Numerical Methods for International Masters
Version 0.75

Wolfgang Mackens

October 22, 2010

Contents

1 Introduction **3**
- 1.1 What is Numerical Mathematics? . . . 3
- 1.2 Prerequisites . . . 5
  - 1.2.1 Necessary . . . 5
  - 1.2.2 Nice to have . . . 7
- 1.3 Aims of the course . . . 8
- 1.4 Your duties . . . 9
- 1.5 Software used . . . 9

2 Computers, Numbers, Errors **10**
- 2.1 Numbers . . . 10
  - 2.1.1 Whole numbers - Integers . . . 11
  - 2.1.2 Real Numbers - floating-point-numbers . . . 11
  - 2.1.3 Complex Numbers . . . 13
  - 2.1.4 Computer arithmetic and machine constants . . . 13
- 2.2 Errors in Scientific Computing . . . 19
  - 2.2.1 Truncation Error . . . 19
  - 2.2.2 Rounding error . . . 22
  - 2.2.3 Data Errors . . . 23
  - 2.2.4 Programming Errors . . . 24
  - 2.2.5 Stupid errors . . . 24

3 Interpolation **25**
- 3.1 Introduction . . . 25
  - 3.1.1 Recovery of functions from limited information . . . 25
  - 3.1.2 General Interpolation Approach . . . 26
  - 3.1.3 The linear interpolation problem . . . 27
  - 3.1.4 Some common 1D-basis-functions . . . 30
  - 3.1.5 Standard Interpolation conditions . . . 31
  - 3.1.6 Preview: Applications . . . 33
- 3.2 "Lagrangean" Polynomial Interpolation . . . 39
  - 3.2.1 Different forms of the interpolating polynomial . . . 40
  - 3.2.2 Error estimation . . . 46
  - 3.2.3 Use of polynomial interpolation . . . 48

3.3 Spline Interpolation . . . 50
3.3.1 Piecewise linear interpolation . . . 50
3.3.2 Cubic Spline Interpolation . . . 52
3.4 Further Issues on Interpolation . . . 56
3.4.1 Trigonometric Interpolation . . . 56
3.4.2 Exponential Interpolation . . . 57
3.4.3 Rational Interpolation . . . 57
3.4.4 Extrapolation . . . 57
3.4.5 Interpolation of curves in the plane and in space . . . 57
3.4.6 Further Hints . . . 58

4 Approximation of Linear Functionals **60**
4.1 Introduction . . . 60
4.2 Quadrature Formulae for Simple Integral . . . 62
4.2.1 Introductory Ideas . . . 62
4.2.2 The Newton-Cotes Formulae . . . 64
4.2.3 Gauss-Formulae . . . 66
4.2.4 Compound Formulae . . . 69
4.2.5 Error-Estimation . . . 70
4.2.6 Adaptive Quadrature . . . 70
4.2.7 Quadrature for Integrals with Weight Functions . . . 70
4.3 Differentiation Formulae . . . 70
4.4 Extrapolative Improvement of Functional Approximations . . . 71

5 Solution of systems of linear equations **72**
5.1 Introductory Example . . . 72
5.1.1 A Simple Truss . . . 72
5.1.2 Governing Equations . . . 73
5.1.3 System of Equations . . . 74
5.1.4 A Combination Problem . . . 75
5.1.5 Sparse Matrix Formulation . . . 75
5.2 Solvability of General Linear Systems . . . 76

Chapter 1

Introduction

1.1 What is Numerical Mathematics?

Numerical Mathematics means doing mathematics by using *NUMBERS* for your calculation and getting out numbers describing the results of your computations.

Hence you did Numerical Mathematics,when you learned the multiplication tables as a first grader?

No. In fact you did not. You did not do mathematics then,is was just calculating.

In order to be able to do Numerical Mathematics, first you must have done real mathematics, including analysis, linear algebra, some analytical treatment of differential- and might be of integral-equations. And if you then try to execute all that mathematics on a computer, you will see, that you will have to perform Numerical Methods in quite some cases.

During their lectures on analysis your professors spent quite some time to make you acquainted with "analytic items" like "limit point" and "accumulation point" or "continuity" and "continuum" or "closedness of sets" or even "completeness of spaces" or "differential quotient" and "integral". All these concepts inherit some good portion of "infinity", and it took you quite a heavy portion of work to finally understand that these concepts were not only constructed for mathematicians to have their fun with but instead to successfully describe the world, at least the world of scientists and engineers[1]. One of the last items that you learned to know in the maths courses of your Bachelor-education were most probably connected with the theory of ordinary differential equations (ODE), might even be you encountered partial differential equations (PDE) together with modeling examples from process engineering such as describing reaction dynamics, transport of reacting material through a reactor, diffusion through porous catalysts, flow through pipe systems and other problems of that kind.
Furthermore you may have been dealing with the problem of minimizing real functions, together with examples wherein the function to be minimized was a cost-functional and its minimization meant max-

[1] All other persons do in fact live in the same world. Though they would not believe that their world is governed by mathematics such as differential equations and large linear systems of equations and optimization methods - actually they even don't know the words - they trust in the engineers art by unconcernedly filling high explosives into the tanks of their cars, drinking water from the river Elbe, that engineers cleaned for them and traveling in a hight of more than 30.000 feet in planes, albeit everybody knows how it hurts to fall on ones knees from a hight of only 3 feet. "In Engineers they trust". - Actually, since I educate engineers, I am always a little nervous, when I have to go to a farther place by plane.

imizing the profit that you could realize with your chemical plant. Now, entering the master-part of your studies, you have come close to really computing solutions of such real life mathematical problems associated with your subject of study[2].
Now, that you are on the brink of really performing the construction of such mathematical models, I will tell you, that all that "infinity junk" can not really be carried out.
It is actually not that simple to prove, that the number π has an infinite number of decimal digits and that there is no periodicity in them[3]. But in practice this means, that you will not be able to store π on any computer, since every computer has only a finite number of storage places.
More general, if you have any real sequence $\{a_k\}_{k\in\mathbb{N}}$ which converges to a limit a, which is not a rational number, you will have no chance to really compute a, since the execution of the infinite number of values would take an infinite time[4]. Most of the functions that you can compute values of with your pocket calculator[5] are in fact defined as the limit of infinite series, such that they should not be calculable in finite time.
Finally, the solutions of most differential equation problems[6] can not be expressed in terms of all the functions that mathematicians[7] introduced during the last centuries. Though - under mild conditions on the differential equation - existence and uniqueness theorems[8] state that these solution do exist.
But what the hell do all these pocket calculator do, when showing numbers for the cited function values? What does it mean, when - as we will see before short - MATLAB gives us a perfect drawing of the solution of a differential equation.
The answer is easy: All the pocket calculators as well as MATLAB and all the other numerical software programmes are cheating. What they give you is not the solution to the problem, but in fact only an approximation to the real solution. As long as the approximation is of sufficient quality, this is enough for the practitioner[9]. When you press the π-button on your calculator, it probably shows the decimal places 3.141592653589793 or less than that. Well, that's in fact not π, but it is an approximation which is correct to all the places that you can see on the display.
So, what numerical mathematics does is to calculate finitely many results in an finite time[10] with a finite precision and to see how these data can satisfy the customer.
If, e.g. these results have to stand for the solution function $v(t), t \in [0,1]$ of a differential equation

$$v'(t) = v(t)^2 \cdot \sin(v(t) \cdot t), t \in [0,1] v(0) = 0.1,$$

which involves more than a countably infinite number of conditions[11] for the more than countably infinite degrees of freedom of $v(t), t \in [0,1]$, then the original problem to determine a "continuum" of values has to be replaced by a "discrete" substitute with a finite number of unknowns and likewise a finite number of equations. Many processes of this kind will be discussed in the text under the common name "discretization."

[2] To endow you with some of the methods to do so is actually the aim of this course.
[3] Maybe, it was not easy as well to believe it. But you do by now, don't you?
[4] Psalm 90,10: The years of our life are seventy, or even by reason of strength eighty; yet their span is but toil and trouble....
[5] Such as exponential function, sine and cosine function, all the other function from trigonometry, ...
[6] not artificially constructed for means of eduction
[7] and all the many other persons doing mathematics professionally (this includes a lot of engineers as well)
[8] We will refresh your knowledge on Peano's and Picard an Lindelöf's theorem in the sequel.
[9] Have you ever thought of adjusting a potentiometer to π?
[10] Actually, we are striving to do the job as fast as possible. It might be a calculation during the landing process of a plane, and if it takes too long to get the information on how to change the drop rate in hight it might be too late to prevent a crash.
[11] In fact the condition $v'(t) = v(t)^2 \cdot \sin(v(t) \cdot t)$ for every $t \in [0,1]$.

Such discretizations have of course to be constructed in such a way that

- information about the actual solution can be gained from the calculated data to a sufficient quality,
- this information is gained with as small computing time and computer storage as possible[12]

The development of numerical mathematics started early in the 40th of the last century together with the development of electronic computers. Similar to their improvement the numerical methods have been improved as well and are still being improve. Actually, the gain in computing time per year due to new algorithms is at least as large as the gain through improved speed of the computers.
This means that "Numerical Methods" will certainly change considerably during the years following your studies at the university. This implies that you will not only have to renew your computer equipment every three years if you want to be up to date and competitive but you will have to continuously improve your knowledge on numerical mathematics and available software and see whether there have been new developments in the field.
Engineers who missed the development of stiff solvers for ordinary initial value problems[13] in the late 60s or the 70s of the last century were punished with factors up to 1000 in the computing time for certain problems.

1.2 Prerequisites

1.2.1 Necessary

Calculus

For this course it is assumed that you at least have a good knowledge of elementary real analysis including the rigorous study of derivatives and integrals of functions of real variables. A basic requirement for the study of these in the knowledge of sequences and series and their limits.
The usual knowledge includes: derivative of a constant, sum rule in differentiation, constant factor rule in differentiation, linearity of differentiation, calculus with polynomials, chain rule, product rule, quotient rule, inverse functions and differentiation, implicit differentiation, stationary points (maxima and minima, first derivative test, second derivative test) Taylor's Theorem, l'Hôpital's rule, Leibniz's rule, mean value theorem, logarithmic derivative, antiderivative, indefinite integral, sum rule in integration, constant factor rule in integration, linearity of integration, arbitrary constant of integration, fundamental theorem of calculus, integration by parts, inverse chain rule method, differentiation under the integral sign.

Vector Calculus

In vector calculus (or vector analysis), being the branch of mathematics concerned with differentiation and integration of vector fields (primarily in 3 dimensional Euclidean space $\mathbf{R}^3$) you should remember the notions of partial differentiation and of multiple integration.

[12] In most cases, this is to be read as "as possible for the scientist in charge". It will not be possible in general to find an optimal solution, an algorithm which cannot be beaten any more. Actually, one is striving for a solution which is as good as one can do at the moment or one looks for a method which is at least provably better than existing ones. We usually know that sooner or later there will be someone who will - may be taking our developments as a fundament, may be showing that our approach lead to a dead end of research - improve our work. There are not too much (and not really very complicated) problems, where an final optimal algorithm is known. The part of computer science which deals with investigating problems intrinsic complexity is "Complexity Theory". We will not be concerned with this issue here.

[13] See Section **??**.

Functions

The construct of a function[14] should be very clear to you. In addition to elementary functions of real numbers, in the abstract set-theoretic setting a function is a relation between the domain, say D, and the codomain, say C, that associates each element in the domain with exactly one element in the codomain. If we name the function, say by f, then we write

$$f : D \longrightarrow C$$

as a shorthand notation for the above-described context. If we have a description of the way the function works,we can add it to the description in a second line as in the following example, which "colours the positive integers".

$$C : \begin{cases} \mathbb{N} & \longrightarrow \{\text{red}, \text{black}\} \\ n & \longmapsto \begin{cases} \text{red} & \text{if } n \text{ is even} \\ \text{black} & \text{if } n \text{ is odd.} \end{cases} \end{cases}.$$

Notice that x^2 is **not a function** but just some algebraic term. This term can of course be used to define a function by the operation of multiplying an argument by itself. But it is not a function as long as it is not yet declared which sets are to be used as domain and as codomain of the function[15].
It is necessary that the notion of a function it totally clear to you, since modelling of entities from engineering sciences in MATLAB is mostly done in terms of functions.
In this connection it is useful to remember the composition

$$(g \circ f)(x) := g(f(x))$$

of two function f and g, since most models will be built in a modular way and the fusion of modules will very often been done by composition.

Differential Equations

You should have a first idea of what a differential equation is. You should know elementary solution techniques for special first order equations like separation of variables and variation of constants techniques as well as the solution of linear systems of first order $y' = Ay$ using the Jordan normal form of the matrix A.

Elementary Linear Algebra

Vectors in real n-space $\mathbb{R}^n$, subspaces of $\mathbb{R}^n$, lines and planes and hyperplanes and their various descriptions, linear maps, the representation of maps by matrices, matrix calculus, Gaussian elimination, solvability of linear systems, solution spaces of general (underdetermined,overdetermined, non singular) sets of equations should be known. Linear dependence and linear indencence of a set of vectors should be known.
The Euclidian inner product (dot product) and its connection to orthogonality should be totally clear.
Determinants of square matrices should be known. You must be able to compute them for determinats of matrices of small size.

[14] Which is of course a part of elementary calculus.

[15] The operation of "squaring" can be used for different entities among which are natural numbers, real numbers, complex numbers, real and complex matrices, elements of mathematical groups,subsets of alle these and so on and so on. And at the same time, the codomains are not clear at all. Every suitable choices of domains and codomains will result in different functions "x^2".

Eigen Theory

You should know what eigenvalues and eigenvectors of a matrix are. You shoold know how tocompute them by hand. You should know, what their use in linear systems of first order differential equations is.

Computer Programming

You should know the basic ideas of computer programming using whatever programming language. You should at least know, what loops are and how they are controlled by commands like `for` or `while` or similar and you should know about conditional execution of statements controlled by commands like `if` and `end` or `else` or `elseif` and so forth.

1.2.2 Nice to have

Vector Calculus

It would be useful, if you were able to parametrize curves and surfaces in 3-space.

Differential Equations

It would be nice if you have some theoretical background including the existence- and uniqueness-theorems of Peano and of Picard and Lindelöf. As well every knowledge on stability analysis of stationary solutions and the description of periodic solutions in the phase space will be helpful to you. Furthermore it would be useful if you knew already how to transform an explicit differential equation of n-the order to a system of first order. We will be dealing with these subjects rather quickly in a recapitulating way.

