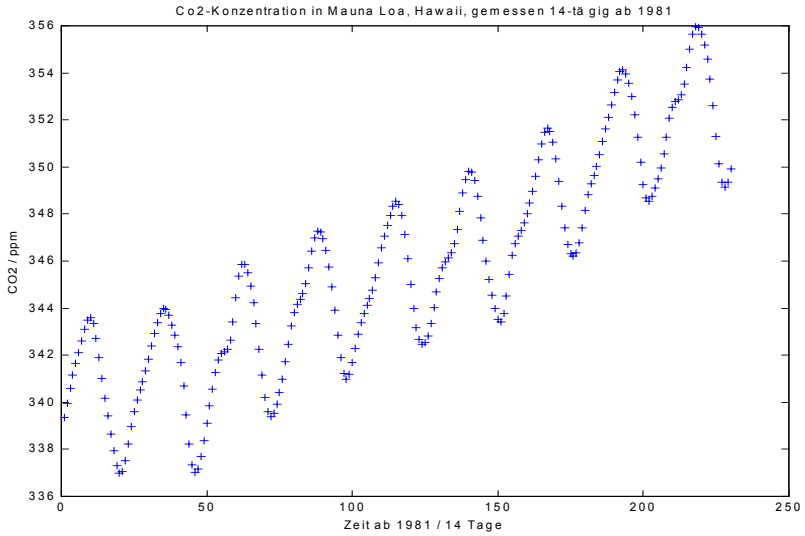
   Moreover, assume that the samples are Poisson-distributed:

   $$p(k,\lambda) = e^{-\lambda}\frac{\lambda^k}{k!}$$

   k being the number of decays and λ the expected number of decays (the expected value is the product of activity and counting interval: $\lambda = A(t) * \Delta t$)
   (Hint: The merit function to be minimized is available on request as Matlab script f_pois.m)

   ii. By which percentage does the estimated half-life value change, if Gauss-distributed samples are assumed (least-squares fit)?

3. The following figure shows the measured values of CO_2 concentration in Hawaii from 1981 on (registered two-weekly). The standard deviation of the measurements is about σ = 0.16 ppm.



Hint: The data are found in the file mauna.dat. Use the command „load mauna.dat –ascii" for loading the data into Matlab's workspace.

a) Determine the rate of increase in the CO_2 concentration in ppm/year. As a model use a linear increase in concentration. When will it be 15% higher than that of 1981 according to this model?
Use the Chi-square (σ = 0.16 ppm for all samples) as merit function and the Simplex method for minimization.

b) Like a), however, under the model assumption that the concentration increases quadratically.
Hint: The linear model function used in the script linfit.m can be easily extended to polynomials of the n-th order by simply stating n+1 initial values of the parameters.

c) Assess the goodness of fit of the models used in a) and b) by applying the "rule-of-thumb for the Chi-square. Can we rely on the results from a) and b) on the expected time of a 15% increase of the concentration?

d) Which simple extension of the quadratic model function would improve the model? Repeat the adjustment with this extended model function and assess the goodness of fit of this model using the "rule-of-thumb".

e) (*) Repeat the adjustment as performed in exercise d) using the general least-squares" method (with SVD).

8 Analog-Digital Transform and Discrete Fourier Transform

Numerical methods play an important role in the analysis of sampled signals (e.g. the time course of temperature and pressure or voltages and currents as functions of time). Among these methods, **spectral analysis**, i.e. the analysis of the frequency contents of a signal by means of **Discrete Fourier Transform (DFT)** is particularly worth mentioning and will be explained in the following. First, the process of **Analog-Digital Transform (A/D-Transform)** is described which is used to discretize analog signals in such a way that they can be processed on computers. On that basis the DFT as well as the Fast Fourier Transform (FFT) as a fast algorithm for calculating the DFT are introduced. Finally, the **filtering** of signals is described as an application of DFT. The so-called **convolution theorem** is of special importance in this context.

8.1 Analog-Digital Transform (A/D Transform)

Physical signals are generally continuous in time and value. For example, the time course of the pressure can generally be written as a real-valued function of continuous time $p = p(t)$. We speak of analog signals and continuous functions, respectively[11]. Since computers can only process finite quantities of numbers, these measured signals/functions have to be transformed into finite sequences of numbers:

$$x(t) \rightarrow \{x(n)\} \quad ; \quad n \in K \subset \mathbb{Z} \ .$$

The sequence consists of sampled values of the function at multiples of a predetermined period T:

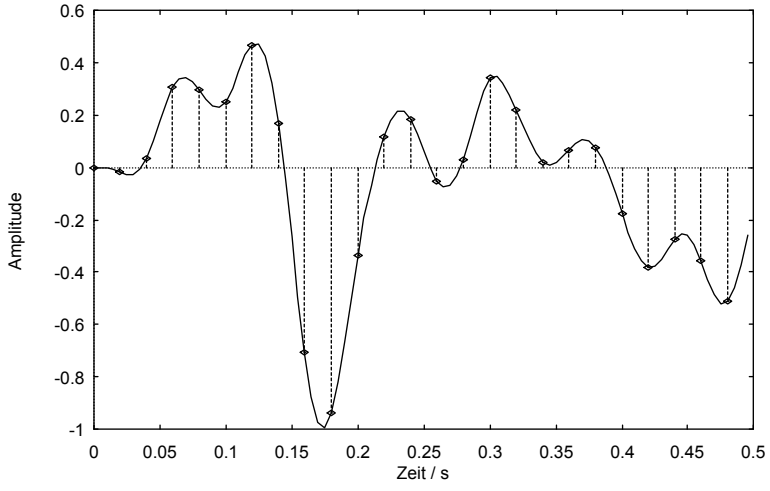
$$x(n) := x(n \cdot T) \quad ; n \in K \subset \mathbb{Z}$$

$$x(n) : Samples$$

$$T : Sampling\ \ period$$

$$f_s = 1/T : Sampling\ \ period$$

This process of transformation into a sequence of numbers is called sampling or discretization. Its most important quantity is the sampling frequency f_s, the reciprocal value of which is the sampling period T. The technical realization of sampling in order to discretize physical signals is called Analog-Digital Transform (A/D Transform) or digitization.



Analog-Digital Transform: Analog signal *x(t)* (solid line), sampling points (dashed lines) and samples (rhombs). The samples form the sequence of numbers *{x(n)} = { x(0), x(1), x(2) , x(3) , . . .}* related to the function *x(t)*.

[11] In case a real physical quantity is given as a function of time, the term "signal" is used in the following. The term "function" is used as an abstract mathematical representation of a signal.

Remark: Sampling is mathematically also feasible for infinite signals, however, on the computer it can be applied to finite signals only, otherwise the sequence of samples would become infinitely long.

Remark: Because of the finite precision of the number representation on the computer the values of the samples do not exactly correspond to the values of the function, but have a random roundoff error depending on the number representation. The deviation of the value of the sample represented on the computer from the true value of the function at the related point of time is called **quantization error**. If, for example, a sine function is represented by a series of samples, the quantization error just corresponds to the precision with which the values of the sine function are calculated.

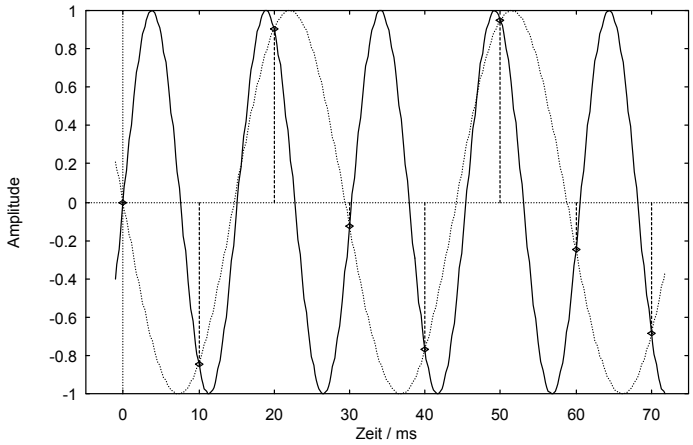
The course of the function between the sampling points is lost by sampling, however, the so-called **sampling theorem** states that no information is lost under a certain condition:

> **Sampling theorem** (without proof):
> A function *x(t)* can be uniquely reconstructed from the sequence of its samples $\{x(n)\}$, if it comprises only frequencies below half of the sampling frequency. In other words: The sequence of samples is a unique representation of the function only under this condition.

To illustrate the latter, the signal cannot alter arbitrarily fast, if high frequencies are lacking. Therefore, it becomes predictable in between two samples (can be interpolated).

Remark: In order not to infringe the sampling theorem, the analog signals must be low-pass filtered before sampling, i.e. all frequencies above half of the sampling frequency are blocked and only the low frequencies are passed.

Remark: If a signal contains frequencies higher than half of the sampling frequency, they will be reflected at lower frequencies in the sequence of samples ("mirror frequencies"). This phenomenon is called **aliasing** (see Figure below and the Exercises).