Elementary Linear Algebra

Euclidian inner product, the notion of orthogonality as well as orthogonal projections onto subspaces will be repeated for the 3D-space and in the general setting of Euclidian vectorspaces.
Dyadic products of vectors will be dealt with in a recapitulating fashion as well.
It would be advantageous for you if you had a vague idea of these issues already.

General Vector Spaces, Norms, Banach- and Hilbert Spaces

As a generalization of the three-dimensional real space we will very briefly introduce you to the notion of a general (finite dimensional) vector space. We shallbe dealimg with the definition of a "norm" as a generalization of the usual Euclidian length of a vector and the notion of a general inner product. We will define a Banach Space as a complete normed vector space, as well as a Hilbert Space, which is a Banach space, where the norm is being induced by an inner product. As special norms on $\mathbb{R}^n$, we will discuss the $1-, 2-$ and ∞-Norm of a vector.
If you do not know yet, you will learn very quickly, that the set $F(S, V)$ of functions f from a general set S to a vector-space V is a vector space itself if addition of two function and multiplication by a scalar λ is defined pointwise via

$$(f_1 + f_2)(x) := f_1(x) + f_2(x), \quad \forall \quad x \in S$$

and

$$(\lambda f)(x) := \lambda f(x), \quad \forall \quad x \in S.$$

For the case that S is a square or cube from $\mathbb{R}^2$ or $\mathbb{R}^3$, respectively, the generalization of the Euclidian inner product and the 1-, 2-, and ∞-norms to be used on adaquate subspaces of $F(S, V)$ will be needed in the course.
Finally you will be well off if you know the notion of an operator norm before we introduce it in the course.

Eigen Theory

It would be very helpful if you knew the spectral decomposition of normal matrices and the Jordan normal form of general matrices.

Implicit functions

Finally, the name of the Implicit Function Theorem should be remembered by you and you should have a rough idea what Banach's Fixed Point Theorem is all about.

1.3 Aims of the course

After having successfully completed the course you should be able to

- solve linear and nonlinear systems of equation efficiently,
- interpolate given data by polynomials, by elements of general vectorspaces of functions, and by splines,
- compute integrals and derivatives of function numerically,
- compute approximate solutions of integral equations of the second kind,
- fit parameter dependent functions in the least squares sense to given data,
- solve constrained and unconstrained nonlinear optimization problems,
- solve eigenvalue problems,
- approximate the solution of ordinary differential equations (ODE), both subject to initial and to boundary conditions,
- study the stability of stationary points of ODEs as well as depict and interpret phase space diagrams,
- follow solution branches of parameter-dependent systems of equations,
- solve parabolic and hyperbolic initial-boundary value problems by means of the method of lines.
- solve special elliptic problems through discretization using the finite difference method.

1.4 Your duties

The course consists of three hours lecture every week and two hours hands-on-computer-exercises. During the lectures the material will be explained, which is necessary to solve the practical problems dealt with in the exercises. In order that we can explain every new item and have time to recapitulate some (more complicated) items which should have been in most bachelor-math-courses (but which might not be fully understood yet) it is necessary that you read the corresponding material in these lecture notes carefully at least once before you come to the lecture.
It might occur that some of the facts which we assume you to know are in fact unknown to you. It is essential that you teelltzhis the lecturer as soon as you realize this gap in your knowledge. He will try to fill it or will give you advice how you can fill it on your own.
If it turns out that you have too many gaps, he will advise you to join the course one year later again and he will try to suggest material with which you could close the gap. It is undouptably preferrable to delay the course than to spend all Wintersemester five hours a week without being able to understand.
There will be texts for the exercises handed out before the exercise-lessons. You will have to read them before the lessons and try to solve those parts which are labeld as preparatory homework.
Finally, you have to participate in the class, such that at the end of the course you are able to solve really every exercise.

1.5 Software used

As you might have inferred from the previous text we will do most of our numerical work by using the MATLAB environment, which integrates programming, computation and visualization.
You should install MATLAB on your home computer in order that you are able to do the homework.

Chapter 2

Computers, Numbers, Errors

2.1 Numbers

Computer calculation with MATLAB may involve quite some different types of data. The data which represent numbers, namely integers (whole numbers), floating point numbers ("real numbers"), complex numbers (pairs of floating-point numbers with complex arithmetic) and matrices (including vectors and multidimensional arrays) built out of these numbers will be the ones we are interested in during this course[1].
For each of these numbers there is only a fixed finite number of storage places reserved, such that for each sort of numbers there are only finitely many of them. Since in mathematics there are clearly countably many integers and more than countably many real and complex numbers, there must occur conflicts, when trying to mimic mathematical calculation by computer calculation.
Though most computers internally will use binary digits and hence try to store an integer N of the form

$$N = \pm b_m 2^m + b_{m-1} 2^{m-1} + \cdots + b_1 2^1 + b_0 2^0$$

by storing the sign and the sequence of binary digits

$$N \sim \pm b_m b_{m-1} \dots b_1 b_0$$

we will use for our treatment the decimal system[2]

$$N = \pm d_k 10^k + d_{k-1} 10^{k-1} + \cdots + d_1 10^1 + d_0 10^0.$$

Actually, there are programmes in every computer that translate between both systems, such that the user has the impression to have a decimal-system-using computer. We will neglect the minor discrepancies that might occur due to this translation.

[1] The other data like logical variables, characters, cell arrays and structures - see [?], page 16, are of no importance for us. Maybe strings out of characters can be of some use for us in editing results.

[2] Due to our ten fingers we like this system better.

2.1.1 Whole numbers - Integers

In MATLAB there are different ways to store integers. You can choose to store integers in 8 bits (binary digits) or 16 bits or 32 bits or 64 bits. With these you can represent the integers from -128 up to 127 or -32,768 up to 32,767 or -2,147,483,648 to 2,147,483,647 or -9,223,372,036,854,775,808 up to 9,223,372,036,854,775,807. These integer storing modes have to be explicitly declared as INT8 up to INT64. Actually, you won't use them in numerical computations[3].
In numerical applications integers are usually stored in double 64-bit floating point storages. These will be used automatically by MATLAB for indices or for loop variables. Actually, you have not as much of them as the $2^{64} - 1$ INT64-integers, namely -2^{53} up to 2^{53}. But this is certainly enough for these applications[4].
In ancient times[5] one used integer variables to mimic computation with real numbers by storing with the integer a position where the decimal point had to be set. The number 987.654321 would have been stored as 987654321 and the number 3, indicating that the decimal dot had to be placed after the third decimal. These real numbers - better the way to store information about real numbers - where called *fixed point numbers*[6].
For scientific purposes the storage scheme of the *floating-point numbers* together with an dapted arithmetic is far better suited. We will explain both next.

2.1.2 Real Numbers - floating-point-numbers

As we discussed before, with a finite number symbols, only a finite set of numbers can be represented. The number representing real numbers on a computer are called machine numbers. They are also called floating point numbers and their specific distribution depends on the computer and on the programming language. A decimal floating point number f has the form

$$f = m \times 10^e.$$

m is called the mantissa and e the exponent. If $m \neq 0$, then usually m is a signed decimal number with one decimal to the left of the decimal point[7] The exponent e, a signed integer, is adjusted correspondingly. Both for m and for e there are - besides storage places for their signs - storage places for a fixed number of decimal digits reserved[8]. Hence, if we assume as a model case[9], that m may have three digits and e one, then one could store the numbers

$$3.14, \qquad -2010, \qquad 0.00000345$$

as follows

$$3.14 \cdot 10^0, \quad -2.01 \cdot 10^3, \text{ and } \quad 3.45 \cdot 10^{-6},$$

[3] You would use them in signal processing, image processing etc., whenever you need large amounts of these entities.
[4] The reason for MATLAB using these so called *flints* (= FLoating point INTegerS) is a little bit special and will be omitted.
[5] This is when I started to learn numerical methods and computer programming
[6] If you just want to calculate with Euros and Cents, it makes sense to calculate everything in Cent and - only for printing - put a decimal dot left from the two farthest right digits.
[7] There are other normalizations, e.g. letting the leading nonzero decimal be the element right to the decimal point.
[8] These number varied considerably for quite some time depending on which computer you where using.
[9] We will make repeated use if this model lateron.

respectively[10].

Notice that there are only 34201 different such numbers, which lie in the set $[-9.99 \cdot 10^9, -1.00 \cdot 10^{-9}] \cup \{0\} \cup [1.00 \cdot 10^{-9}, 9.99 \cdot 10^9]$.
If one wants to store a real number $R \neq 0$ from input or some calculation on the computer by use of the above given machine numbers, then probably the best to do is to chose that machine number R_m with shortest distance to R. If the representation of R is normalized such that

$$|R| = r_1.r_{-1}r_{-2}r_{-3}r_{-4}... \cdot 10^E$$

with decimal digits $r_i \in 0, 1, \ldots, 9, i = 1, -1, -2, ...$ and $r_1 \neq 0$ and

$$|R| \in [0.995 \cdot 10^{-9}, 9.995 \cdot 10^9] \tag{2.1}$$

then the next machine number is found by rounding[11]

$$R_m = \text{sign}(R) \cdot 10^E \cdot \cdot \begin{cases} r_1.r_{-1}r_{-2} & \text{if } r_{-3} < 5, \\ r_1.r_{-1}[r_{-2}+1] & \text{else.} \end{cases} . \tag{2.2}$$

In any case - if (2.1) is satisfied - the relative error

$$\frac{|R_m - R|}{|R|} \leq \frac{0.005 \cdot 10^E}{r_1.r_{-1}... \cdot 10^E} \leq 0.005$$

is always bounded by 0.005.
Actually, the floating point numbers are designed to uniformly control the relative error, when storing a number[12].
Please notice, that the latter is not the case if condition (2.1) is violated. Both if $|R|$ is either larger than the upper bound of the interval (2.1) or less than its lower bound, the choice of the nearest machine number will no more guarantee, that the relative error is bounded by 0.005. If the first case occurs after a computation the computer normally signals an *overflow* and in the latter an *underflow*.

The number which bounds the relative error when "storing" numbers in the above way[13] is called the "machine precision" or the "machine epsilon" and denoted bei $\varepsilon_{\text{Mach}}$ or simply `eps`.
From the above it should be clear, that

$$\varepsilon_{\text{Mach}} = 0.5 \cdot 10^{-n}$$

[10] One would call the arithmetic, connected with these three-digit-mantissa-numbers a three-digit-floating-point-arithmetic. Normally, the exponent is large enough, that the errors caused by this restriction due not occur as regularly, as those coming from finite length of the mantissa. If problems occur due to a too short exponent, such problems are severe. See *under-* and *overflow* below.

[11] Notice that the rounding formula (2.2) in case of equal distances to the larger an the smaller machine number decides to choose the number with larger absolute value.

[12] Notice that in scientific computation it is always the relative error that really is of interest: If someone tells you, that he measured a length with an error not larger than 1 meter, than you would certainly ask, what he measured. If he would say the size of his shoes, you would certainly call in nuts. If on the other hand he would say that he measured the distance of a footprint in his garden from the footprint of Neil Armstrong's first footstep on the moon, you would declare him nuts as well, but the reason would be completely different.

[13] assuming that no over- or underflow occurs

for n-digit-floating-point-storage.
The machine epsilon is of course dependent on the way the computer stores numbers. In order to be able to write machine independent programmes, MATLAB delivers the actual machine epsilon (for normal double real calculations) on the variable `eps` (As long as you do not redefine it.) as `eps` $= 2^{-52} \approx 2.220446049250313 \cdot 10^{-16} < 0.5 \cdot 10^{-15}$. This means that one can calculate as if on had a machine with a 15-decimal-digit-mantissa.

2.1.3 Complex Numbers

Complex numbers are stored as a pair of floating point numbers, where the first i interpreted as the real part and the second as the imaginary part of the corresponding complex number. MATLAB makes these accessible as one complex variable. As long as the variable i is nor redefined, it is the imaginary unit[14]. Hence if you want to store the complex number $z = 2 + 4i$ you just enter:

$$z = 2 + 4 * i.$$

2.1.4 Computer arithmetic and machine constants

If one performs elementary operations[15] with machine numbers on a computer, the result will in general not be a machine number.
Examples (using our above three-decimal-digits-storage) :

$$\begin{array}{lcll}
1.00 \cdot 10^0 + 1.11 \cdot 10^{-6} & = & 1.00000111 \cdot 10^0 & \text{Mantissa too large,} \\
1.00 \cdot 10^0 - 1.00 \cdot 10^{-6} & = & 9.99999 \cdot 10^{-1} & \text{mantissa too large,} \\
9.99 \cdot 10^0 * 9.99 \cdot 10^0 & = & 9.98001 \cdot 10^1 & \text{mantissa too large,} \\
1.00 \cdot 10^8 * 1.00 \cdot 10^7 & = & 1.00 \cdot 10^{15} & \text{exponent too large,} \\
1.00 \cdot 10^{-8} * 1.00 \cdot 10^{-7} & = & 1.00 \cdot 10^{-15} & \text{exponent too small,} \\
1.00 \cdot 10^0 / 3.00 \cdot 10^0 & = & 0.33333333... \cdot 10^0 & \text{mantissa too long,} \\
1.23 \cdot 10^0 + 2.44 \cdot 10^0 & = & 3.67 \cdot 10^0 & \text{everything allright.}
\end{array}$$

The results have to be stored as machine-numbers, of course. Thus they have to be replaced by adequate machine numbers. In the cases of underflow, the substitute will be the zero (and there will be a notice to the user, that an underflow occurred). In case of an overflow there will be a massage as well and (depending on the compiler that you use) the result will be the sign of the result times the largest floating-point number (this is done in MATLAB as well, as long as the result is less than about the square of the largest floating-point number) or it is set to a nonnumerical value "Inf" for infinity. Thus in MATLAB infinity is as close as 10^{599}.
For the the above examples we would get the results

$$\begin{array}{lcll}
1.01 \cdot 10^0 + 1.11 \cdot 10^{-6} & \approx & 1.00 \cdot 10^0 & , \\
1.00 \cdot 10^0 - 1.00 \cdot 10^{-6} & \approx & 1.00 \cdot 10^0 & , \\
9.99 \cdot 10^0 * 9.99 \cdot 10^0 & \approx & 9.98 \cdot 10^1 & , \\
1.00 \cdot 10^8 * 1.00 \cdot 10^7 & \approx & 9.99 \cdot 10^9 & \text{Overflow!} \\
1.00 \cdot 10^{-8} * 1.00 \cdot 10^{-7} & \approx & 0.00 \cdot 10^0 & \text{Underflow!} \\
1.00 \cdot 10^0 / 3.00 \cdot 10^0 & \approx & 3.33 \cdot 10^{-1} & , \\
1.23 \cdot 10^0 + 2.44 \cdot 1^0 & = & 3.67 \cdot 10^0 & .
\end{array}$$