The aliasing phenomenon: Sampling of a sine wave of high frequency (solid line) at a sampling frequency which is too low. The samples (rhombs) also represent a sine wave of a lower frequency (dotted line) so that an ambiguity occurs (aliasing). The lower frequency is the 'mirror frequency' to the original frequency.

8.2 Diskrete Fourier Transform (DFT)

The Fourier Transform is defined for continuous functions in general. It describes an expansion of the function into sine and cosine functions comprising any possible frequency (continuous frequency spectrum). It states at which amplitude and phase each frequency is contained in the function. In order to obtain a version of the Fourier Transform which uses finite sequences of numbers only and thus can be performed on the computer, sampled periodic functions are considered in the following. Let us assume the sampling period to be T and the period of the signal $T_0 = N \cdot T$, with N being a positive integer. Then such a function is unambiguously represented by the N samples of a period

$$\{x(n)\} \quad ; \quad n = 0,\ldots,N-1$$

$x(N)$ is again identical with $x(0)$ due to periodicity etc. This finite sequence of numbers can be processed on a computer and nevertheless represents the entire infinitely long periodic function.

What does a frequency spectrum of a sampled periodic function look like? It can only be composed of sine functions, which also have the period T_0. Otherwise, a sine function would not be identical in two successive periods of the function, which contradicts the assumed periodicity. All sine functions with the period T_0 / k with k as an arbitrary integer number also have the period T_0. Thus, the spectrum comprises only frequencies $f_k = k / T_0$; hence, it is discrete with the sampling period $f_0 = 1 / T_0$ (this corresponds to the known Fourier series). Therefore, one prerequisite for representing the Fourier integral on the computer is already fulfilled: The spectrum can be represented as a sequence of numbers. But is this sequence really finite? This question is treated in the next paragraph.

Owing to the symmetry of the Fourier Transform (transform and reverse transform are only distinguished by a negative sign, which corresponds to a time reversal) the argument used above is valid universally as well: If the frequency spectrum is periodic with the period $f_s = 1 / T$ (periodic with the sampling frequency), the time function is discrete with the sampling period T (which is just the starting point). In case of frequency samples at $f_k = k / T_0$ and a period of the spectrum of $1 / T = N / T_0$, there are exactly N different frequency components.

Altogether, the frequency spectrum is periodic and discrete for discrete periodic functions. The function as well as its frequency spectrum can uniquely be represented as a finite sequence of numbers of the length N on the computer by the values of samples of one period each. The Discrete Fourier Transform relates the samples of the time function to those of the spectrum (without proof):

$$\textit{Discrete Fourier Transform of Length } N:$$

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot \exp\left(-2\pi i \frac{kn}{N}\right) \quad ; \quad k = 0 \ldots N-1$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot \exp\left(2\pi i \frac{kn}{N}\right) \quad ; \quad n = 0 \ldots N-1$$

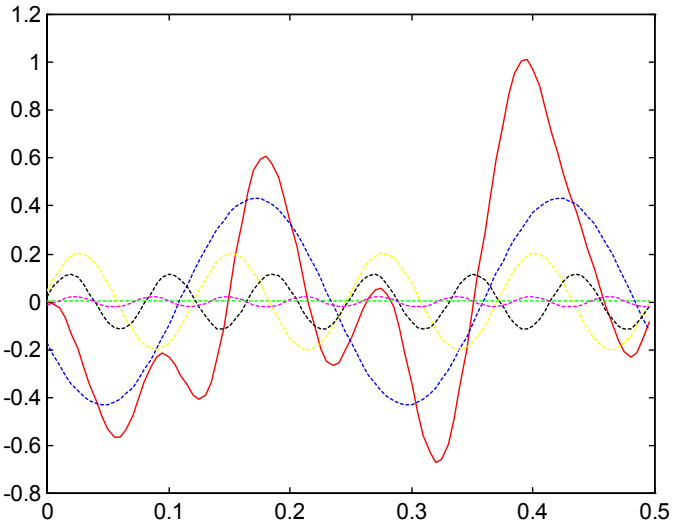
$X(k)$ and $x(n)$, respectively, represent the samples of the spectrum and the time function, respectively. The first equation denotes the DFT and the second equation denotes the inverse DFT. The relation between the actual index k and n, respectively and the frequency and time, respectively, is given by:

$$n = 0,1,2,\ldots,N-1 \quad \triangleq \quad t = 0,T,2T,\ldots,(N-1)T$$

$$k = 0,1,2,\ldots,N-1 \quad \triangleq \quad f = 0,\frac{1}{T_0},\frac{2}{T_0},\ldots,\frac{(N-1)}{T_0}$$

$$= 0,1\frac{f_s}{N},2\frac{f_s}{N},\ldots,(N-1)\frac{f_s}{N}$$

These formulas are applied to calculate the frequency and time axes of the sequences of numbers calculated by the DFT. The sample values $x(n)$ may be real- or complex-valued, physical signals being always real-valued. The $X(k)$ are complex-valued in both cases. The value $|X(k)|$ represents the amplitude and the argument ($\arg(X(k))$) represents the phase (shift), with which a sine/cosine function of the corresponding frequency is represented in the function $x(n)$.



DFT: Periodic physical signal (solid line, only one period is represented) and five of its different frequency components. Note the different frequencies, amplitudes, and relative shift (”phases“) of the components.

Remark: It is important to know that the DFT observes periodic functions **exclusively**. Considering an arbitrary finite function or signal, the DFT does not calculate the spectrum of that finite signal, but the spectrum of the periodically continued version of that period (for the consequences of this fact see the Exercises).

8.2.1 Real-valued time functions

If the time function $x(n)$ is real-valued, the following relation holds for the spectral samples:

$$X(N-k) = X^*(k) \quad ; \quad k = 0,\ldots,\frac{N}{2}$$

, * denoting the complex conjugation. Thus, $N/2$ complex values $X(k)$ are sufficient in order to represent N real values $x(n)$ and the spectrum is uniquely represented within the range of up to half the sampling frequency (cf. formula presented above for conversion between frequency index and frequency). Therefore, the spectrum is only represented in the range between 0 Hz and half the sampling frequency $f_s/2$ for real signals. The remainder is redundant.

Remark: The sampling theorem and the aliasing mentioned above result from the periodicity of the spectrum. If the spectrum is periodic with a sampling frequency f_s, only one of the periods is physically significant. For example, the values of the spectrum at the frequencies $f = 0.3\,f_s$ and $f = (0.3\,f_s + f_s)$ are always identical. Thus, the sampled signal cannot have comprised sine functions of those frequencies independent of each other, so they are indistinguishable. For real signals this applies also to the frequencies $f = 0.3\,f_s$ and $f = (0.3\,f_s + f_s/2)$ owing to the property mentioned above, so that the sampling theorem follows.

8.2.2 Intensity spectrum

Often we are not interested in the complex spectral values $X(k)$, but the question is: With which intensity is a sine of a certain frequency contained in the signal? The intensity is just the square of the spectral value $|X(k)|^2$. Within the range of typical physical signals, it may fluctuate by several orders of magnitude across the frequency so that it is often useful to take the logarithm for plotting the data. As **intensity spectrum in dB (decibel)** the following quantity is defined:

$$L(k)/dB = 10 \cdot \log_{10}\left(\left|X(k)\right|^2\right) \quad ; \quad k = 0,\ldots,N-1$$

Remark: When the amount $|X(k)|$ just increases tenfold, the value in the intensity spectrum changes by 20 dB.

8.3 Fast Fourier Transform (FFT)

The DFT is a relatively time-consuming method when calculating the spectral values $X(k)$ directly according to the formula of the DFT. For each of the N spectral values N multiplications and additions are to be calculated so that the total expenditure is proportional to N^2. However, Gauss already found out that partial sums of different frequencies (index k) are identical, if N is a power of 2, i.e. $N = 2^1, 2^2, 2^3, 2^4, 2^5, 2^6$.etc. The FFT makes use of these symmetry properties and calculates partial sums only once. Thus, the expenditure is reduced from N^2 to $N \cdot \log_2(N)$, which means a considerable difference for large N. Therefore, the FFT is nothing but a fast algorithm for calculating the DFT. It can be assumed that the DFT would not have gained such an importance within the field of computer-controlled signal analysis, if this fast algorithm for its calculation had not been found.

One possible derivation of the FFT algorithm is explained in the following. With the definition

$$W_N := \exp\left(-\frac{2\pi i}{N}\right)$$

the DFT reads

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn} \quad ; \quad k = 0,\ldots,N-1 \ .$$

First, this sum is divided into two partial sums containing the even and odd samples, respectively:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn} \\ &= \sum_{n=0}^{N/2-1} x(2n) \cdot W_N^{k2n} + \sum_{n=0}^{N/2-1} x(2n+1) \cdot W_N^{k(2n+1)} \\ &= \sum_{n=0}^{N/2-1} x(2n) \cdot \left(W_N^2\right)^{kn} + W_N^{-k} \cdot \sum_{n=0}^{N/2-1} x(2n+1) \cdot \left(W_N^2\right)^{kn} \end{aligned}$$

With the identity (see definition)

$$W_N^2 = W_{N/2}$$

we obtain:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) \cdot W_{N/2}^{kn} + W_N^k \cdot \left(\sum_{n=0}^{N/2-1} x(2n+1) \cdot W_{N/2}^{kn} \right)$$

The two remaining sums just represent the formula for the DFT of the even and odd samples, respectively, which are referred to as $X_g(k)$ and $X_u(k)$ in the following:

$$\begin{aligned} X_g(k) &= \sum_{n=0}^{N/2-1} x(2n) \cdot W_{N/2}^{kn} \quad ; \quad k = 0,\ldots,N/2-1 \\ X_u(k) &= \sum_{n=0}^{N/2-1} x(2n+1) \cdot W_{N/2}^{kn} \quad ; \quad k = 0,\ldots,N/2-1 \end{aligned} \ .$$

For these DFT's of length $N/2$ the index k ranges between 0 and $N/2 - 1$, however, for the spectral values $X(k)$ it ranges from 0 to N-1. Owing to the identity

$$W_{N/2}^{(k+N/2)n} = W_{N/2}^{kn}$$

the partial sums for $k = N/2$ up to N-1, however, again represent the DFT of the even and odd samples, respectively. Thus, the modulo function mod yields:

$$X(k) = X_g\left((k)_{\mathrm{mod}N/2}\right) + W_N^k \cdot X_u\left((k)_{\mathrm{mod}N/2}\right) \quad ; \quad k = 0,\ldots,N-1$$

The unknown spectral values $X(k)$ thus result from the two spectral values $X_g(k)$ and $X_u(k)$ by multiplication and addition. Each $X_g(k)$ and $X_u(k)$ is used twice due to the modulo function. This is the time-saving fact.

Altogether, these conversions divided the DFT of the length N into two DFT's of the length $N/2$ each and a subsequent arithmetic combination. The two DFT's work on the samples with even and odd indices, respectively, and the subsequent combination calculates the result of the entire DFT from the results of the partial DFT's. The subsequent combination requires a total of N operations (one operation per spectral value). The two partial DFT's can be split in the same way again in a further step, which results in two DFT's of the length $N/4$ and a subsequent combination with $N/2$ operations each. Altogether, there are four DFT's of the length $N/4$ and two subsequent combinations with a total of N operations in this next stage. This splitting can be repeated until the DFT works on two samples only. Then $N/2$ DFT's of the length 2 with an expenditure of 2 each are required, i.e., a total of N operations again.

Thus, N operations are required in each stage (for the DFT's of the length 2 in the lowest stage and for the subsequent combinations in the following stages, respectively) and there is a total of 10 $\log_2(N)$ stages (e.g. N=8 can be divided by 2 twice, until a DFT of the length 2 is reached. Thus, there are three stages altogether (one stage with DFT's of the length 2 and two stages with subsequent combinations)). The total expenditure of the FFT is therefore proportional to $N \cdot \log_2(N)$, as stated above.

Remark: The time required for calculating the DFT can be reduced to a certain degree also for the lengths N that do not correspond to a power of two. For this purpose N is split into a product of as many powers of two as possible, a separate FFT is performed for these partial sums of samples and the whole thing is assembled skillfully afterwards. This is called **mixed-radix FFT**, which is implemented in Matlab. The command **fft** performs a pure FFT or a mixed-radix FFT depending on the length of the given vector. The expenditure for a mixed-radix FFT ranges between that for a DFT and that for an FFT depending on how a number can be split iin powers of two. As it is not known beforehand in most cases how well a number can be split, powers of two should be used if possible. In order to obtain a desired sequence length, zeros should be added to the signal which does not alter the spectrum.

8.4 Filtering and convolution theorem

In signal theory the term **filtering** means the change of the frequency spectrum of a signal/function. For example, if certain frequency ranges in a physical signal to be analyzed are of particular interest, they may be increased by filtering, while the unimportant ones, e.g. the frequency ranges covered by noise can be reduced. Basically, four types of filters can be distinguished:

1. **Low-pass filter**: Passes only frequencies **below** a certain cutoff frequency.
2. **High-pass filter**: Passes only frequencies **above** a certain cutoff frequency.
3. **Band-pass filter**: Passes only frequencies **within a range** between a given lower and a higher cutoff frequency.
4. **„Notch filter"**: Passes only frequencies **beyond a range** between a given lower and higher cutoff frequency.

In the frequency domain the filtering process can be mathematically formulated in a simple way. Since the spectral values state the amplitude and phase of the frequency content, filtering can be described as a multiplication of spectral values by a frequency-dependent factor:

$$Y(k) = X(k) \cdot H(k) \quad ; \quad k = 0,\ldots,N-1$$

$X(k)$ and $Y(k)$ are the spectral values of the original and the filtered signals. The sequence of (complex) factors $H(k)$ is a unique definition of the filter and can be freely predetermined for realizing the required type of filtering. It is called **transfer function** of the filter.

Which operation within the time domain corresponds to the multiplication in the frequency range, i.e., how must x(n) and h(n) be linked in order that y(n) results (capitals and small letters mean pairs of time signals and spectral values that belong together)? For this purpose the **convolution theorem of the DFT** is applied (without proof):

$\{x(n)\}$ and $\{h(n)\}$ be two sequences of length N. Be $y(n)\}$ calculated by means of a cyclic convolution:

$$\begin{aligned} &zyklic\ \ convolution: \\ &y(n) = (x \otimes h)(n) \\ &\quad = \sum_{m=0}^{N-1} x(m) \cdot h\left((n-m)_{\bmod N}\right) \quad ; \quad n = 0,\ldots,N-1 \end{aligned}$$

The convolution theorem states that the spectral values $Y(k)$ of the sequence $\{y(n)\}$ are just the product of the spectral values of $\{x(n)\}$ and $\{h(n)\}$:

$$\begin{aligned} &convolution\ \ theorem: \\ &\quad y(n) = (x \otimes h)(n) \quad ; \quad n = 0,\ldots,N-1 \\ &\Leftrightarrow \quad Y(k) = X(k) \cdot H(k) \quad ; \quad k = 0,\ldots,N-1 \\ &mit: Y = DFT(y); X = DFT(x); H = DFT(h) \end{aligned}$$

Thus, a (cyclic) convolution in the time domain corresponds to a multiplication in the frequency domain. This is valid for the inverse, too: Multiplication in the time domain corresponds to conversion in the frequency domain.

Thus, the realization of a filter by multiplication in the frequency domain with the transfer function $H(k)$ corresponds to the cyclic convolution with the sequence $\{h(n)\}$ in the time domain, which is calculated from the $H(k)$ by inverse DFT. The sequence $\{h(n)\}$ is called **impulse response**.

Hence, filtering can be realized in two ways:

1. Applying the convolution theorem: First, calculate the DFT of the sequence, multiply it by the selected transfer function and then form the inverse DFT, in order to obtain the time function of the output signal:

$$y = IDFT\left(DFT(x) \cdot H\right)$$

2. Generate the output sequence directly by convolution in the time domain.

$$y(n) = (x \otimes h)(n) \quad ; \quad n = 0,\ldots,N-1$$

8.4.1 Convolution and cyclic convolution

In general, convolution for signals of any length is defined as:

$$\begin{aligned} y(n) &= (x \times h)(n) \\ &= \sum_{m=-\infty}^{\infty} x(m) \cdot h(n-m) \quad ; \quad n \in \mathbb{Z} \end{aligned}$$

For indices beyond the domain of x or h, a zero is inserted as value for this general definition. Therefore, the sum may always range from $-\infty$ to ∞.

Since convolution is a time-consuming algorithm, it is often realized by means of the convolution theorem, first switching to the frequency domain, then multiplying and finally transforming back into time domain. If the signals have the length of a power of two, the FFT can be applied. In that case the total expenditure for forward and inverse FFT as well as for the multiplication is lower than for a direct application of the convolution sum for signal lengths exceeding N=128. For realizing the convolution using the convolution theorem the following has to be considered: If the signals *x* and *h* are periodic and equal in length, the general form of the convolution is identical with the cyclic convolution. The cyclic convolution yields exactly one period of the periodic signal generated by convolution of two periodic signals. Since the DFT deals with periodic signals exclusively, the convolution theorem of the DFT applies to cyclic convolution and not to the general convolution sum, even if only finite signals are considered! The cyclic convolution realized by applying the convolution theorem and the general convolution sum thus yield different results in general (cf. Exercises).