[14] So is j.
[15] I.e. addition, subtraction, multiplication and division

Notice that in the first two cases the first summand does not change by adding or subtracting the second one. Normally the machine epsilon is defined as the smallest number, that will change the number 1 by adding it to 1. In three digit decimal arithmetic this is of course (as defined above) the number 0.005, because 1.005 will be rounded to 1.01, whereas everything less than 0.005 will be neglected after addition.
If x and y are positive numbers with $x \geq y$, then

$$x + y = x\left(1 + \frac{y}{x}\right)$$

shows, that in computer arithmetic we have

$$x + y \approx x$$

as long as

$$\frac{y}{x} < \texttt{eps}$$

i.e.

$$y < x \cdot \texttt{eps}.$$

If a number x is read into the computer, then the relative error δ_x between x and the corresponding machine number x_{mach} satisfies

$$\frac{x_{mach} - x}{x} = \delta_x \text{ with } |\delta_x| \leq \texttt{eps}.$$

Hence one may write

$$x_{mach} = x(1 + \delta_x) \text{ with } |\delta_x| \leq \texttt{eps}.$$

This is very useful for the analysis of "error propagation", i.e. the accumulation of existing errors during operations involving several faulty entities.
If, e.g., x and y are positive and $x_{mach} = x * (1 + \delta_x)$ and $y_{mach} = y * (1 + \delta_y)$ with $|\delta_x| < \texttt{eps}$ and $|\delta_y| < \texttt{eps}$ then

$$x_{mach} + y_{mach} = x * (1 + \delta_x) + y(1 + \delta_y) = (x + y) + x\delta_x + y * \delta y = (x + y) + (x + y)\delta_{x+y}$$

with a δ_{x+y} satisfying $|\delta_{x+y}| < \texttt{eps}$.
Hence addition behaves stable in the sense that the addition of two machine numbers[16] gives a result which resembles to the addition of the true numbers with subsequent rounding.
For multiplication and division the relative errors will (approximately) add because of

$$x_{mach} * y_{mach} = x * (1 + \delta_x) * y(1 + \delta_y) = (x * y) * (1 + \delta_x + \delta_y + \delta_x \cdot \delta_y) \approx (x * y) * (1 + (\delta_x + \delta_y))$$

and

$$\frac{x_{mach}}{y_{mach}} = \frac{x * (1 + \delta_x)}{y(1 + \delta_y)} = \frac{x}{y} * \frac{1 + \delta_x}{1 + \delta_y} \approx \frac{x}{y} * (1 + \delta_x)(1 - \delta_y + \delta_y^2 - ...) \approx \frac{x}{y} * (1 + (\delta_x - \delta_y)).$$

Care has to be taken, when nearly equal numbers are being subtracted. To see this we will use an example with our above three-decimal-mantissa-numbers and assume that the machine numbers to be subtracted are

$$x_{mach} = 1.01 \text{ and } y_{mach} = 1.02.$$

[16] of equal sign

and that

$$x_{mach} = x(1+\delta_x) \text{ and } y_{mach} = y(1+\delta_y).$$

Then

$$\frac{(y_{mach} - x_{mach}) - (y-x)}{y-x} = \frac{y\delta_y - x\delta_x}{x-y}$$

Due to Murphy's law, the nominator of this fraction can be as large as $2 \cdot \texttt{eps} = 0.01$. Since $x = 1.015$ and $y = 1.015$ are in accordance with

$$|x - x_{mach}| = 0.005 = \texttt{eps} \text{ and } |y - y_{mach}| = 0.005 = \texttt{eps}$$

the denominator could b zero, such that the relative error would be infinite.
Even if we assume, that $|y - x| = |y_{mach} - x_{mach}| = 0.01$, the relative error would be 1, meaning, that no digit of the result would be correct.
This bad behavior is know as **catastrophic cancelation**. Hence, one should avoid subtraction of nearly equal numbers.

Example 1: The polynomial $r(x) = (x-2)(x-10^{-20})$ obviously has the zeros $x_1 = 2$ and $x_2 = 10^{-20}$. If we do the multiplication of the two factors of $r(x)$ we arrive at

$$r(x) = x^2 - 2.00000000000000000001\, x + 2 \cdot 10^{-20} \approx \hat{r}(x) := x^2 - 2\,x + \texttt{2e-20},$$

where $\hat{r}(x)$ is what we can store under MATLAB.
Remark 1: `1.234565432e-20` is the way you enter the number $1.234565432 \cdot 10^{-20}$ into the computer, since you have to write everything within one row of symbols. We will henceforth use this way of writing, whenever it is appropriate.

Most German students would use the formula

$$x_{1/2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q} \tag{2.3}$$

for the roots of

$$x^2 + px + q = 0$$

to find the roots of $\hat{r}(x) = 0$.
This results in

$$x_{1/2} = 1 \pm \sqrt{1 - \texttt{2e-20}} \approx 1 \pm \sqrt{1} = \begin{cases} 2 & \text{in case of } + \text{ ,} \\ 0 & \text{in case of -.} \end{cases}$$

Thus the addition gives a correct result, whereas the subtraction delivers 0, which inherits an error of 100%.
Notice that the subtraction could have been avoided by using Viete's law, which says, that the product of the two roots gives the term $q =$ 2e-20, which is independent of x. If we know $x_1 = 2$ by use of the "addition-part" of formula (2.3) then we can calculate

$$x_2 = q/x_1 = 1e - 20,$$

which is the correct result.

Example 2: Another occasion, where you **have to** substract nearly equal numbers is "numerical differentiation". The differential quotient

$$f'(x_0) := \frac{df}{dx}(x_0) := \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

is defined as the result of a limit process, which to execute - as we know by now - we do not live long enough. A remedy is not to execute the limit but to be content with a "finite difference"

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} \tag{2.4}$$

with a small but positive real number[17] $h > 0$.
Since $f'(x_0)$ would be the theoretical result for $h \to 0$ at a first glance one might conjecture, that it would be best to choose h as small as possible, to use the
smallest possible positive real value, accessible as `realmin` in MATLAB. But already a little bit of thinking[18] shows that this is not the right way to go. If we assume x_0 to be of the oder of magnitude of 1, then $x_0 + h$ would not be different from x_0 for $h <$ `eps` and thus every estimate of a derivative would be equal to zero.
Even if we chose values greater than `eps`, this would not necessarily lead to good approximations of the derivative.

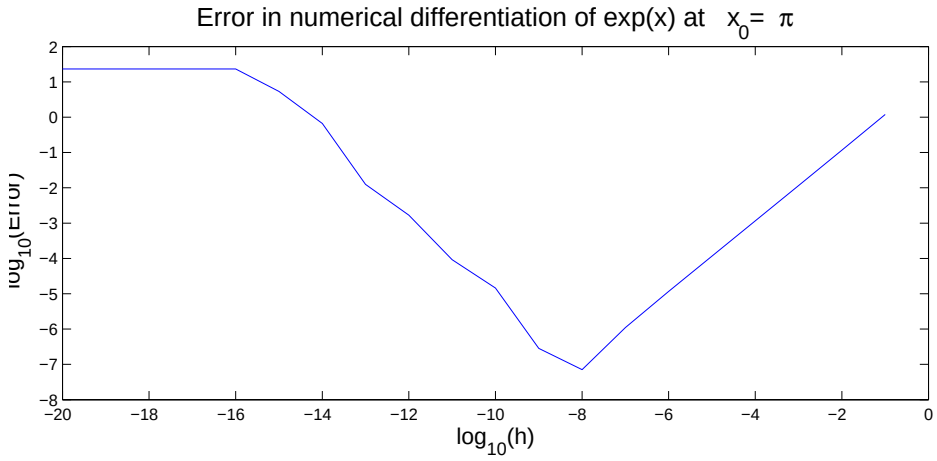

Figure 2.1: Observed error of onesided finite difference approximation of first derivative

[17] Just on the fly let us remark that small positive numbers in numerical mathematics are in most cases named h. Actually, I do not know whether there is a connection with Planck's constant, or whether there is another reason for that. I always joke that it is the abbreviation of the German "Häppchen". Whoever knows better about this, please let me know.
[18] Which may have been initialized by some trial calculations.

Figure 2.1 shows the computational results for the approximation error

$$\text{Observed\_Error}(h) \approx \frac{f(x_0+h)-f(x_0)}{h} - f'(x_0)$$

with $f(x) = [\exp(x)]_{mach}$, $x_0 = [\pi]_{mach}$ and for $h = 10^{-20}, 10^{-19}, ..., 10^{-1}$ in a double-logarithmic plot. As can be seen, for relatively large h-values the error decreases with falling h. But at $h \approx 10^{-8} \approx \sqrt{\texttt{eps}}$ it starts to increase. At values lower than `eps`, the difference always gives the value zero, such that the error is 100% for all $h < \texttt{eps}$.
What is the reason for this phenomenon?

First, let us investigate what analysis can say about the error. If we use Taylor's series with Cauchy's error estimate[19] to expand $f(x_0+h)$ at x_0 we get from

$$f(x_0+h) = f(x_0) + f'(x_0)h + f''(\zeta)\cdot\frac{h^2}{2!} \text{ for some } \zeta \in (x_0, x_0+h)$$

the following description of the theoretical error:

$$\text{Trunc}(h) = \left(f(x_0) + f'(x_0)h + f''(\zeta)\cdot\frac{h^2}{2!} - f(x_0)\right)/h - f'(x_0) = \frac{f''(\zeta)}{2}h. \tag{2.5}$$

This **truncation error** is the error, which we would make already by replacing the "differential quotient" by the "difference quotient". We se that this theoretical error decreases like h^1 does. Because of the first power of h describing this error behavior[20] we call the onesided divided difference an approximation of $f'(x_0)$ of the first (error) order.
If one adds the values for the truncation error to the last picture (see Figure 2.2) one observes that the decreasing behavior of the observed error is precisely that of the truncation error for larger h-values.

So, what destroys this amenable theoretical feature within practical computation?

It is the rounding error, when calculating and storing the values $f(x_0)$ and $f(x_0+h)$. Actually, not these values go into the formula, but instead we have to calculate with $f(x_0)(1+\delta_1)$ and $f(x_0+h)(1+\delta_2)$ with $|\delta_i| \le \texttt{eps}$. Hence with Error$(h)$ from (2.5) we have

$$\text{Observed\_Error}(h) = \frac{f(x_0+h)(1+\delta_1)-f(x_0)(1+\delta_2)}{h} - f'(x_0) = \text{Trunc}(h) + \frac{f(x_0+h)\delta_1 - f(x_0)\delta_2}{h}$$

Since $f(x_0+h)$ is close to $f(x_0)$ we can approximately bound the second "rounding error" by

$$|\text{Round}(h)| := \left|\frac{f(x_0+h)\delta_1 - f(x_0)\delta_2}{h}\right| \le 2\cdot|f(x_0)|\cdot\frac{\texttt{eps}}{h}$$

If we add this rounding error to the last picture (see Figure 2.3), we have the explanation for the behavior of the total practical error: It is the sum of rounding and truncation error[21] From this we can estimate

[19] see below for a proof of this fact
[20] We will see below, hat powers of higher order are possible and aimed for.
[21] Actually, our bound of the rounding error overestimated the true value a little bit.

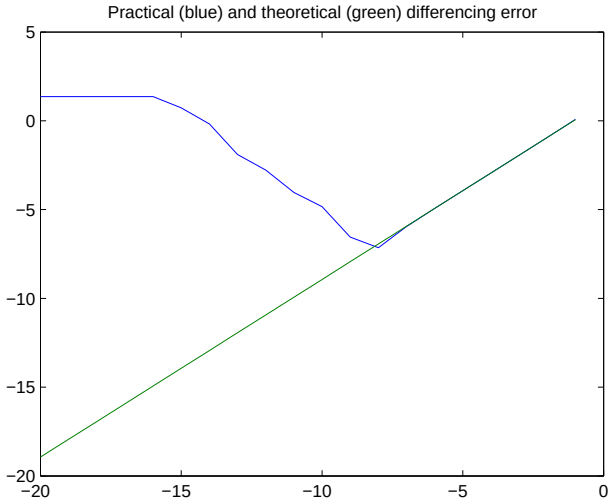

Figure 2.2: Theoretical (truncation) and observed errors of onesided finite difference approximation of first derivative

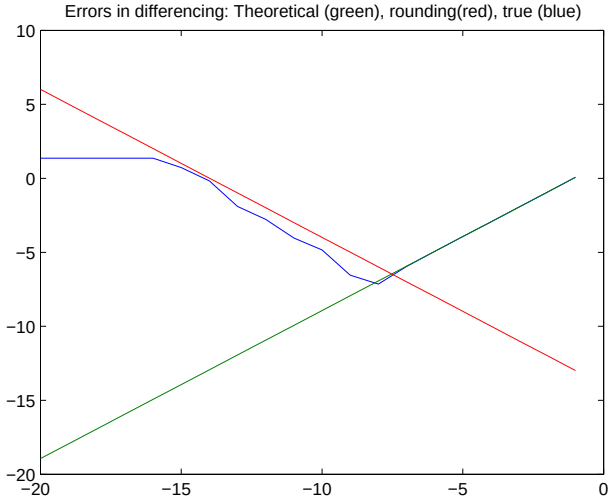

Figure 2.3: Truncation, rounding and observed errors of onesided finite difference approximation of first derivative

an optimal h-value, which minimizes the sum of both. As is seen from the last figure, this value is close to the point where rounding and theoretical error are equal

$$2 \cdot |f(x_0)| \cdot \frac{\texttt{eps}}{h} \approx \text{Round}(h) = \text{Trunc}(h) = \frac{|f''(\zeta)|}{2} h.$$

Hence we calculate

$$h_{opt} \approx 2\sqrt{\frac{|f((x_0)|}{|f''(x_0)|}} \cdot \sqrt{\texttt{eps}}. \tag{2.6}$$

From this example we see that theoretical considerations have always to be complemented by estimates of possible errors. There are lots of them. We shall discuss them in the next section.