Remark: The general form of the convolution sum is often applied as a mathematical definition of a filter. This also explains, why *h* is called the impulse response: If a pulse ($x(1)=1$, $x(n)=0$ otherwise), is used as input signal, the output function *y* is identical to *h*.

8.4.2 Deconvolution

The inversion of the filtering process is called **deconvolution**, which is applied, if the filtering effect of a filter needs to be compensated for (e.g. frequency-dependent alteration of the sensitivity of a sensor). Thus, an impulse response h^* is required which exactly neutralizes the effect of a known filter (impulse response *h*, transfer function *H*):

$$\begin{aligned} be: &\qquad y(n) = (x \times h)(n) \\ desired: &\qquad h^*(n) \\ with: &\qquad x(n) = \left(y \times h^*\right)(n) = \left(x \times h \times h^*\right)(n) \end{aligned}$$

With the help of the convolution theorem, h^* is easy to determine:

$$\begin{aligned} & DFT(y) = DFT(x) \cdot DFT(h) \\ \Rightarrow \quad & DFT(x) = DFT(y) \cdot \frac{1}{DFT(h)} \\ \Rightarrow \quad & x = IDFT(DFT(y) \cdot \frac{1}{DFT(h)}) \\ & \quad = y \times IDFT\left(\frac{1}{DFT(h)}\right) \\ \Rightarrow \quad & h^* = IDFT\left(\frac{1}{DFT(h)}\right) \end{aligned},$$

This equation shows that deconvolution is formally very easy to perform: For deconvolution, we simply divide by the transfer function *H* in the frequency range. Thus *H* is cancelled. In the time domain this corresponds to a convolution with the inverse DFT of the inverse transfer function $1/H$. Division by *H* is, however, numerically very problematic, if – as frequently occurring in practice – the value of *H* approaches zero for one or several frequencies. There are special deconvolution methods, which are not treated here, which solve this problem by approximation.

8.5 Exercises

1. Generate the different signals stated below with the sampling frequency f_s = 10 kHz and a duration of 10 ms (corresponding to100 samples). Calculate the intensity spectrums using the FFT (Matlab function **fft**) and plot four periods (frequency axis -2 f_s to +2 f_s). Plot also four periods of each of the time signals (time axis – 20 ms up to +20 ms). Where do the noticable differences in the spectra come from (qualitative explanation)?

   (a) Sine of the frequency 1 kHz.
   (b) Sine of the frequency 925 Hz
   (c) Sine of the frequency 1050 Hz

2. Aliasing phenomenon: Generate a sine of the frequency 1 kHz with the sampling frequencies f_s = 1.5, 1.7, 1.9, and 2.1 kHz and a duration of 50 periods (50 ms) each. Calculate the respective intensity spectrum and plot four periods (frequency axis -2 f_s up to +2 f_s). Identify the aliasing components. According to which formula can their frequency be calculated?

3. Convolution theorem: The following sampled functions be defined:

   $$x1 = \{0,0,0,1,0,0,0,0,0,0\}$$

   $$x2 = \{1,2,3,4,5,0,0,0,0,0\}$$

   $$x3 = \{1,2,3,4,5,6,7,8,9,10\}$$

   Calculate the convolution $x2 * x1$ and $x3 * x1$ by insertion into the convolution sum (non-cyclic convolution) and by application of the convolution theorem of the DFT (Hint: Matlab provides the functions fft and ifft for the Discrete Fourier Transform). What strikes us about the result and how can it be explained?

9 Partial Differential Equations

While ordinary differential equations (ODE) describe functions of one variable, e.g. $x=x(t)$, partial differential equations (PDE) deal with functions of several variables, e.g. the temperature as a function of space and time, $T = T(x,t)$. PDE are generally subdivided into three classes. Unfortunately, there are no universally applicable numerical methods for solving PDE like, e.g., the Runge-Kutta method for solving ODE. Thus, the three classes will be treated in the following by means of important examples and their specific solution methods.

9.1 Classification of partial differential equations

9.1.1 Parabolic equations

Parabolic PDE describe diffusion processes in a broadest sense, for example the **heat equation**

$$\frac{\partial}{\partial t}T(x,t) = \kappa \frac{\partial}{\partial x^2}T(x,t) \ .$$

(with: T, temperature, x and t, variables of position and time and κ; coefficient of thermal conduction) and the **time-dependent Schrödinger equation** from quantum mechanics

$$i\hbar \frac{\partial}{\partial t}\Psi(x,t) = H\Psi(x,t)$$

with the **Hamilton operator** H.

9.1.2 Hyperbolic equations

Hyperbolic PDE describe transport processes, for example the **advection equation**

$$\frac{\partial a(x,t)}{\partial t} = -c\frac{\partial a(x,t)}{\partial x}$$

and the **wave equation**

$$\frac{\partial^2 A(x,t)}{\partial t^2} = c^2 \frac{\partial^2 A(x,t)}{\partial x^2} \ .$$

A is the amplitude of the wave, x and t are variables of position and time, and c is the velocity of the wave.

9.1.3 Elliptic equations

The prototype of an elliptic PDE is the **Poisson equation** of electrostatics describing the potential of a charge distribution:

$$\frac{\partial^2 \Phi(x,y)}{\partial x^2} + \frac{\partial^2 \Phi(x,y)}{\partial y^2} = -\frac{1}{\varepsilon_0}\rho(x,y)$$

Φ is the potential, x and y are position variables, and ρ is the charge density distribution. If the charge density is zero, the resulting equation is called **Laplace equation**.

Remark: All equations are written one- or two-dimensional here for clarity. Formulation in several dimensions is possible (see textbooks).

Remark: The equations stated here are prototypes of the three classes. Real problems are often represented by mixtures of these prototype equations, e.g., the wave equation with damping.

9.2 Initial value problems

Equations that are position- as well as time-dependent are treated as initial value problems. The state at time t = 0 must be known, just as in the case of ordinary DE's. For the heat equation, for example, this means

$$T_0 = T(x, t = 0)$$

and for the wave equation (2 integration constants, because it is an equation of 2nd order)

$$A_0 = A(x, t = 0) \text{ and } A_0' = \frac{dA(x, t = 0)}{dx} .$$

$T(x,t)$ and $A(x,t)$, respectively for $t>0$ are then to be calculated. To determine the initial state, however, is not sufficient for solving the problem. In addition, the **boundary conditions** must be formulated. For this purpose, the solution is restricted to the range of x-values

$$x \in \left[-L/2, L/2\right]$$

with arbitrary and fixed L. There are several ways to restrict the solution to the boundary (stated for the heat equation here):

1. Fixed values of the solution on the boundary (Dirichlet's boundary condition):

$$T\left(x = -L/2, t\right) = T_a \quad , \quad T\left(x = L/2, t\right) = T_b$$

2. Periodic boundary conditions:

$$T\left(x = -L/2, t\right) = T\left(x = L/2, t\right)$$

or

$$\left.\frac{dT}{dx}\right|_{x=-L/2} = \left.\frac{dT}{dx}\right|_{x=L/2}$$

Thus, the solution is searched within an area limited by t=0 and x=± L/2. The question of why boundary conditions are needed is solved later on.

9.2.1 Discretization

For numerical calculation, time variable as well as position variable are discretized by sampling. The sampling points are

$$t_n := n \cdot \tau \quad ; \quad n = 0, 1, 2, 3, 4, \ldots$$

with the sampling period/step size τ. The area x=± L/2 is divided into N intervals, i.e., the sampling period/step size of the position variable is

$$h = \frac{L}{N}$$

and the sampling points (number of points: N+1) are

$$x_i := i \cdot h - L/2 \quad ; \quad i = 0, 1, 2, \ldots, N .$$

The solution is then searched for the variable pairs (x_i, t_n), i.e. the solution points

$$T_i^n := T\left(x_i, t_n\right)$$

are to be determined (here written for the heat equation). These are the definitions of sampling points and sampling positions used to describe the different methods in the following.

9.2.2 Heat equation: FTCS scheme

The FTCS scheme (Forward-Time-Centered-Space) makes use of the right-hand derivative formula for discretizing the time variable (Forward Time) and the centered derivative formula for the position variable. For the heat equation this means that the derivatives are being replaced by the following discrete approximations:

$$\begin{aligned}\frac{\partial T(x,t)}{\partial t} &\to \frac{T\left(x_i,t_n+\tau\right)-T\left(x_i,t_n\right)}{\tau}=\frac{T_i^{n+1}-T_i^n}{\tau}\\ \frac{\partial^2 T(x,t)}{\partial x^2} &\to \frac{T\left(x_i+h,t_n\right)+T\left(x_i-h,t_n\right)-2\cdot T\left(x_i,t_n\right)}{h^2}=\frac{T_{i+1}^n+T_{i-1}^n-2\cdot T_i^n}{h^2}\end{aligned}.$$

Note that the index notation introduced above was used for time and position index, respectively. Insertion into the heat equation yields

$$\boxed{\begin{aligned}&\frac{T_i^{n+1}-T_i^n}{\tau}=\kappa\cdot\frac{T_{i+1}^n+T_{i-1}^n-2\cdot T_i^n}{h^2}\\ \Rightarrow\quad &T_i^{n+1}=T_i^n+\frac{\kappa\tau}{h^2}\left(T_{i+1}^n+T_{i-1}^n-2\cdot T_i^n\right)\end{aligned}}.$$

From the initial values T_i^0 and the boundary values T_0^n and T_N^n the solution values T_i^n can be calculated iteratively with this formula.

Remark: It is necessary to predetermine the boundary values for the centered derivative!

The step sizes τ and *h* have to be chosen according to the length and time scales occurring in the individual physical problem. The relation between both values is of special importance, because it contradicts physical intuition to calculate the solution with a large time step τ from closely neighbouring points (*h* relatively small). The maximum time step for given *h* is (without proof):

$$\tau_{\max}=\frac{h^2}{2\kappa}$$

If the time step is chosen larger than this maximum, the solution becomes unstable. This equation can be empirically derived (see Exercises) or it can be derived by means of the von-Neumann stability analysis (cf. Section 9.2.5).

9.2.3 Time-dependent Schrödinger equation: Implicit Scheme (Crank-Nicholson)

The **time-dependent Schrödinger equation**

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t)=H\Psi(x,t)$$

with the wave function Ψ has the **Hamilton operator**

$$H=-\frac{\hbar^2}{2m}\frac{\partial}{\partial x^2}+V(x)\ .$$

for a particle of the mass *m* and the potential *V*. The square of the wave function describes the sojourn probability density

$$P(x,t)=\left|\Psi(x,t)\right|^2\ .$$

With this FTCS scheme the discretized Schrödinger equation reads:

$$i\hbar\frac{\Psi_j^{n+1}-\Psi_j^n}{\tau}=-\frac{\hbar^2}{2m}\frac{\Psi_{j+1}^n+\Psi_{j-1}^n-2\Psi_j^n}{h^2}+V_j\cdot\Psi_j^n$$
$$\Leftrightarrow\quad \Psi_j^{n+1}=\Psi_j^n-\frac{i\tau}{\hbar}\left(-\frac{\hbar^2}{2m}\frac{\Psi_{j+1}^n+\Psi_{j-1}^n-2\Psi_j^n}{h^2}+V_j\cdot\Psi_j^n\right)$$

using the above-mentioned notation for time and position indices. Writing the wave function as a time-dependent column vector

$$\underline{\Psi}^n:=\begin{pmatrix}\Psi_1^n\\ \vdots\\ \Psi_N^n\end{pmatrix}$$

we can write the discretized equation as a matrix equation solving the scheme for all sampling points of the position simultaneously:

$$\begin{aligned}&\underline{\Psi}^{n+1}=\underline{\Psi}^n-\frac{i\tau}{\hbar}\underline{\underline{H}}\cdot\underline{\Psi}^n\\ \Leftrightarrow\quad &\underline{\Psi}^{n+1}=\left(\underline{\underline{1}}-\frac{i\tau}{\hbar}\underline{\underline{H}}\right)\cdot\underline{\Psi}^n\end{aligned}\qquad (*).$$

$\underline{\underline{H}}$ is the matrix of the discretized Hamilton operator with the components

$$H_{i,j}=-\frac{\hbar^2}{2m}\frac{\delta_{i,j+1}+\delta_{i,j-1}-2\delta_{i,j}}{h^2}+V_i\delta_{i,j}\ .$$

The matrix equation (*) solves the Schrödinger equation according to the FTCS scheme. Since such matrix formulations are often found in the literature, this form is emphasized here.

Proceeding from the equation (*) the Crank-Nicholson method was developed as the most important type of the so-called **implicit schemes**. On the right side of the equation not only the old value of $\underline{\Psi}$, but the mean of the old and new values is inserted:

$$\underline{\Psi}^{n+1}=\underline{\Psi}^n-\frac{i\tau}{2\hbar}\underline{\underline{H}}\cdot\left(\underline{\Psi}^{n+1}+\underline{\Psi}^n\right)$$
$$\Leftrightarrow\quad \left(\underline{\underline{1}}+\frac{i\tau}{2\hbar}\underline{\underline{H}}\right)\underline{\Psi}^{n+1}=\left(\underline{\underline{1}}-\frac{i\tau}{2\hbar}\underline{\underline{H}}\right)\underline{\Psi}^n$$

Solving for $\underline{\Psi}^{n+1}$ then yields the Crank-Nicholson scheme:

$$\boxed{\underline{\Psi}^{n+1}=\left(\underline{\underline{1}}+\frac{i\tau}{2\hbar}\underline{\underline{H}}\right)^{-1}\left(\underline{\underline{1}}-\frac{i\tau}{2\hbar}\underline{\underline{H}}\right)\underline{\Psi}^n}.$$

As compared to the FTCS scheme, this scheme has the advantage that it is always stable and that the applied matrix operators are unitary (without proof).

9.2.3.1 Wave package of a free particle

As an example of solving the Schrödinger equation we observe a **Gaussian wave package** which is used in quantum mechanics to describe a free particle (potential V=0). As initial value for the wave function

$$\Psi(x,t=0)=\frac{1}{\sqrt{\sigma_0\sqrt{\pi}}}\exp\left(ik_0x\right)\exp\left(-\left(x-x_0\right)^2\Big/2\sigma_0^2\right)\ .$$

is used, x_0 being the current average position of the particle. σ_0 is the current width of the package and $p_0=\hbar\cdot k_0$ is the current impulse of the particle. The time development of the wave function can be calculated from the initial state by means of the Crank-Nicholson scheme. We know from quantum mechanics that there is also an analytical solution for the wave package. It reads:

$$\Psi(x,t)=\frac{1}{\sqrt{\sigma_0\sqrt{\pi}}}\frac{\sigma_0}{\alpha}\exp\left(ik_0\left(x-p_0t/2m\right)\right)\exp\left(-\left(x-x_0-p_0t/2m\right)^2\Big/2\alpha^2\right)\ .$$

$$with: \qquad \alpha^2=\sigma_0^2+i\hbar t/m$$

Thus, the form of the wave package is preserved in time. The mean value (expected value) moves at the velocity p_0/m:

$$\langle x\rangle=\int x\cdot P(x,t)dx=x_0+\frac{p_0}{m}t$$

and the standard deviation expands according to

$$\sigma(t)=\sigma_0\sqrt{\left(\frac{|\alpha|}{\sigma_0}\right)^4}=\sigma_0\sqrt{1+\frac{\hbar^2t^2}{m^2\sigma_0^4}}\ .$$

This analytical solution can be used in order to test the Crank-Nicholson scheme (see Exercises).

9.2.4 Advection equation: Lax-Wendroff scheme

We proceed from the wave equation

$$\frac{\partial^2A(x,t)}{\partial t^2}=c^2\cdot\frac{\partial^2A(x,t)}{\partial x^2}\ .$$

As in the case of ODE we rewrite this equation of the 2nd order into two equations of the 1st order. For this, we define the intermediate variables

$$p=\frac{\partial A}{\partial t}\quad;\quad q=c\cdot\frac{\partial A}{\partial x}\ .$$

With that we can write the wave equation as a pair of equations

$$\frac{\partial p}{\partial t}=c\cdot\frac{\partial q}{\partial x}\quad;\quad\frac{\partial q}{\partial t}=c\cdot\frac{\partial p}{\partial x}$$

The first equation has resulted from the insertion of the variable into the wave equation. The second equation is yielded by a comparison of the mixed derivatives from p to x und from q to t. These two equations can be written in a vector form again:

$$\frac{\partial \underline{a}(x,t)}{\partial t} = c \cdot \underline{\underline{B}} \cdot \frac{\partial \underline{a}(x,t)}{\partial x}$$

$$with: \quad \underline{a}(x,t) = \begin{pmatrix} p(x,t) \\ q(x,t) \end{pmatrix} \quad ; \quad \underline{\underline{B}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The wave equation is not the simplest equation of this kind. Setting

$$\underline{\underline{B}} = \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix}$$

, we obtain the so-called **advection equation**[12]

$$\boxed{\frac{\partial a(x,t)}{\partial t} = -c \frac{\partial a(x,t)}{\partial x}}$$

, with *a* being the unknown quantity which we can imagine as the amplitude of a wave. In the following we observe this simplest form of a hyperbolic equation. By insertion it is easy to demonstrate that each function of the form:

$$a(x,t) = f_0(x - ct)$$

with an arbitrary function f_0 is a solution of the advection equation, *c* representing a diffusion velocity. For example, a **cosine-modulated Gauss pulse**

$$a(x,t) = \cos\left(\frac{2\pi}{\lambda}(x - c \cdot t)\right) \cdot \exp\left(-\frac{(x - c \cdot t)^2}{2\sigma^2}\right)$$

is a solution. The function represents a short wave package of the wave length λ and of the spatial extent σ, which moves in the direction *x* at the velocity *c*.