2.2 Errors in Scientific Computing

Whenever you are busy you will make mistakes. "Errare humanum est", would the Roman say. When you do numerical mathematics, you are active, an hence you will produce errors as well, but these will - hopefully - be no unexpected errors, stupid mistakes which we will talk about in Subsection 2.2.5. Instead these should only be errors which you are aware of in advance - but likewise will not be able to fully avoid them. Besides some others (and the cited stupid errors) these are truncation errors, rounding errors and data errors. Programming errors belong to the category of the stupid errors, but since nowadays programmes tend to be very long, the occurrence of such errors is very likely, and there are extra techniques to detect and eliminate them.

2.2.1 Truncation Error

Truncation errors are those systematic errors which arise if one (knowingly) neglects certain items in a problem to make it solvable (in shorter time, with less effort, or generally).
As an example we just treated the case of numerical differentiation. Similar to this approach, numerical methods involve one[22] **discretization parameter**. Letting this parameter p go to a specific limit value p^* (like h going to 0 in the above example) is usually connected with the following observations

- The closer p is to p^* the closer is the approximate solution $s(p)$ to the exact one s^*.
- $\lim_{p \to p^*} s(p) = s^*$ in a "certain[23] way"
- The closer p is to p^* the more expensive is it to calculate $s(p)$.

Example 2: Using the difference approximations of the first derivative

$$(f(x+h) - f(x))/h \approx f'(x) \tag{2.7}$$

we can for example try to approximately solve the initial value problem

$$y' = y; y(0) = 1 \tag{2.8}$$

[22] For the beginning not more than one
[23] by no means trivial

on the interval $[0, 1]$ by transforming it to a difference equation for y-values at the gridpoints

$$x_0 = 0, x_1 = h, x_2 = 2h, x_3 = 3h, \ldots, x_n = nh = 1; \text{ with } h = 1/n \text{ for some positive integer } n \tag{2.9}$$

through replacing the derivative in (2.8) by a difference and thus generating as the discrete counterpart of the differential equation (2.8) the difference equation

$$\left(\tilde{y}(x_{k+1}) - \tilde{y}(x_k)\right)/h = \tilde{y}(x_k), k \geq 0; \quad \tilde{y}(0) = 0. \tag{2.10}$$

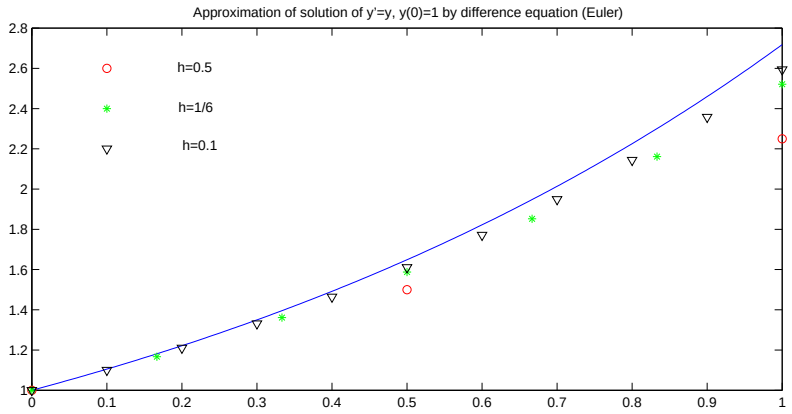

Figure 2.4: Approximate solutions of $y' = y$, $y(0) = 1$

As discretization parameter on can choose h with $h^* = 0$. On the other hand one could equally well choose n with $n^* = \infty$. In the latter case it is obvious, that with $n \to \infty$ work will increase since one has to calculate n values.

While we know from (2.5) how to describe the truncation error caused by the local replacement of the derivative by the divided difference, the analysis of the global truncation error

$$G(h) = \max_{k=1,\ldots,1/h} |y(kh) - \tilde{y}(kh)|$$

has to take into account the recurrent dependencies of the calculated values. The analysis of such more complicated "global truncation errors" is one of the main tasks of numerical analysis. In the actual simple example the solution of the difference equation (2.10) can be easily written down as

$$\tilde{y}(kh) = (1 + h)^k.$$

If we recall that $h = 1/n$ and concentrate on the gridpoint $x_n = 1$, then we have

$$\tilde{y}(1) = \tilde{y}(nh) = (1 + h)^n = \left(1 + \frac{1}{n}\right)^n.$$

Hopefully, you do recall that the right hand side is the nth element of the sequence $\left\{\left(1+\frac{1}{k}\right)^k\}_{k\in\mathbb{N}}\right\}$ which was used in Analysis to introduce as its limit the number

$$\lim_{k\to\infty}\left\{\left(1+\frac{1}{k}\right)^k\right\} = e = \exp(1) = y(1).$$

So the truncation error at $x = 1$ is

$$\exp(1) - \left(1+\frac{1}{n}\right)^n.$$

Notice however, that the result of the discretization is a vector of function values, which is different for different h-values. Thus the comparison between these vectors and the continuous solution is not that easy.
Furthermore, numerical analysis has to estimate truncation results for general differential equations, specifically those, where the solution is not available analytically. It is due to the fact that we are not able to solve analytically that we use numerical techniques.

We will not have enough time in this course to work out too much of the theory of solvers of differential equations intensively. But we will try to give hints on what are the basic ideas of this subject and we will intensively work on how to apply the solvers adequately.

Example 3: In most introductary calculus courses integrals of continuous functions are defined via Riemann-sums[24]:

$$\int_0^1 f(x)dx = \lim_{|P|\to 0}\sum_{k=1}^{n(P)} f(\zeta_k^P)(x_k^P - x_{k-1}^P) \tag{2.11}$$

where

$$P : 0 = x_0^P < x_1^P < \cdots < x_{n(P)-1}^P < x_{n(P)}^P = 1 \tag{2.12}$$

is a partition of $[0,1]$, $|P| := \max_k\left(x_k^P - x_{k-1}^P\right)$ is the "norm" or the "mesh" of the partition and the evaluation points $\zeta_k^P \in [x_{k-1}^P, x_k^P]$, for $k = 1,\ldots,n(P)$ are used to evaluate the function to be integrated. Again this is a transfinite process and the discretization has to replace the limit of sums (2.11) by a finite one. Usually this is done by using an "intregration rule", which is a weighted sum

$$\int_0^1 f(x)dx \approx \sum_{k=1}^{n} w_k f(x_k),$$

where the "nodes" x_k are[25] chosen from $[0,1]$ such that

$$0 \le x_1 < x_2 < \cdots < x_n \le 1.$$

[24] As an example we choose an integral from 0 to 1. The adaption of the process to other integration intervals is obvious. In addition this will be studied in Section 4.
[25] In most cases.

The numbers w_k are called "quadrature weights". They are designed in such a way, that the "truncation error"

$$\text{Trunc}(f) := \int_0^1 f(x)dx - \sum_{k=1}^{n} w_k f(x_k)$$

is zero for a prescribed subspace of functions[26].
A very simple example of such a "quadrature rule" is the "midpoint rule"

$$\int_0^1 f(x)dx \approx f(0.5) \cdot (1-0). \tag{2.13}$$

This is a Riemann-sum (2.11) with the partion (2.12) consisting of the boundary points of the interval and the one evaluation point in the center of the interval. It delivers the exact integral for all linear functions, i.a. polynomials of first order.
The adaption of this method to a general interval is obvious

$$\int_a^b f(x)dx \approx M(a,b,f) := f(\frac{a+b}{2}) \cdot (b-a). \tag{2.14}$$

Often a discretization parameter $h = 1/n, n \in \mathbb{N}$ is being introduces by subdividing the interval of integration (most often) in n subintervals of length h

$$[a,b] = \bigcup_{k=1}^{n} [a+(k-1)h, a+kh]$$

and applying the quadrature rule (}refMPR2) to each of the integrals over the subintervals. Thus one defines the "compound rule" :

$$\int_a^b f(x)dx \approx M_h(a,b,f) := \sum_{k=1}^{n} M(a+(k-1)h, a+kh, f) = h \cdot \sum_{k=1}^{n} f(a+(k-\frac{1}{2})h) \tag{2.15}$$

One can prove[27] that (for a function $f \in C^2[a,b]$) the truncation error fulfills

$$\int_a^b f(x)dx - M_h(a,b,f) = \frac{1}{24}(b-a)h^2 f''(\zeta) \text{ for some } \zeta \in [a,b]. \tag{2.16}$$

2.2.2 Rounding error

Rounding errors arise - as explained already intensively in Subsection 2.1.4 - through the necessity to compute with finitely many machine-numbers.
The rules for storing numbers have already been explained in the cited subsection and we have seen there, that calculations like divided differences can extremely be influenced by rounding errors. Thus one always has to be aware of such errors and one has to study whether actual calculations may amplify rounding errors.

[26] Can you imagine wht this is good for? - Think that space to be polynomials of a fixed maximal order, e.g. Then consider Taylor's expansion of a function.
[27] We will do this later.

Very often, calculations are good natured what concerns the influence of rounding errors. Assume for example that the calculation of an integral of a function (with values of the same sign on all of the integration interval) is computed by the compound midpoint rule.
If we take into account, that the function evaluation is subject to rounding errors[28] the practically computed substitute of the integral is not the righthand side of (2.15) but instead

$$M_h^{round}(a,b,f) = h \cdot \sum_{k=1}^{n} f(a + (k - \frac{1}{2})h)(1 + \delta_k), \text{ with } |\delta_k| < \texttt{eps}, k = 1, \ldots, n.$$

Hence

$$\left| \int_a^b f(x)dx - M_h^{round}(a,b,f) \right| = \left| \int_a^b f(x)dx - M_h(a,b,f) \right| + h \cdot \left| \sum_{k=1}^{n} f(a + (k - \frac{1}{2})h)\delta_k \right|$$

Since

$$h \cdot \left| \sum_{k=1}^{n} f(a + (k - \frac{1}{2})h)\delta_k \right| \leq h \cdot \sum_{k=1}^{n} \left| f(a + (k - \frac{1}{2})h) \right| \cdot \texttt{eps} = |M_h(a,b,|f|)| \cdot \texttt{eps} \approx \int_a^b |f(x)|dx \cdot \texttt{eps}$$

one can expect that the theoretical error will be disturbed by a value proportional to `eps`. In case that f does not change sign on $[a,b]$ one has

$$\int_a^b |f(x)|dx = \left| \int_a^b f(x)dx \right|$$

So the additional term is just of the size of the error that one gets by storing the correct velue on the computer.
If one takes into account the process of summing (for not too large values of n) one can show that stability is preserved[29].

2.2.3 Data Errors

Data errors are like rounding errors, only may their size be larger and less systematic. They may arise when data are measured with instruments. Have you ever read temperatures from thermometers or measured running times at a track meeting by means of a stopwatch? Measurements with relative error less 10^{-4} will not be standard there, will they?
Values from measurements have always to be treated with much care. Very often it helps to have a lot of such measurements. By means of least squares methods[30] the influences of the errors can even out to some degree.
If derivatives of functions have to be computed from measurements of that function the direct application of finite differences can be disastrous. In such cases it is better to approximate the data with a smooth function of a function class the measured function is believed to belong to and then analytically differentiate that function[31]

[28] But ignore at this point that summation introduces rounding errors as well
[29] The factor `eps` in the last error term has to be multiplied by $\frac{n(n+1)}{2}$.
[30] See section **??**.
[31] See the sections on splines and on parameter estimation in ODEs.

2.2.4 Programming Errors

Today programming has become much easier than it has been some 10 to 20 years ago. This is due to the fact that large portions of numerical techniques do not have to be programmed from scratch but can be invokes by simple but very high level commands in MATLAB. Whereas the programming of a Gauss-type elimination algorithm to solve a linear system

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad x, b \in \mathbb{R}^n \tag{2.17}$$

for x with A and b given has always been very error prone, the MATLAB command

$$x = A \backslash b$$

to solve (2.17) has not to much parts where you could make mistakes, has it?
On the other hand today's programmes written in connection with a Bachelor- or Master- or even PhD-research-projects have become so involved and extensive, that it is worthwhile to know some techniques of Software-Engineering to allocate and eliminate programming errors. Please ask in the TUHH- Institute of Software Technology for courses on these techniques.

2.2.5 Stupid errors

Very annoying in scientific computing is what I would like to call "stupid errors". These arise mainly in those parts of your project where you feel that you are really expert. Since you have heard all about the subject you are sure to make no mistakes in that part.
But actually, sometimes you are tired, nervous or depressed due to some sad event not at all connected with your work. And hence you make a mistake in your very special personal area of expertise.
Since you know that you are expert in that field, you will not be able to find the erroneous formula, since you do not believe that the error could be here of all places. Thus you work for days to find the error at another place.
What to do? - My experience is, that you'd better go get a friend and explain him the whole - yes, the whole - programme. Though - after some 10 to 15 minutes - he will no more be able to follow you, your loud explanation will show you soon where the error is.

Chapter 3

Interpolation

3.1 Introduction

3.1.1 Recovery of functions from limited information

As you heard already in the introductory chapter every practical computation consists of a finite number of steps and delivers a finite number of numbers as a result. If one wants to compute functions by this process, one has to use functions which are determined by a finite number of data, like polynomials, e.g.:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0.$$

These functions are known and their value can be computed[1] in finite time as soon as their $n+1$ coefficients $a_0, a_1, \ldots, a_n$ are known.
Or one can try to represent the functions by a vector of values, like we did in the solution of the ordinary differential equation in Example 1 of Subsection 2.2.1.
Only a few years before I began my studies of mathematics in 1968 the latter was the case for most functions we are used now to have them available on our pocket calculator : Exponential functions, logarithms, sines, cosines, hyperbolic sines and even more complicated ones. Quite a lot of my costudents owned a copy of the "Milton-Abramowitz", [**?**], a rather thick book where long vectors of function values for all these functions where collected in "function tables".
If one wanted to know a value of, e.g., the Airy function at some argument value $\hat{x} = 0.3023$, then one would find values for $x = 0$ up to $x = 0.5$ in steps of 0.01. Near $\hat{x}$ one would find the values

x	**Ai(x)**
0.28	0.28372586
0.29	0.28126209
0.30	0.27880648
0.31	0.27635923
0.32	0.27392055
0.33	0.27149064

[1] By use of Horner's scheme the computation will be done with n additions and n multiplications. If you don't know this, have a look at PPP

and then one had to "read between the lines of this table".
This was done by constructing a function which reproduced the function values for those arguments closest to $\hat{x}$ and then to evaluate this function at $\hat{x}$.
In most cases one would use polynomials of low degree. The simple reason for using polynomials is that - as we will see - this procedure always works and the polynomials can be evaluated.
Today we have programmes which supply us with precise approximations of all the trigonometric functions, the exponential function, the logarithm and quite some other functions, which you even don't know yet - like the above cited Airy function. Thus you can use as well (linear) combinations of all of these to construct substitutes for a new function that you might have found in your world and from which most probably you would extract information - in a first attempt - by measuring function values.

3.1.2 General Interpolation Approach

Furthermore not only function values may be known of that function but other measurements instead like derivatives of different orders at different places or (weighted) integrals or linear or nonlinear combinations of all these. Hence we may assume that some functional[2] measurements are known and carry the information to fix a substitute for the measured function.
Usually one would use a parameter dependent function

$$\Phi(x; a_1, \ldots, a_n) : x \longrightarrow \mathbb{R}$$

as a model (we assume as much differentiability with respect to x as our computation needs) where one tries to determine the parameters $a_1, \ldots, a_n$ by requiring[3]

$$L_x^i(\Phi(x; a_1, \ldots, a_n)) = L_x^i(f(x)), \quad i = 1, \ldots, n.$$

Example 4: If $\phi(x; a_1, a_2) := a_1 e^{a_2 t}$ and one knows for the measured function f that

$$L^1(f) := f(0) = 1 \text{ and } L^2(f) := \int_0^1 f(x)dx = 2$$

then the equations

$$\begin{array}{rcccccl} L^1(\phi(\cdot; a_1, a_2)) & = & a_1 \cdot e^{a_2 \cdot 0} & = & a_1 & = & 1, \\ L^2(\phi(\cdot; a_1, a_2)) & = & \int_0^1 a_1 e^{a_2 \cdot x}\, dx & = & \frac{a_1}{a_2}\,(e^{a_2} - 1) & = & 2 \end{array}$$

would have to be used to determine a_1 and a_2.
Clearly $a_1 = 1$ is determined by the first of these equations, but for a_2 there are two solutions of the remaining equation

$$e^{a_2} = 1 + 2a_2,$$

as is easily seen[4] from Figure 3.1:

[2]Recall: A "functional" is a function from some set into the real or complex numbers. It is the mathematical model of a simple measurement. The input to the function is some thing to be measured and the output is a number: The measure of that thing.
Actually, it is common that one has more than one measure to identify a thing: You may have its height, its width, its depth, its weight, its eye colour, its date and place of birth, its name, or what else is in your identity card. - These are not all numbers? Well, although with an appropriate coding you could express these "measurements" as numbers, your right. In this course we prefer measurements which directly deliver real numbers as their values.

[3]The subscript x indicates in the following, that the functionals have to be applied with respect to the variable x.

[4]Central Theorem of practical Mathematics (Wodicka, 1984) : "A sketch in time saves nine."

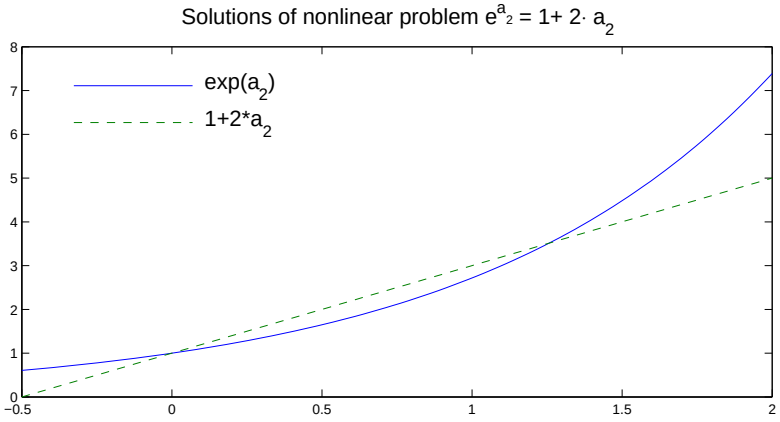

Figure 3.1: Nonlinear equation to determin a_2

This shows that it is by no means clear that the same number of "interpolation conditions" as unknown parameters will always lead to a single solution.
Notice that slight variation of the condition could equally well lead to a system of equations with no solution.
This possibly unexpected behavior is caused by the nonlinearity of the function Φ with respect to the parameter a_2. Very often the function Φ will be chosen to depend linearly on these parameters, and this is the case that we will deal with in some detail in the next section[5].

3.1.3 The linear interpolation problem

In linear interpolation problems[6]

- the function $\Phi(x; a_1, a_2, \ldots, a_n)$ is a linear combination of n functions $\varphi_1(x), \ldots, \varphi_n(x)$, where the $a_1, \ldots, a_n$ are the coefficients of the combination:

$$\Phi(x; a_1, a_2, \ldots, a_n) = a_1\varphi_1(x) + \ldots + a_n\varphi_n(x)$$

  and

- the interpolation conditions are given by linear functionals on the space of functions from span$\{\varphi_1, \ldots, \varphi_n, f\}$:

$$L^1(f) = w_1, \ldots, L^n(f) = w_n.$$

[5] You might ask, why one could have chosen such a stupid function like $\phi(x; a_1, a_2) := a_1 e^{a_2 t}$ which seems to lead directly to trouble. Actually, the form of the modeling function usually allows to express some extra (qualitative) information about the measuerd function. If e.g. one knows that the function describes the time evolution of the concentration of bacteria, then it is known, that an exponential ansatz would hit the spot.

[6] Attention: We do not think of interpolating with linear functions, but we think of functions which are linear in the determining parameters!

Then the interpolation conditions

$$L_x^m(\Phi(x;a_1,\ldots,a_n) = w_m, m = 1,\ldots,n \tag{3.1}$$

will simply form an $n \times n$-linear systems of equations to determine the parameters $a_1 \ldots, a_n$:

$$\begin{pmatrix} L^1(\varphi_1) & L^1(\varphi_2) & \cdots & L^1(\varphi_n) \\ L^2(\varphi_1) & L^2(\varphi_2) & \cdots & L^2(\varphi_n) \\ \vdots & & & \vdots \\ L^n(\varphi_1) & L^n(\varphi_2) & \cdots & L^n(\varphi_n) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} L^1(f) \\ L^2(f) \\ \vdots \\ L^n(f) \end{pmatrix}. \tag{3.2}$$

From Linear Algebra we recall:

A: If the matrix on the left side is nonsingular, then the system is always uniquely solvable. This is the desired case: For every set of data there is precisely one interpolating function.

B: If on the other hand the matrix is singular, then there are data sets for which there is no solution. Furthermore if there is a solution in this case, it is not unique. Hence, this case is rather undesirable.

Actually, the variable x must not be one dimensional nor must this $\Phi(x)$ be.

Example 5: To give a simple example with a multidimensional argument let the functions $\varphi_i : \mathbb{R}^2 \to \mathbb{R}$ be given by

$$\varphi_1(x,y) = 1, \quad \varphi_2(x,y) = x, \quad \varphi_3(x,y) = y, \quad \varphi_4(x,y) = xy,$$

and let interpolation conditions be given by

$$L^1(f) := f(0,0) = 1, \quad L^2(f) := f(1,0) = 0, \quad L^3(f) := f(0,1) = 0, \quad L^4(f) := f(1,1) = 1.$$

Then the system (3.2) reads

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} a = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

with the unique solution $a = [1, -1, -1, 2]^T$, such that the interpolating function is

$$\mathrm{Ipo}(x,y) = 1 - x - y + 2xy.$$

The graph is depicted in Figure 3.2.

If in the system (3.2) one chooses the first unit-vector as right-hand side

$$\begin{pmatrix} L^1(\varphi_1) & L^1(\varphi_2) & \cdots & L^1(\varphi_n) \\ L^2(\varphi_1) & L^2(\varphi_2) & \cdots & L^2(\varphi_n) \\ \vdots & & & \vdots \\ L^n(\varphi_1) & L^n(\varphi_2) & \cdots & L^n(\varphi_n) \end{pmatrix} \begin{pmatrix} a_1^1 \\ a_2^1 \\ \vdots \\ a_n^1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{3.3}$$

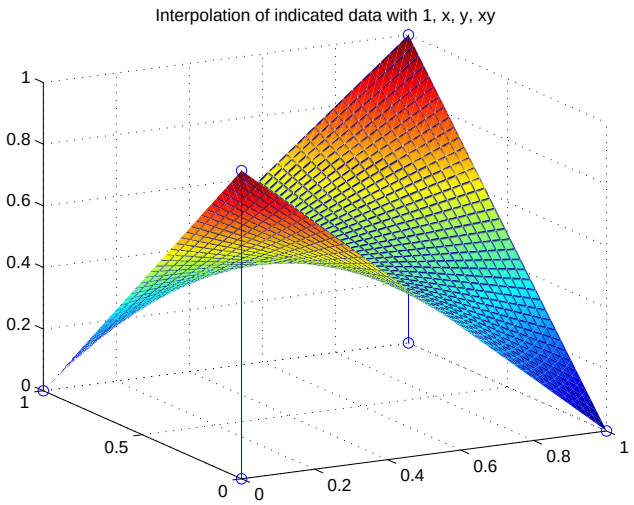

Figure 3.2: 2D-Interpolation

then the function

$$l_1(x) := \sum_{k=1}^{n} a_k^1 \, \varphi_k \tag{3.4}$$

will satisfy:

$$L^1(l_1) = 1, L^k(l_1) = 0, k = 2, \ldots, n.$$

Similarly, if we choose the mth unit vector as right hand side and build l_m in analogy to (3.4), then generally

$$L^k(l_m) = \delta_{km} := \begin{cases} 1 & \text{if } m = k, \\ 0 & \text{else.} \end{cases}.$$

Clearly, with the functions $l_1, \ldots, l_n$ the solution $Ipo[f](x) \in \text{span}\{\varphi_1, \ldots, \varphi_n\}$ fulfilling (3.1) may be written as[7]

$$Ipo[f](x) = \sum_{k=1}^{n} L^k(f) \cdot l_k(x). \tag{3.5}$$

The elements $l_1, \ldots, l_n$ and the functionals $L^1, \ldots, L^n$ are said to form **biorthogonal sets**.
For the case of polynomial interpolation of function values, the l_i-functions are usually called "Lagrangian elements". We will adopt this name for the case that the functionals are simple evaluations at an argument[8].

[7] Most probably you will have to spend some minutes to really understand the following formula (3.5). It will pay off later to invest that time. It may help you to understand if you concretize (3.5)for the situation of the Examples 5 and 6.

[8] cf. Section 3.1.5

Example 6: With the setting of the last example the Lagrangian elements are given by

$$l_1(x,y) = 1 - x - y + xy, \quad l_2(x,y) = x - xy, \quad l_3(x,y) = y - xy, \text{ and } l_4(x,y) = xy.$$

$l_1(x,y)$, $l_2(x,y)$ and $l_4(x,y)$ are shown in the subsequent figures.

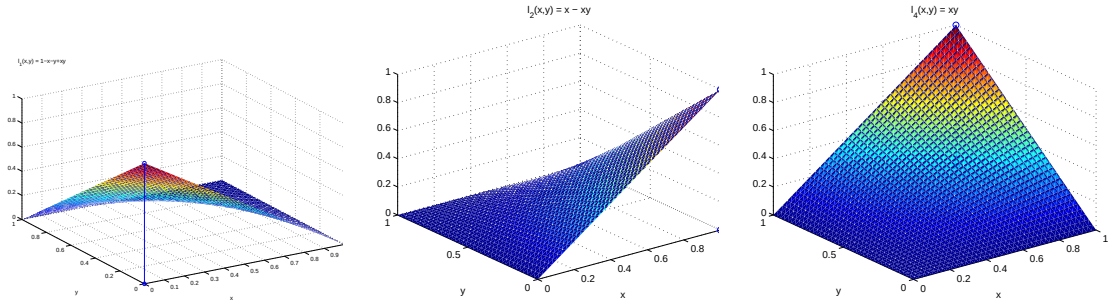

Figure 3.4: $l_2(x,y)$

3.1.4 Some common 1D-basis-functions

Though the first example of a linear interpolation problem has been with functions of two independent variables, we are mostly interested in interpolating real functions of one real variable in this course.

Some of the most often used interpolation systems are:

Polynomial Interpolation

$$\varphi_k(x) := x^{k-1}, k = 1, \ldots, n$$

Form of interpolating function:

$$p(x) = a_0 + a_1 x^1 + a_2 x^2 + \ldots + a_{n-1} x^{n-1}.$$

Trigonometric Interpolation

Used for interpolating functions with period $T \in \mathbb{R}, T > 0$:

$$\varphi_1(x) = \frac{1}{2}, \quad \varphi_{2k}(x) = \cos(k\frac{2\pi}{T}x), \quad \varphi_{2k+1}(x) = \sin(k\frac{2\pi}{T}x), \quad k = 1, \ldots m. (n = 2m+1)$$

Form of interpolating function[9]:

$$t(x) = \frac{a_0}{2} + \sum_{k=1}^{m} \left(a_k \cos(k\frac{2\pi}{T}x) + b_k \sin(k\frac{2\pi}{T}x) \right)$$

[9]You may be bemused by the factor 1/2 of a_0 or equivalently the choice of $\varphi_1(x) = \frac{1}{2}$. Couldn't one more simply write a_0 instead, which would mean to chose $\varphi_1(x) = 1$? - The reason for not doing so is the following. If one uses the given form, all the a_k coefficients will be calculable later by the same algorithm. Otherwise one would have to introduce an extra factor 0.5 for a_0 in the latter calculations.

Exponential Interpolation

With real values $\lambda_1 < \lambda_2 < \cdots < \lambda_n$ let

$$\varphi_k(x) := e^{\lambda_k t}, \quad k = 1, \ldots, n.$$

Form of interpolating function:

$$E(x) = a_1 e^{\lambda_1 x} + a_2 e^{\lambda_2 x} + \cdots + a_n e^{\lambda_n x}.$$

Often applied when modeling decay.

3.1.5 Standard Interpolation conditions

Standard interpolation conditions are as follows:

"Lagrangian Interpolation"

With real values $x_1 < x_2 < \cdots < x_n$ the functionals are given by

$$L^k(f) := f(x_k), \quad k = 1, \ldots, n.$$

Thus the interpolating function is required to have the same function values at $x_1, \ldots, x_n$ as f has or as being prescribed by a list of values.