Since the analytical solution is known in the case of the advection equation, it can be used to test numerical methods for solving hyperbolic equations. In the following the Lax-Wendroff scheme is presented for this purpose.

Remark: The advection equation represents the simplest form of a **conservation equation**.

$$\frac{\partial \underline{a}}{\partial t} = -grad\left(\underline{F}(\underline{a})\right) \quad \text{oder 1-D:} \quad \frac{\partial a}{\partial t} = -\frac{\partial F(a)}{\partial x}$$

$\underline{a}$ can be the density of the mass, impulse or energy and $\underline{F}(\underline{a})$ can be related to the flow of the mass, impulse or energy, for example. The continuity equation for the mass in a current be stated as an example (one-dimensional):

$$\frac{\partial \rho(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left(\rho(x,t) \cdot v(x,t)\right) .$$

ρ is the mass density and *v* the velocity of the current.

[12] Advection: Horizontal movement of masses of air or horizontal flow of masses of water in the sea (according to some dictionary).

9.2.4.1 Lax-Wendroff scheme

Theoretically, the advection equation can be discretized according to the FTCS scheme, however, this does not yield a stable solution (see Section 9.2.5 and Exercises). The Lax-Wendroff scheme is of a higher order and represents a stable method for solving the advection equation, although it is not without numerical errors (see Exercises). To derive the scheme, we proceed from the Taylor expansion of the 2nd order for the time development:

$$a(x,t+\tau) \approx a(x,t) + \tau \cdot \frac{\partial a(x,t)}{\partial t} + \frac{\tau^2}{2} \cdot \frac{\partial^2 a(x,t)}{\partial t^2}$$

Then the second derivative with respect to time is determined from the advection equation:

$$\frac{\partial a}{\partial t} = -\frac{\partial F(a)}{\partial x} \qquad with: \quad F(a) = c \cdot a$$

$$\Rightarrow \quad \frac{\partial^2 a}{\partial t^2} = -\frac{\partial}{\partial t}\frac{\partial F}{\partial x} = -\frac{\partial}{\partial x}\frac{\partial F}{\partial t}$$

With

$$\frac{\partial F(a)}{\partial t} = \frac{dF}{da} \cdot \frac{\partial a}{\partial t} = F'(a) \cdot \frac{\partial a}{\partial t} = -F'(a) \cdot \frac{\partial F(a)}{\partial x}$$

we obtain the following form of the second derivative by insertion

$$\frac{\partial^2 a}{\partial t^2} = \frac{\partial}{\partial x} F'(a) \frac{\partial F(a)}{\partial x}$$

and hence for the Taylor development:

$$a(x,t+\tau) \approx a(x,t) - \tau \cdot \left(\frac{\partial}{\partial x} F\left(a(x,t)\right) \right) + \frac{\tau^2}{2} \cdot \left(\frac{\partial}{\partial x} F'\left(a(x,t)\right) \frac{\partial F\left(a(x,t)\right)}{\partial x} \right)$$

This formula again allows the state at time $t+\tau$ to be calculated from states at time t. It is discretized now (nomenclature of indices as for the heat equation). For the first derivative of time the centered formula is applied:

$$\frac{\partial}{\partial x} F(a)\Big|_{x_i,t_n} \quad \rightarrow \quad \frac{F_{i+1}^n - F_{i-1}^n}{2h}$$

The calculation of the multiple derivative in the last term is rather complex. The outer derivative is a right-hand formula, just like the inner derivative. The derivative F' is calculated from the mean of the amplitude values found in both terms of the outer right-hand derivative:

$$\frac{\partial}{\partial x}\left(F'(a)\frac{\partial F(a)}{\partial x}\right)\Big|_{x_i,t_n} \rightarrow \frac{\left(F'(a)\frac{\partial F(a)}{\partial x}\right)\Big|_{x_{i+1},t_n} - \left(F'(a)\frac{\partial F(a)}{\partial x}\right)\Big|_{x_i,t_n}}{h}$$

$$\left(F'(a)\frac{\partial F(a)}{\partial x}\right)\Big|_{x_{i+1},t_n} \rightarrow F'\left[\left(a_{i+1}^n + a_i^n\right)/2\right]\cdot\frac{F_{i+1}^n - F_i^n}{h}$$

$$\left(F'(a)\frac{\partial F(a)}{\partial x}\right)\Big|_{x_i,t_n} \rightarrow F'\left[\left(a_i^n + a_{i-1}^n\right)/2\right]\cdot\frac{F_i^n - F_{i-1}^n}{h}$$

Altogether, this yields:

$$a_i^{n+1} = a_i^n - \tau\cdot\frac{F_{i+1}^n - F_{i-1}^n}{2h} + \frac{\tau^2}{2}\frac{1}{h}\cdot\left(F'^n_{i+1/2}\cdot\frac{F_{i+1}^n - F_i^n}{h} - F'^n_{i-1/2}\cdot\frac{F_i^n - F_{i-1}^n}{h}\right)$$

with

$$F_i^n \equiv F\left(a_i^n\right) \quad and \quad F'^n_{i\pm1/2} \equiv F'\left[\left(a_{i\pm1}^n + a_i^n\right)/2\right].$$

For the advection equation with

$$F(a) = c\cdot a$$

$$\Rightarrow \quad F_i^n = c\cdot a_i^n \quad and \quad F'^n_{i\pm1/2} = c$$

the considerably simplified form reads:

$$\boxed{a_i^{n+1} = a_i^n - \frac{c\tau}{2h}\left(a_{i+1}^n - a_{i-1}^n\right) + \frac{c^2\tau^2}{2h^2}\left(a_{i+1}^n + a_{i-1}^n - 2a_i^n\right)}$$

Just as in the case of the FTCS scheme, a sample at time $t+\tau$ can be calculated from three neighbouring points at a time t in this way. Contrary to the FTCS scheme, however, it is stable if the time step is maximally τ_{max}, with

$$\tau_{\max} = \frac{h}{c}$$

A proof is feasible by means of the von-Neumann stability analysis (cf. Section 9.2.5) (see also Exercises).

9.2.5 von-Neumann stability analysis

The von-Neumann stability analysis allows to determine stability rules for the different schemes for solving PDE. We start from a little perturbation ”injected“ into the iteration equations and then track, whether this perturbation diverges (instable scheme) or converges (“damping” of the perturbation, stable scheme). If the perturbation is chosen skillfully, this can be decided analytically, i.e. without actual numerical iteration of the equation. For the von-Neumann analysis the following special perturbation is used, which represents a fixed wave with time-dependent amplitude:

$$a_j^n = \xi^n \cdot e^{i \cdot k \cdot h \cdot j}$$

$$with:$$

$$\xi : Amplification\ factor$$

$$k = \frac{2\pi}{\lambda} : Wave\ number\ (arbitrary)$$

$$h : spatial\ step$$

$$n : time\ index\ (left\ hand\ side) / exponent\ (right\ hand\ side)$$

$$j : spatial\ index$$

$$i : complex\ number\ \sqrt{-1}$$

The dependence on position (complex *e*-function) and the dependence on time (power function) are separated here, i.e. they are represented as a product (**Remarks**: The index *n* means the index of time on the left hand side of the equation (discretization) and the power of the amplitude on the right hand side. In order to avoid a confusion with the complex number *i*, *j* was used here as the index of time). The amplitude ξ is also called **amplification factor**. If the amplification factor for an iterative equation is larger than 1, it is instable. It is stable only for values smaller than 1. Calculation of ξ by means of the FCTS scheme is shown in the following exemplary for the advection equation.

9.2.5.1 Stability of the FTCS scheme for the advection equation

The FTCS scheme for the advection equation (insertion of the right-hand and centered formulas, respectively, for time and position) reads:

$$a_j^{n+1} = a_j^n - \frac{c\tau}{2h}\left(a_{j+1}^n - a_{j-1}^n\right).$$

Insertion of the perturbation yields:

$$\begin{aligned} \xi^{n+1} \cdot e^{ikhj} &= \xi^n \cdot e^{ikhj} - \frac{c\tau}{2h}\left(\xi^n \cdot e^{ikh(j+1)} - \xi^n \cdot e^{ikh(j-1)}\right) \\ &= \xi^n \cdot e^{ikhj}\left(1 - \frac{c\tau}{2h}\left(e^{ikh} - e^{-ikh}\right)\right) \end{aligned}.$$

Dividing both sides by $\xi^n \cdot e^{ikhj}$ we directly obtain an expression for the amplification factor:

$$\begin{aligned} \xi &= 1 - \frac{c\tau}{2h}\left(e^{ikh} - e^{-ikh}\right) \\ &= 1 - i\frac{c\tau}{h}\sin(kh) \end{aligned}$$

The absolute value of the amplification factor then is:

$$|\xi| = \sqrt{1 + \left(\frac{c\tau}{h}\right)^2 \sin^2(kh)}$$

This expression is always larger than 1 independent of the time step chosen. By this we have analytically shown that the scheme is instable without having to iterate the equation numerically. This is made possible by the skillful choice of the perturbation. Please refer to exercises for the application of the von-Neumann stability analysis to the Lax-Wendroff scheme.

9.3 Boundary value problems: Poisson and Laplace equations

Equations independent of time are described as **boundary value problems**. The Poisson and Laplace equations from electrostatics are two important examples. As initial condition, the solution on the boundary of an area needs to be predetermined. For two dimensions a rectangle with the side lengths L_x and L_y can be chosen as the boundary[13]. The x-variable then runs from x=0 to x= L_x and the y-variable from 0 to L_y. The boundary conditions read

$$\Phi(x=0,y)=\Phi_1 \quad \Phi(x=L_x,y)=\Phi_2$$
$$\Phi(x,y=0)=\Phi_3 \quad \Phi\left(x,y=L_y\right)=\Phi_4$$

(Dirichlet's boundary condition) and

$$\Phi(x=0,y)=\Phi(x=L_x,y)$$
$$\Phi(x,y=0)=\Phi\left(x,y=L_y\right)$$

(periodic boundary conditions). The solution is then to be determined in the interior of the rectangle. For the numerical solution, the x- and y-variables are again discretized by sampling. For this purpose, N and M intervals are used for the x- and y-dimension, respectively:

$$h_x=\frac{L_x}{N};\quad x_i:=i\cdot h_x \quad ;\quad i=0,1,2,\ldots,N$$
$$h_y=\frac{L_y}{M};\quad y_j:=j\cdot h_y \quad ;\quad j=0,1,2,\ldots,M$$

Then the solution is calculated for the variable pairs (x_i , y_j), i.e. the solution points

$$\Phi_{i,j}:=\Phi\left(x_i,x_j\right)$$

are calculated. Basic methods for solving the Laplace and Poisson equations are investigated in the following.

9.3.1 Laplace equation: Relaxation methods (Jacobi method)

The Laplace equation

$$\frac{\partial^2\Phi(x,y)}{\partial x^2}+\frac{\partial^2\Phi(x,y)}{\partial y^2}=0$$

is identical to the heat equation in two dimensions in its spatial coordinate except for the factor (coefficient of thermal conduction)

$$\frac{\partial T(x,y,t)}{\partial t}=\kappa\cdot\left(\frac{\partial^2 T(x,y,t)}{\partial x^2}+\frac{\partial^2 T(x,y,t)}{\partial y^2}\right)$$

For large times (t infinite) the solution of the heat equation approaches a stationary state (stationary temperature distribution):

$$\lim_{t\to\infty}T(x,y,t)=T_s(x,y)$$
$$\Leftrightarrow \quad \frac{\partial^2 T_s(x,y)}{\partial x^2}+\frac{\partial^2 T_s(x,y)}{\partial y^2}=0$$

[13] There are other special solutions for cyclindrical and spherical geometries.

Thus, the stationary state T_s is at the same time a solution of the Laplace equation. The general idea of the **relaxation methods** is to introduce a virtual dependence on time into a time-independent equation and then to iterate it until a stationary state is reached. The latter is the solution for the stationary, time-independent equation. In the simplest case this means for the Laplace equation that we write the equation formally like the heat equation, as shown above, making the potential time-dependent:

$$\frac{\partial \Phi(x,y,t)}{\partial t} = \mu \cdot \left(\frac{\partial^2 \Phi(x,y,t)}{\partial x^2} + \frac{\partial^2 \Phi(x,y,t)}{\partial y^2} \right)$$

μ being an arbitrary convergence factor (corresponding to the coefficient of heat conduction). According to the FTCS scheme the equation can be discretized as follows:

$$\Phi_{i,j}^{n+1} = \Phi_{i,j}^{n} + \frac{\mu\tau}{h_x{}^2}\left(\Phi_{i+1,j}^{n} + \Phi_{i-1,j}^{n} - 2\cdot\Phi_{i,j}^{n}\right) + \frac{\mu\tau}{h_y{}^2}\left(\Phi_{i,j+1}^{n} + \Phi_{i,j-1}^{n} - 2\cdot\Phi_{i,j}^{n}\right) .$$

The nomenclature for time and position indices is as stated above. Note, however, that the scheme has been applied to both spatial dimensions here. Hence, the new state at the point (x_i,y_j) is calculated from the values in this point and all directly neighbouring points.

Similar to the FTCS scheme of the one-dimensional heat equation, the stability condition for this scheme reads (without proof):

$$\frac{\mu\tau}{h_x{}^2} + \frac{\mu\tau}{h_y{}^2} \leq 1/2 \ .$$

Observing the equation in case of identical step sizes h_x and h_y, i.e.

$$h_x = h_y = h \ ,$$

the stability condition reads

$$\frac{\mu\tau}{h^2} \leq 1/4 \ .$$

Since only the stationary state is of interest, the maximal time step is chosen. Insertion of

$$\frac{\mu\tau}{h^2} = 1/4$$

into the relaxation scheme yields

$$\boxed{\Phi_{i,j}^{n+1} = \frac{1}{4}\left(\Phi_{i+1,j}^{n} + \Phi_{i-1,j}^{n} + \Phi_{i,j+1}^{n} + \Phi_{i,j-1}^{n}\right)}.$$

Please note that the central point (x_i,y_j) does not enter the calculation of the potential at all but only the neighbouring points because of the maximum time step chosen. This scheme is a special relaxation scheme for solving the Laplace equation and is called the **Jacobi method**. Proceeding from the Jacobi method the convergence velocity can be increased by varying the iteration rules. The schemes resulting from that are called **superrelaxation schemes**. Well-known methods are **Gauss-Seidel** and **simultaneous superrelaxation**. Since they are derived from the Jacobi method rather empirically, they will not be treated in detail here (refer to the textbooks).

9.3.1.1 Calculating initial values

The convergence of relaxation methods depends on the initial values chosen to initialize the unknown values in the interior of the rectangle. If they are chosen skillfully, the system can converge with distinctly less steps. It is known from the literature that the solution of a Laplace equation can be determined analytically within a rectangle using the method of **separation of variables**. Typically, this results in infinite series, which are exactly defined analytically, but converge slowly in general. Therefore, a numerical solution is still desirable, although an analytical solution exists. If the infinite series is known, it is the obvious thing to use the first elements of the series as initial value for the relaxation methods. As an example let us look at the solution of the Laplace equation on a rectangle with the boundary conditions

$$\Phi(x=0,y)=\Phi\left(x=L_x,y\right)=\Phi(x,y=0)=0$$
$$\Phi\left(x,y=L_y\right)=\Phi_0$$

The ansatz of the separation of variables

$$\Phi(x,y)=X(x)\cdot Y(y)$$

with these boundary conditions yields the solution (cf. textbooks)

$$\Phi(x,y)=\Phi_0\cdot\sum_{N=1,3,5,\ldots}^{\infty}\frac{4}{\pi n}\sin\left(\frac{n\pi x}{L_x}\right)\sinh\left(\frac{n\pi y/L_x}{n\pi L_y/L_x}\right).$$

If the first element of the series is taken as initial value of the Jacobi method, the convergence velocity increases considerably as compared to a suboptimal choice of initial values (cf. Exercises).

9.3.2 Poisson equation

The Poisson equation

$$\frac{\partial^2\Phi(x,y)}{\partial x^2}+\frac{\partial^2\Phi(x,y)}{\partial y^2}=\frac{1}{\varepsilon_0}\rho(x,y)$$

generally has no analytical solutions for an arbitrary distribution of charges ρ by means of separation of variables. Therefore, two numerical methods are presented in the following. The charge density is discretized like the potential:

$$\rho_{i,j}:=\rho\left(x_i,x_j\right)$$

9.3.2.1 Jacobi method

The Jacobi method can be directly extended to solve the Poisson equation. The formalism used in Section 9.3.1 leads to the relaxation equation

$$\boxed{\Phi_{i,j}^{n+1}=\frac{1}{4}\left(\Phi_{i+1,j}^{n}+\Phi_{i-1,j}^{n}+\Phi_{i,j+1}^{n}+\Phi_{i,j-1}^{n}+\frac{1}{\varepsilon_0}h^2\rho_{i,j}\right)}.$$

Since there are generally no analytical solutions, the initial values cannot be chosen optimally. Therefore, the convergence is to be expected to be slow.