If the interpolation-data are values at certain points, the points are called "interpolation points or nodes".

"Osculatory Interpolation"

With real values $x_1 < x_2 < \cdots < x_n$ there are $2n$ functionals given by

$$L_{2k-1}(f) := f(x_k), L_{2k}(f) := f'(x_k), \quad k = 1, \ldots, n.$$

The interpolating function is hence urged to touche the function at the nodes. The osculatory interpolation of $\sin(x)$ at the nodes $x_1 = 0$ and $x_2 = \pi$ is shown in Figure 3.5.

"Hermite Interpolation"

Hermite Interpolation is a generalization of Lagrangian and Osculatory Interpolation in that for each x_k from the set of points $x_1 < x_2 < \cdots < x_n$ an individual integer $d(k) \geq 1$ is specified (in Lagrangian interpolation $d(k) = 1$ in osculatory Interpolation $d(k) = 2$) which gives the number of successive derivatives to which f and the interpolating function agree at x_k.

$n = 3$ and $x_k = k, k = 1, 2, 3$ and $d(1) = 2, d(2) = 1, d(3) = 3$ would define the interpolation functionals to be:

$$L^1(f) = f(1), L^2(f) = f'(1), L^3(f) = f(2), L^4(f) = f(3), L^5(f) = f'(3), L^6(f) = f''(3).$$

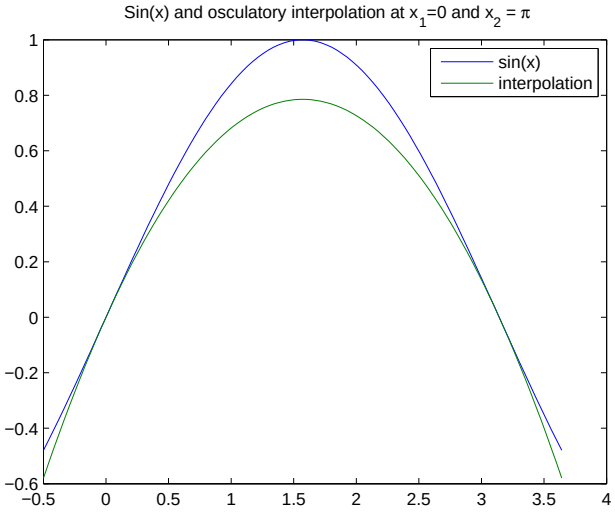

Figure 3.5: Osculatory Interpolation

Example 7: Taylor-Interpolation: The n-th order Taylor-expansion for $f(x) \in C^n(\mathbb{R})$ at $x_0 \in \mathbb{R}$ which you[10] remember to be

$$T(x) = \sum_{k=0}^{n} f^{(k)}(x_0) \frac{(x - x_0)^k}{k!}$$

is a polynomial Hermite-Interpolation, with one single node x_0 and the interpolation order $d(1) = n + 1$. You can easily check[11], that the functions

$$\frac{(x - x_0)^k}{k!}, k = 0, \ldots, n$$

and the interpolation functionals

$$L^k(f) = f^{(k)}(x_0), k = 0, \ldots, n$$

are biorthogonal.

"Hermite-Birkhoff-Interpolation"

If in Hermite-interpolation data one introduces gaps, meaning that at least for one of the nodes one or several derivatives between the zero-th an the highest occurring one do not belong to the interpolation functionals, this is called Hermite-Birkhoff interpolation.

Example 8: Hermite-Birkhoff-Interpolation is complicated and actually used very rarely. As an example for the problems that can arise let us look at polynomial interpolation with the Hermite-Birkhoff-

[10] HOPEFULLY
[11] DO IT!

functionals

$$L^1(f) = f(0), L^2(f) = f'(1), L^3(f) = f(a), \text{ with } a \text{ close to } 2.$$

Then with

$$\varphi_1(x) = 1, \varphi_2(x) = x \text{ and } \varphi_3(x) = x^2$$

the matrix from (3.2) reads

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 1 & a & a^2 \end{pmatrix}$$

with

$$\det A = a(a-2),$$

which means that interpolation with $a = 2$ is impossible. If interpolation is done with $a \neq 0$, e.g. for the data

$$f(0) = 0, f'(1) = 1, f(a) = 0,$$

then the interpolation polynomial

$$p(x) = \frac{a}{a-2} x \left(1 - \frac{x}{a}\right)$$

will explode with $a \longrightarrow 2$, which is not a desirable behavior.

Example 9: A well behaved Hermit-Birkhoff-type polynomial interpolation is **"Abel Interpolation"** where - as in Taylor's expansion - values of subsequent derivative values are prescribes. What makes the interpolation different from Taylor's method is that each derivative may be specified at a new node. The outcome is a series similar to Taylor's:

$$A(x) = \sum_{k=0}^{n} f^{(k)}(x_k) \cdot a_k(x)$$

where $a^k \in \Pi_k$ fulfills

$$a_k^{(m)}(x_m) = \delta_{km}, \quad k, m \in \{0, \ldots, n\}.$$

Exercise 1: Continue the series

$$a_0 \equiv 1, a_1 = (x - x_0), a_2(x) = \int_{x_0}^{x} \int_{x_1}^{s_1} 1 \, ds_2 \, ds_1, a_3(x) = \int_{x_0}^{x} \int_{x_1}^{s_1} \int_{x_2}^{s_2} 1 ds_3 ds_2 ds_1 ....$$

3.1.6 Preview: Applications

Before going into mathematical details of polynomial interpolation we will give some ideas of further applications of interpolation. We do not delay these examples until we did the theory, because we feel, that it is useful for the user of mathematics to know what he can do with the knowledge to be acquired.

Table lookup

As already mentioned in Section 3.1 one often encounters the situation, that one knows a function at a finite number of x-values and would like to use this knowledge to find y-values at x-values not in our table.

Assume, for instance, that we have prepared the following table by measuring:

x_i	0	1	3	5	5.5	8	9	10
y_i	1	3	5	6	4	5	7	9

A plot of these data is shown in Figure 3.6

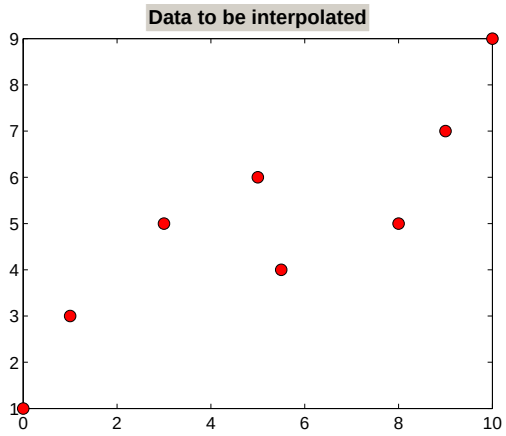

Figure 3.6: Plot of measured data

Since we have function values at our disposal a first idea would be to use polynomial interpolation with $\varphi_1(x) = 1, \varphi_2(x) = x, \ldots, \varphi_8(x) = x^7$, though we do not yet know, whether the matrix from (3.2) will be nonsingular[12].
The outcome is depicted in Figure 3.7.
After a brief look at this result we are a little disappointed: These large oscillation is most probably not what we were looking for, were we? Wouldn't it even be better to connect the points just by straight lines, i.a. do "piecewise linear interpolation".
If we do so we arrive at Figure 3.8). Actually, we eliminated nasty oscillation, but now we are not content with the sharp edges of the graph. We do not believe, that our measured function has such features.
What if we prescribed the derivatives at the points as well and try to use piecewise osculatory interpolation? If we prescribe derivatives, then coming from two side to a data point one would find the same inclination and hence no jumps of the derivative. - There are no derivatives given, is your objection? - You are right, but could't we use the difference quotient of adjacent data, e.g.?
If we do so, we arrive at Figure 3.9.
Ok, there are no more jumps of the derivatives now, but the change between the 4th and the 5th data point is somewhat too sudden. Isn't there something even smoother than this?

[12]Keep in mind: We should develop some general theorem, which helps to decide upon nonsingularity of this matrix, if possible.

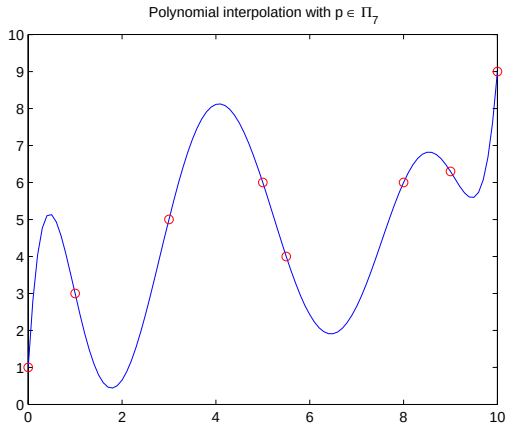

Figure 3.7: Polynomial Interpolation

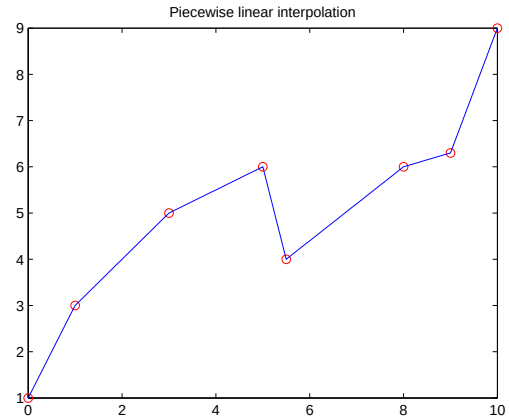

Figure 3.8: Piecewiese linear interpolation

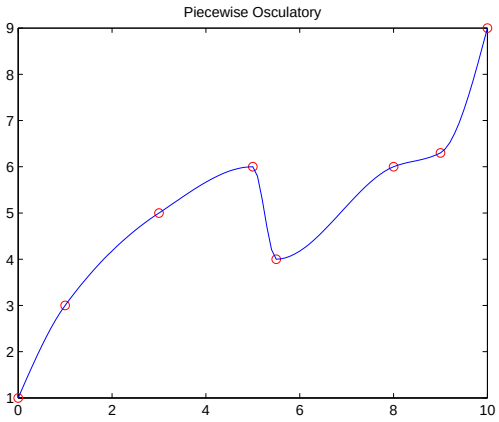

Figure 3.9: Piecewiese osculatory interpolation

Actually, in the sixties of the last century quite some mathematicians concentrated on developing "smoothest interpolants". For this they defined a mathematical abstraction of what design draughtsmen used lang time before: Flexible wooden rods where fastened with clamps on the drawing table such that they passed through given points. From physical considerations it was known that these rods would minimize their total curvature while reproducing given data. If we use the corresponding mathematical functions, called **"splines"** we will arrive at Figure 3.10.

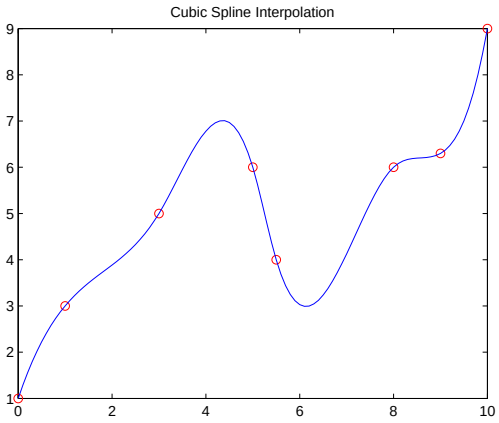

Figure 3.10: Cubic spline interpolation

Still not content? - Well in this case you would have to enter special courses on "Interpolation and Approximation" because we will not go deeper into the matter than this[13].

[13] See [?], [?]

Differentiation formulae

Using the definition of the derivative as a limit of simple difference quotients we arrived at the approximation

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h}. \tag{3.6}$$

From the first order approximation quality[14]

$$Trunc(h) = f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h} = O(h)$$

together with the estimate

$$Round(h) = \texttt{eps} \cdot O\left(\frac{1}{h}\right)$$

we derived by equating these errors the rule of thumb

$$h_{opt} \approx \sqrt{\texttt{eps}}.$$

for an optimal differentiation step width.
If we had a formula with higher than first order of approximation, say $Trunc(h) = O(h^2)$ then from

$$O(h^2) = \texttt{eps} \cdot O\left(\frac{1}{h}\right)$$

we would get

$$h_{opt} \approx h^{1/3}.$$

Best results would be found for a larger h where the rounding error is smaller and hence the overall error will be smaller as well.
A road to such improved differentiation formulae is opened by interpreting the finite difference as the result of differentiating a substitute for f gained from the data $f(x_0)$ and $f(x_0+h)$. Indeed the linear interpolation of f with these data is

$$Ipo(x) = f(x_0) + \frac{f(x_0+h)-f(x_0)}{h} \cdot (x-x_0)$$

and the first derivative of this is exactly the divided difference (3.6).
The idea to better formulae is to improve the interpolation and differentiate the improved interpolation. If, e.g. one interpolates at the nodes x_0-h, x_0 and x_0+h then the interpolation reads

$$Ipo(t) = \frac{(t-x_0)(t-x_0-h)}{2h^2} f(x_0-h) + \frac{h^2-(t-x_0)^2}{h^2} f(x_0) + \frac{(t-x_0+h)(t-x_0)}{2h^2} f(x_0+h).$$

Taking the derivative at x_0 results in

$$f'(x_0) \approx Ipo'(x_0) = \frac{f(x_0+h)-f(x_0-h)}{2h}$$

[14] If you do not know the Landau symbols $O(h^n)$ and $o(h^n)$ you should read the article http://en.wikipedia.org/wiki/Big_O_notation or (shorter but in German) http://de.wikipedia.org/wiki/Landau-Symbole.

and - using Taylor's expansion for $f \in C^k, k \geq 3$ - one verifies that

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + O(h^2).$$

Of course one could as well differentiate at $x_0 - h$ to get a onesided formula of higher order:

$$f'(x_0-h) \approx Ipo'(x_0-h) = \frac{-3f(x_0-h) + 4f(x_0) - f(x_0+h)}{2h}.$$

Furthermore the interpolation can be used to find a formula for the second derivative:

$$f''(x_0) \approx Ipo''(x_0) = \frac{f(x_0-h) - 2f(x_0) + f(x_0+h)}{h^2}.$$

By use of Taylor's expansion one can show, that the letter two formulae both are of second order.
Exercise 2: DO IT!
Higher order formulae would need more interpolation data.