9.3.2.2 Method of multiple Fourier transform

Contrary to Dirichlet's boundary conditions, the method of multiple Fourier transform enables the Poisson equation to be solved on a square with periodic boundary conditions[14]. The number of samples N and M as well as the step sizes h_x and h_y in x- and y-direction must be identical:

$$N = M$$
$$h_x = h_y = h$$

Proceeding from that the partial derivatives are discretized with respect to the spatial coordinates by the centered formula:

$$\left.\frac{\partial^2 \Phi}{\partial x^2}\right|_{i,j} \rightarrow \frac{\Phi_{i+1,j} + \Phi_{i-1,j} - 2 \cdot \Phi_{i,j}}{h^2}$$
$$\left.\frac{\partial^2 \Phi}{\partial y^2}\right|_{i,j} \rightarrow \frac{\Phi_{i,j+1} + \Phi_{i,j-1} - 2 \cdot \Phi_{i,j}}{h^2}$$

Insertion into the Poisson equation thus yields:

$$\frac{1}{h^2}\left(\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1} - 4 \cdot \Phi_{i,j}\right) = \frac{1}{\varepsilon_0}\rho_{i,j} \quad (*)$$

Now we define the two-dimensional Discrete Fourier transform (2-D DFT) of the potential and the charge density as

$$F_{m,n} = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \Phi_{j,k} \cdot W_N^{jm} \cdot W_N^{kn} \quad ; \quad m,n = 0,\ldots,N-1$$
$$R_{m,n} = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \rho_{j,k} \cdot W_N^{jm} \cdot W_N^{kn} \quad ; \quad m,n = 0,\ldots,N-1$$
$$mit: W_N = \exp\left(-\frac{2\pi i}{N}\right)$$

(as compared to the one-dimensional DFT). The equation (*) is now Fourier-transformed on both sides. This yields:

$$\frac{1}{h^2}\left(W_N^{-m} + W_N^{m} + W_N^{-n} + W_N^{n} - 4\right) \cdot F_{m,n} = -\frac{1}{\varepsilon_0} R_{m,n}$$
$$\Leftrightarrow \quad \left(2\cos\left(\frac{2\pi m}{N}\right) + 2\cos\left(\frac{2\pi n}{N}\right) - 4\right) \cdot F_{m,n} = -\frac{h^2}{\varepsilon_0} R_{m,n} \quad (**)$$

using the theorem that:

$$displacement: \quad \Phi'_{j,k} := \Phi_{j+l,k}$$
$$\Rightarrow \quad F'_{m,n} = W_N^{-lm} \cdot F_{m,n}$$

(for proof refer to Remark below). Hence, displacement means a multiplication of the spectral components by a complex factor.

[14] Since the discrete Fourier transform considers periodic functions only, it is not surprising that this method uses periodic boundary conditions.

Altogether, the spectral components of the potential can be determined from the (known) spectral components of the given distribution of the charge by solving the equation (**) for the $F_{m,n}$:

$$F_{m,n} = P_{m,n} \cdot R_{m,n}$$

$$with: \quad P_{m,n} = -\frac{h^2}{2\varepsilon_0}\left(\frac{1}{\cos\left(\frac{2\pi m}{N}\right)+\cos\left(\frac{2\pi n}{N}\right)-2}\right)$$

$$and: \quad \Phi = IDFT(F)$$

Thus, the unknown potential is yielded by inverse Fourier transform of the spectral components $F_{m,n}$., which, in turn, result from filtering the spectral components of the charge distribution with the filter P.

Remark: Determination of the spectral components $F_{m,n}$ from the $R_{m,n}$ corresponds to a filtering process with the transfer function $P_{m,n}$.

Remark: Proof of the displacement theorem:

$$\Phi'_{j,k} := \Phi_{j+l,k}$$

$$\Rightarrow \quad F'_{m,n} = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \Phi'_{j,k} \cdot W_N^{jm} \cdot W_N^{kn}$$

$$\Leftrightarrow \quad F'_{m,n} = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \Phi_{j+l,k} \cdot W_N^{jm} \cdot W_N^{kn}$$

$$\Leftrightarrow \quad F'_{m,n} = \sum_{j'=l}^{N-1+l}\sum_{k=0}^{N-1} \Phi_{j',k} \cdot W_N^{(j'-l)m} \cdot W_N^{kn}$$

$$j' = j + l$$

$$\Leftrightarrow \quad F'_{m,n} = W_N^{-lm} \cdot \left(\sum_{j'=l}^{N-1+l}\sum_{k=0}^{N-1} \Phi_{j',k} \cdot W_N^{j'm} \cdot W_N^{kn} \right)$$

$$\Leftrightarrow \quad F'_{m,n} = W_N^{-lm} \cdot F_{m,n}$$

$$\Phi \ periodic \ with \ N$$

9.4 Exercises

1. The script dftcs solves the heat equation according to the FTCS scheme for a delta-shaped temperature distribution across the x-axis as initial value. This distribution "decays" in the course of time by diffusion. Examine the stability of the solution for different sets of parameters. Use N=41 and several values of the time step τ within the ranges 10^{-3} and 10^{-5}. Try several different values of N for $\tau = 2.0\text{x}10^{-4}$.

2. Show by insertion that the Gaussian function

$$T(x,t) = \frac{1}{\sigma(t)\sqrt{2\pi}} \exp\left[\frac{-(x-x_0)^2}{2\sigma^2(t)}\right]$$

with

$$\sigma(t) = \sqrt{2\kappa t}$$

is the solution of the heat equation. Derive a stability rule from the time development of the variance σ (**Hint:** Calculate the time required for the variance to rise from 0 to h, h being the grid width of the spatial coordinate). Does this correspond to the results of Exercise 1?

3. Calculate the time development of the wave function of a free particle (Gaussian wave package) according to the Crank-Nicholson scheme with periodic boundary conditions. Use the following parameters: L=100; N=30 **and** 80; m=1; τ=1; $\hbar = 1$; $\sigma_0 = 1$ and the mean velocity $p_0/m = \hbar k_0/m = 0.5$.

   Proceed as follows:
   - Calculate the initial state $\underline{\Psi}^0$. Ensure that the boundary values are periodic.
   - Calculate the matrix of the Hamilton operator. Ensure that it meets the periodic boundary conditions (which demands are to be made on the matrix in this respect?)
   - Calculate the total matrix in order to calculate $\underline{\Psi}^{n+1}$ from $\underline{\Psi}^n$.
   - Iterate the scheme until the wave package has circularly passed the system once (calculate the number of the steps required for this from L, τ, and the mean velocity).
   – Plot $\underline{\Psi}^0$ (real and imaginary parts) as well as $\left|\underline{\Psi}^n\right|^2$ for all time points (3D plot).

   To what extent does the numerical solution correspond to the analytical solution? Why does the numerical solution for N=30 not come up to our expectation (cue: sampling theorem)?

4. The script aftcs solves the advection equation according to the FTCS scheme with periodic boundary conditions and a cosine-modulated Gauss pulse as initial value.

   a.) Apply the script with a time step τ=0.002 and N=50 grid points. How do we know that the obtained numerical solution is not correct?
   b.) Solve the equation according to the Lax-Wendroff scheme. Modify the script accordingly and judge the numerical solution. For this purpose, use time steps just below the maximum stable time step and the maximum time step itself.

5. Apply the von-Neumann stability analysis to the Lax-Wendroff scheme. For this, calculate the formula for the amount of the amplification factor $|\xi|$ (calculation in writing). Derive the stability condition

$$\tau_{\max} = \frac{h}{c}$$

by discussing the maximum of $|\xi|$ for the following cases:

$$\frac{\tau c}{h} : \begin{cases} < 1 \\ = 1 \\ > 1 \end{cases}$$

Hint: Observe the maximum values of the different terms of $|\xi|$ in dependence on the argument $x = k \cdot h \cdot j$ of the sine and cosine functions. For illustration, plot $|\xi|$ as a function of x for x at the interval 0 to 2π for these three cases.

6. The script jacobi solves the Laplace equation on a square. The potential on the margin is zero on three sides and 1 on one side ($y=L$). As initial value for the inner points the first member of the infinite series is used which represents the analytical solution (solution via separation of variables). The time expenditure is critical. It is basically determined by the number of grid points and the initial values chosen.

   a.) Use the script for different spatial solutions (number of grid points N=10 to 30). How much does the time expenditure increase with the spatial solution? Plot the number of iterations required for convergence over N and adjust a power law to the data. Determine the exponent.
   b.) Repeat a.), however, use badly chosen initial values (e.g. potential Φ=0 for all inner points.

7. Solve the Poisson equation

$$\frac{\partial^2 \Phi(x,y)}{\partial x^2} + \frac{\partial^2 \Phi(x,y)}{\partial y^2} = -\frac{1}{\varepsilon_0} \rho(x,y)$$

on a square with the side length 1 for the following charge distributions:

a. One point charge of 1 at [x,y]=[0.5 0.5].
b. One point charge of 1 at [x,y]=[0.4 0.6] (What is striking us as compared to a.?)
c. Two point charges of 1 and –1 at [x,y] = [0.5 0.45] and [0.5 0.55](dipole).

For a solution use the method of multiple Fourier Transform (script fftpoi.m). Try to understand the script as far as possible.