Integration formulae

The above principle to approximate a functional result as applied to a function f by interpolating this function and then to apply the functional to this interpolation is used more often.
Quadrature formulae, i.e. formulae for the numerical integration of a function are generally derived by integrating a functions interpolation.
The above "Midpoint Rule" (2.13) or (2.14), respectively, can be derived by interpolating the function by a constant function at the midpoint of the integration intervall

$$f(x) \approx f(\frac{a+b}{2}) \tag{3.7}$$

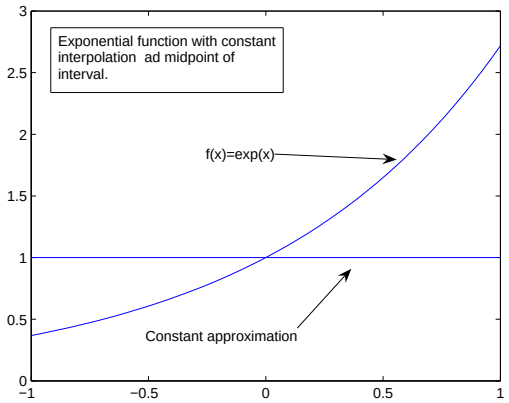

Figure 3.11: Constant interpolation

and then integrating

$$\int_a^b f(x)\,dx \approx \int_a^b f(\frac{a+b}{2})dx = (b-a)f(\frac{a+b}{2}).$$

Alternatively, the midpoint-rule can be obtained by integrating the Taylor-approximation of first order

$$f(x) \approx f(\frac{a+b}{2}) + f'(\frac{a+b}{2})(x - \frac{a+b}{2}). \tag{3.8}$$

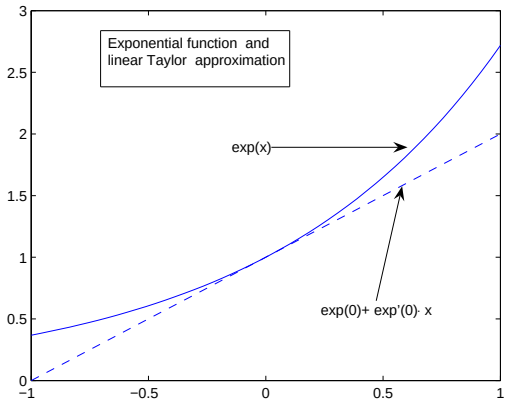

Figure 3.12: Linear Taylor Approximation

The constant parts of the integration are the same and the linear part cancels out due to point symmetry. Though the linear function approximation (3.8) [See Figure (3.12)] is obviously a better approximation to f than the constant (3.7) [See Figure (3.11)], the approximations of the integral are the same. Thus the constant approximation is as good as the linear one what concerns approximation of the integral. In a certain sense the constant approximation is hence better than it should be. Notice that this enjoyable behavior is due to the special choice of the midpoint as evaluation point. We will see this idea generalized under "Gaussian integration".

3.2 "Lagrangean" Polynomial Interpolation

Let us start with the polynomial interpolation of simple function values at real points $x_1 < x_2 < \cdots < x_n$. I.e. let us try to find a polynomial $p(x) = a_0 + a_1 t + a_2 t^2 + \ldots + a_{n-1} t^{n-1}$ such that $p(x_i) = f(x_i), i = 1, \ldots, n$.
Then the matrix (3.2) reads

$$\left(L^i \varphi_j\right) = \begin{pmatrix} 1 & t_1 & \cdots & t_1^n \\ \vdots & & & \\ 1 & t_{n+1} & \cdots & t_{n+1}^n \end{pmatrix}. \tag{3.9}$$

As is well known from Linear Algebra for this VanderMonde-Matrix one has

$$\det\ \left(L^i\varphi_j\right) = \det\ \begin{pmatrix} 1 & t_1 & \cdots & t_1^n \\ \vdots & & & \\ 1 & t_{n+1} & \cdots & t_{n+1}^n \end{pmatrix} = \prod_{i=1}^{n} \prod_{j=i+1}^{n+1} (t_j - t_i)$$

such that the determinant is nonzero - and the interpolation problem is uniquely solvable for all data $f(t_i), i = 1, \ldots, n$ - iff the nodes x_i are pairwise different.

Actually one does not prove the nonsingularity of the matrix (3.9 via the VanderMonde-determinant, but there are simpler ways of reasoning, which deduce nonsingularity by constructing polynomials which are biorthogonal to the point functionals or by showing that only the zero-polynomial can deliver all vanishing functional values.
This will be dealt with in the next section.

3.2.1 Different forms of the interpolating polynomial

Lagrangian form

For the $n \geq 1$ different real nodes

$$x_1 < \ldots < x_n$$

we want to find for each $i \in \{1, \ldots, n\}$ the element l_i from $\Pi_{n-1} := \text{span}\{1, x, \ldots, x^{n-1}\}$ which fulfills

$$l_i(x_k) = 0 \text{ for } k \in \{1, \ldots, n\} \setminus \{i\} \tag{3.10}$$

and

$$l_i(x_i) = 1. \tag{3.11}$$

The equation (3.10) requires that a polynomial of degree $n-1$ has $n-1$ different zeros. Obviously

$$\omega_i(x) = (x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)$$

is such a polynomial. Up to constant factors $\omega_i(x)$ is the unique polynomial that solves (3.10). To fulfill (3.11) to, we just have to divide the polynomial by it's value at $x = x_i$.

$$l_i(x) = \frac{\omega_i(x)}{\omega_i(x_i)} = \prod_{\substack{k=1 \\ k\neq}}^{n} \frac{x - x_k}{x_i - x_k}. \tag{3.12}$$

From (3.5) we see that

$$Ipo(x) = \sum_{i=1}^{n} f(x_i) l_i(x) = \sum_{i=1}^{n} f(x_i) \prod_{\substack{k=1 \\ k\neq i}}^{n} \frac{x - x_k}{x_i - x_k} \tag{3.13}$$

is one form of the interpolating polynomial. It is called "Lagrange's Form of the Interpolation Polynomial".
Notice again that being able to write down an interpolating polynomial, which solves the linear System

$$p(x_i) = f(x_i), i = 1 \ldots, n$$

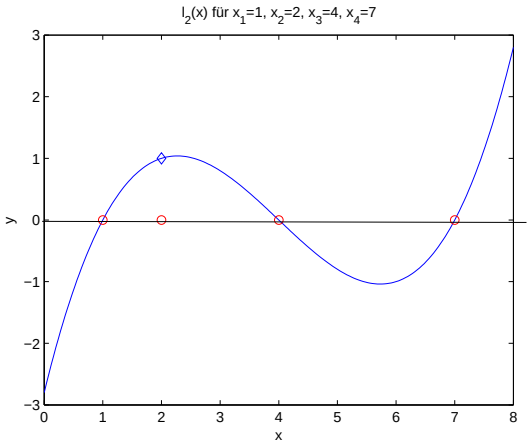

Figure 3.13: $l_2(x)$ for $x_1 = 1, x_2 = 2, x_3 = 4, x_4 = 7$

independently of the data $f(x_i)$ shows, that the system (3.9) is always solvable.
Since for mappings $M : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ defined by the multiplication of a vector with a square matrix from $\mathbb{R}^{n\times n}$ surjectivity, injectivity and bijectivity are all equivalent, the above shown surjectivity proves at the same time injectivity, such that there is a unique interpolation polynomial for every set of date. I.e., the polynomial (3.13) is not only "An interpolating polynomial" but it is "THE ONE AND ONLY INTERPOLATION POLYNOMIAL".
Another proof of the uniqueness of the interpolating polynomial is as follows: Assume that for some set of data, there are two polynomials p_1 and p_2 of degree at most $n-1$ which satisfy

$$p_1(x_i) = f_i = p_2(x_i), \quad i = 1, \ldots, n.$$

Then the polynomial $P(x) := p_1(x) - p_2(x)$ of degree at most $n-1$ has the n different zeros $x_1 < \cdots < x_n$. By the Fundamental Theorem of Algebra every polynomial of exact degree m has exactly m complex zeroes, if these are counted by multiplicity. Thus if p would not be the function equal to zero everywhere, it could posses at most $n-1$ zeros. Since it has n zeros by construction one has $P \equiv 0$.

Newton's form of the interpolating polynomial I

Lagrange's form of the interpolating polynomial is of course easily written down as a formula. But if one wants to evaluate the interpolating polynomial at some concrete x-value it takes quite some operations. If a polynomial is given in its canonical form

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$$

then an optimal evaluation scheme is Horner's Scheme.
This is a simple consequence of the following way to write $p(x)$:

$$p(x) = \underbrace{((\cdots(}_{n-1 \text{ braces}} a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \cdot x + \ldots\ldots + a_1) \cdot x + a_0$$

and then to evaluate it beginning at a_n:

```
p=a_n;
for k=1:n
p=p*x+ a_{n-k} end
```

As is readily seen from these statements Horner's scheme needs n multiplications and $n-1$ additions to evaluate a polynomial of degree n. (It can be shown, that this is best possible for an algorithm which can evaluate general polinomials[15]).

Newton's form of the interpolation polynomial[16] will be able to evaluate the interpolation polynomial with the same optimal number of algebraic operations.
The idea is to build the polynomial in a recursive way. Assume that $p_{n-1} \in \Pi_{n-1}$ interpolates the data

x_1	x_2	$\dots$	x_n
y_1	y_2	$\dots$	y_n

Then we aim at writing $p_n \in \Pi_n$, which interpolates the augmented data

x_1	x_2	$\dots$	x_n	x_{n+1}
y_1	y_2	$\dots$	y_n	y_{n+1}

as

$$p_n(x) = p_{n-1}(x) + a_n \cdot (x - x_1)(x - x_2) \cdots (x - x_n). \tag{3.14}$$

with a constant a_n to be determined.
Since $\omega_n(x) := x - x_1)(x - x_2) \cdots (x - x_n)$ is of degree n, so is p_n. Since the second term vanishes at $x_1, \dots, x_n$ one has

$$p_n(x_i) = p_{n-1}(x_i) + a_n \cdot \omega_n(x_i) = y_i + 0, \text{ for } i = 1, \dots, n.$$

To have

$$p_n(x_{n+1}) = y_{n+1}$$

as well, one finds from (3.14) that

$$y_{n+1} = p_n(x_{n+1}) = p_{n-1}(x_{n+1}) + a_n \omega_n(x_{n+1}).$$

Since $\omega_n(x_{n+1}) \neq 0$ this is easily and uniquely solved by :

$$a_n = \frac{y_{n+1} - p_{n-1}(x_{n+1})}{\omega_n(x_{n+1})}. \tag{3.15}$$

From this inductively one concludes that

$$p_n(x) = a_0 + a_1(x - x_1) + a_2(x - x_1) \cdot (x - x_2) + a_3 ..... + a_n \cdot (x - x_1)(x - x_2) \cdots (x - x_n) \tag{3.16}$$

and from this we draw two conclusions:

[15] Can you give an argument which proves this statement? - Notice that special polynomials can be evaluated with fewer operations of course. How many operations would you need to evaluate $m_n(x) := x^n$, e.g.?

[16] Please notice, that we are only constructing a different FORM of the unique interpolation polynomial. It is not a different interpolation!

Evaluation The evaluation of $p_n(x)$ from (3.16 can be done in a Horner-like way

```
p=a_n;
for k=1:n
p=p· (x-x_{n+1-k}) + a_{n-k}
end
```

Coefficient of highest x-power. a_n is the coefficient of x^n in p_n.

The latter observation allows us to compute the a_n's recursively. The basis for this recursion is

Aitken's Lemma; Nevill-Aitken Scheme

Theorem 1: [Aitken's Lemma]
If a set of interpolation data

x_1	x_2	$\dots$	x_n	x_{n+1}
y_1	y_2	$\dots$	y_n	y_{n+1}

is given and

$$p[x_1, \dots, x_{n-1}](x) \text{ and } p[x_2, \dots, x_n](x)$$

denote the polynomials from Π_{n-2} that interpolate the first and the last $n-1$ data, respectively, then

$$p[x_1, \dots, x_n](x) = \frac{p[x_1, \dots, x_{n-1}](x)(x - x_n) - p[x_2, \dots, x_n](x)(x - x_1)}{x_1 - x_n}. \tag{3.17}$$

Proof: Simple verification. □

Remark 2: [Neville and Aitken's Interpolation Scheme]
If a set of interpolation data

x_1	x_2	$\dots$	x_n	x_{n+1}
y_1	y_2	$\dots$	y_n	y_{n+1}

is given then the interpolation polynomial

$$p[x_i, \dots, x_j](x), \text{ for } 1 \le i \le j \le n$$

can be recursively computed applying (3.17) in

Neville and Aitken's Scheme:

$$
\begin{array}{lllllllll}
x_1 & y_1 = p[x_1] & & & & & & & \\
 & & > & p[x_1, x_2] & & & & & \\
x_2 & y_2 = p[x_2] & & & > & p[x_1, x_2, x_3] & & & \\
 & & > & p[x_2, x_3] & & & \dots & > & p[x_1, \dots, x_{n-1}] \\
x_3 & y_3 = p[x_3] & & & > & p[x_2, x_3, x_4] & & & > \quad p[x_1, \dots . x_n] \\
\vdots & & & & & & \dots & > & p[x_2, \dots, x_n] \\
x_{n-1} & y_{n-1} = p[x_{n-1}] & & & > & \dots & & & \\
 & & > & p[x_{n-1}, x_n] & & & & & \\
x_n & y_n = p[x_n] & & & & & & &
\end{array}
$$

Newton's form of the interpolating polynomial II

If we let

$$[x_i, \dots, x_j] \text{ for } i \leq j$$

denote the coefficient of x^{j-i} in $p[x_i, \dots, x_j]$ then from (3.17) we may infer that

$$[x_1, \dots, x_n] = \frac{[x_1, \dots, x_{n-1}] - [x_2, \dots, x_n]}{x_1 - x_n}, \tag{3.18}$$

and from the Neville-Aitken scheme we see, that these entities may be computed recursively as well

$$
\begin{array}{lllllllll}
x_1 & y_1 = [x_1] & & & & & & & \\
 & & > & [x_1, x_2] & & & & & \\
x_2 & y_2 = [x_2] & & & > & [x_1, x_2, x_3] & & & \\
 & & > & [x_2, x_3] & & & \dots & > & [x_1, \dots, x_{n-1}] \\
x_3 & y_3 = [x_3] & & & > & [x_2, x_3, x_4] & & & > \quad [x_1, \dots . x_n] \\
\vdots & & & & & & \dots & > & [x_2, \dots, x_n] \\
x_{n-1} & y_{n-1} = [x_{n-1}] & & & > & \dots & & & \\
 & & > & [x_{n-1}, x_n] & & & & & \\
x_n & y_n = [x_n] & & & & & & &
\end{array}
$$

Furthermore, Newton's interpolation formula (3.15) may now be written as

$$p[x_1, \dots, x_{n+1}](x) = [x_0] + [x_1, x_2](x - x_1) + [x_1, x_2, x_4](x - x_1)(x - x_2) + \dots + [x_1, x_2, \dots, x_{n+1}](x - x_1) \cdots (x - x_n). \tag{3.19}$$

Remark 3: The numbers $[x_i, \dots, x_j]$ are named **divided differences** due to the formula (3.18)

Newton's Interpolation Formula in the Hermitian Case

In case of Hermitian data (cf. 3.1.5) one calculates the devided differences for the Newton form by writing down each datum in the first colums as often as there are data given at that point.

Example 10: For Taylors polynomial of degree two for $f(x) = \sin(x)$ with expansion point $x_1 = 0$ one writes down

$$\begin{array}{ll} 0 & 0 = [x_1] \\ 0 & 0 = [x_2] \\ 0 & 0 = [x_3] \end{array}$$

Now, for the calculation of $[x_1, x_2]$ and $[x_2, x_3]$ one would have to take the divided differences

$$[x_1, x_2] = \frac{[x_1] - [x_2]}{x_1 - x_2}$$

and

$$[x_2, x_3] = \frac{[x_2] - [x_3]}{x_2 - x_3}$$

when applying the above algorithm, which here would lead to zero divided by zero, which is problematic. But if we interpret $[x_1, x_1]$ as "divided differences with confluent arguments"

$$[x_1, x_1] = \lim_{h \to 0} \frac{f(x_1 + h) - f(x_1)}{x_1 + h - x_1}$$

then we can continue our finite difference scheme by putting $\frac{d}{dx} \sin(x)_{|x=0} = \cos(0) = 1$ into the second column

$$\begin{array}{lll} 0 = x_1 & 0 = [x_1] & \\ & > & [x_1, x_2] = 1 \\ 0 = x_2 & 0 = [x_2] & \\ & > & [x_1, x_2] = 1 \\ 0 = x_2 & 0 = [x_3] & \end{array}$$

Finally, since there are three x-values that flow together, the third column has to contain

$$[x_1, x_1, x_1] := \lim_{h \to 0} [x_1 - h, x_1, x_1 + h] = \frac{f''(0)}{2!} \tag{3.20}$$

$$\begin{array}{lll} 0 = x_1 & 0 = [x_1] & \\ & > & [x_1, x_2] = 1 \\ 0 = x_2 & 0 = [x_2] & > [x_1, x_2, x_3] = 0 \\ & > & [x_2, x_3] = 1 \\ 0 = x_3 & 0 = [x_3] & \end{array}$$

Using Newton's Formula (3.19) and $x_1 = x_2 = x_3 = 0$ we get the usual Taylor expansion

$$p(x) = x$$

of $\sin(x)$ up to the quadratic term.

Exercise 3: Prove formula (3.20).

Example 11: Let us compute the Hermite interpolation for $f(x) = \sin(x)$ with interpolation points $x_1 = x_2 = 0$ and $x_3 = x_4 = \pi$, i.e. let us prescribe the functions valus as well as the derivatives at the argument values 0 and π.
Then the scheme of divided differences reads

$$\begin{array}{llllll}
0 = x_1 & 0 = [x_1] & & & & \\
 & & > & [x_1, x_2] = 1 & & \\
0 = x_2 & 0 = [x_2] & & & > & [x_1, x_2, x_3] = \frac{0-1}{\pi-0} = -(\pi)^{-1} \\
 & & > & [x_2, x_3] = \frac{0-0}{\pi-0} = 0 & & \\
\pi = x_3 & 0 = [x_3] & & & > & [x_1, x_2, x_3] = \frac{-1-0}{\pi-0} = -(\pi)^{-1} \\
 & & > & [x_1, x_2] = -1 & & \\
\pi = x_2 & 0 = [x_3] & & & &
\end{array}$$

$$\begin{array}{lll}
 & & \\
 & & \\
 & & \\
 & > & [x_1, x_2, x_3, x_4] = 0 \\
 & & \\
 & & \\
 & &
\end{array}$$

Thus the Hermite interpolation polynomial reads

$$p(x) = 0 + (x-0) - \frac{1}{\pi}(x-0)(x-0) + 0 \cdot (x-0)(x-0)(x-\pi) = x(1 - \frac{x}{\pi}).$$

The interpolation polynomial together with $\sin(x)$ have already been shown in Figure 3.5 in Section 3.1.5.

3.2.2 Error estimation

Normally the interpolation polynomial differs from the original function.

Exercise 4: Determine under which conditions interpolation polynomials coalesce with the function being interpolated.

As we have seen in examples[17] errors can be of significant size.
Thus one usually wants to estimate or bound the errors.
We are going to explain some fundamentals on errors with polynomial interpolation.

Error of the Taylor-series

Theorem 2: [Taylor's Error Representation]
If $f \in C^n([a,b], \mathbb{R})$ and $x_0 \in [a,b]$ then

$$f(x) = \sum_{k=0}^{n-1} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \int_{x_0}^{x} \frac{(x-s)^{n-1}}{(n-1)!} f^{(n)}(s)\, ds. \tag{3.21}$$

for all $x \in [a,b]$.

[17] Each of the interpolates from the Figure 3.7 up to 3.10 could serve as a function delivering the data in Figure 3.6 and the others, respectively, as its interpolants.

Proof: The proof is by in induction. Obviously (3.21) is true for $n = 1$ since

$$f(x) - f(x_0) = \int_{x_0}^{x} f'(s)ds.$$

Now assume that (3.21) is true for some $n \geq 1$. Then the induction step follows by integration by parts from

$$\int_{x_0}^{x} \frac{(x-s)^{n-1}}{(n-1)!} f^{(n)}(s)ds = -\frac{(x-s)^n}{(n)!} f^{(n)}(s)\Big|_{x_0}^{x} + \int_{x_0}^{x} \frac{(x-s)^n}{(n)!} f^{(n+1)}(s)\, ds$$

As a corollary to the last theorem we have

Theorem 3: [Cauchy's Error Representation]
If $f \in C^n([a,b],\mathbb{R})$ and $x_0, x \in [a,b]$ $(x \neq x_0)$ then there is a ζ in the open interval between x_0 and x such that

$$f(x) = \sum_{k=0}^{n-1} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \frac{f^{(n)}(\zeta)}{n!}(x-x_0)^n. \tag{3.22}$$

Proof: Apply the Mean Value Theorem of Integration to Taylor's Error from the last theorem. □

Error of the polynomial Hermitian and Lagrangean interpolation

Now we are going to prove a Cauchy-type error expression for the polynomial Hermitian interpolation problem. Since Taylor-expansion is a special case of Hermite-interpolation this is another version for a proof of the last theorem. Included is as well the case of a Lagrangian polynomial interpolation.

Theorem 4: [Error of polynomial Hermite Interpolation]
For $f \in C^n([a,b],\mathbb{R})$ let $\mathrm{Ipo}[f]$ denote the interpolation polynomial of degree at most $n-1$, which fulfills

$$\left(\frac{d}{dx}\right)^k (\mathrm{Ipo}[f](x) - f(x))_{|x=x_i} = 0, \text{ for } k = 0, \ldots, m_i - 1$$

and $i = 1, \ldots, p$ with $a \leq x_1 < x_2 < \ddots < x_p \leq b$ and $\sum_{i=1}^{p} m_i = n$. Then for every $x \in [a,b]$ there is a $\zeta \in I(x_1, \ldots, x_p; x)$ such that

$$f(x) - \mathrm{Ipo}[f](x) = \frac{f^{(n)}(\zeta)}{n!} \cdot \prod_{i=1}^{p} (x - x_i)^{m_i}, \tag{3.23}$$

where $I(x_1, \ldots, x_p; x)$ is the smallest interval which contains $x_1, \ldots, x_p$ and x.
Proof: For $x \in \{x_1, \ldots, x_p\}$ formula (3.23) is trivial. Hence let $x \neq x_i, i = 1, \ldots, p$. Then choose $\alpha \in \mathbb{R}$ such that

$$R(t) = f(t) - \mathrm{Ipo}[f](t) - \alpha \cdot \prod_{i=1}^{p} (t - x_i)^{m_i} \tag{3.24}$$

is zero at x. Then $R(t)$ has the zeros $x_1, \ldots, x_k$ with the summed up multiplicities giving n. The additional zero x increases the number of zeros to $n+1$. Due to the mean value theorem the first derivative of R

has at least n zeros in $I(x_1, \ldots, x_p; x)$ possibly counted with multiplicities. Then the second derivative of R has $n-1$ zeros in the smallest interval containing these zeros of R' (which is of course a subinterval of $I(x_1, \ldots, x_p; x)$. Continuing this way we find that $R^{(n)}(\zeta) = 0$ for at least one $\zeta \in I(x_1, \ldots, x_p; x)$. Since the nth derivative of $\mathrm{Ipo}[f]$ vanishes one has

$$0 = R^{(n)}(\zeta) = f^{(n)}(\zeta) - \alpha \cdot n!.$$

Hence

$$\alpha = \frac{f^{(n)}(\zeta)}{n!}$$

and

$$0 = R(x) = f(x) - \mathrm{Ipo}[f](x) - \alpha \cdot \prod_{i=1}^{p} (x - x_i)^{m_i}$$

shows the desired result. □

Example 12: To bound the interpolation error in Example 11 for $x \in [0, \pi]$ we may write

$$f(x) - \mathrm{Ipo}[f](x) = \frac{f^{(4)}(\zeta_x)}{4!} x^2 (x - \pi)^2, \quad \zeta_x \in [0, \pi].$$

Bounding the absolute value of $f^{(4)}(\zeta_x) = \sin(\zeta_x)$ by 1, we infer that

$$|f(x) - \mathrm{Ipo}[f](x)| \leq \frac{x^2 (x - \pi)^2}{24} \leq \frac{\pi^4}{384} < 0.2537.$$

Comparing with the real maximal error, $f(\pi/2) - \mathrm{Ipo}[f](\pi/2) = 1 - \frac{\pi}{4} \approx 0.2146$ we see that the estimate is not too bad, though the approximation quality itselve could of course be improved.

3.2.3 Use of polynomial interpolation

As already seen in Figure 3.7 interpolating polynomials of higher degree tend to show undesirably large oscillations. This is reflected in the part $\prod_{i=1}^{p} (x - x_i)^{m_i}$ of the error expression (3.23). Figure 3.14 shows the large oscillations of this term for the Lagrangian interpolation with interpolation points $x_k = k, \quad k = 1, \ldots, 10$.

Thus - in order to approximate a given function better and better - one should not use polynomial interpolation with increasing polynomial degree and equidistributed nodes. Actually, one can get better results by using certain non equidistant distribution of the interpolation points. The Chebyshev-nodes

$$x_i = \cos\left(\frac{2i - 1}{2n} \pi\right), \quad i = 1, \ldots, n$$

e.g. minimize the maximum of

$$\prod_{i=1}^{p} |x - x_i|$$

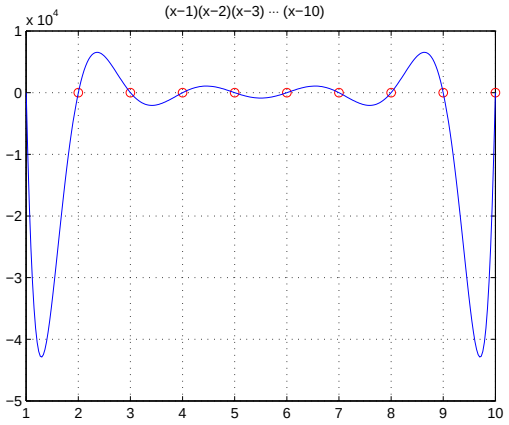

Figure 3.14: Graph of $\prod_{k=1}^{10}(x-k)$.

over $[-1,1]$. By linear transformation, one gets corresponding nodes

$$\tilde{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1,\ldots,n$$

for a general interval (a,b).
Figure 3.15 compares the polynomials $\omega(x)$ for the equidistributed points $x_k = k, k = 1,\ldots,10$ and the Chebyshev-Points for $(a,b) = (1,10)$ and $n = 10$.

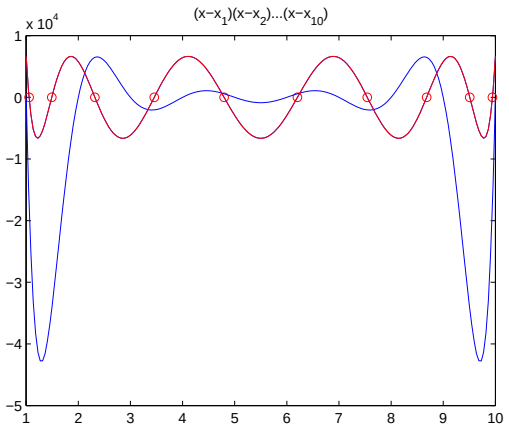

Figure 3.15: Graph of $\prod_{k=1}^{10}(x-\tilde{x}_k)$.

Actually, using Chebyshev nodes is better than using eqidistributed nodes, but

- 1. Sometimes the data come with equidistributed x-values

- 2. Rarely one is in the position to prescribe the x-values.

Then one would prefer what is known as spline-interpolation, what is the theme of the next section.

3.3 Spline Interpolation

Let

$$\Delta : a = x_0 < x_1 < \cdots < x_n = b \tag{3.25}$$

be a subdivision of the interval $[a, b]$.
Then $S(\Delta, p, q)$ with $p, q \in \mathbb{N}_0, 0 \leq q < p$ denotes the class of functions $s \in C^q[a, b]$, with

$$s_{|[x_i, x_{i+1}]} \in \Pi_p, \quad i = 0, \ldots, n-1;$$

i.a. the restriction of s to every interval $[x_i, x_{i+1}]$ is a polynomial of degree at most p.
$s \in S(\Delta, p, q)$ is called a polynomial spline of degree p, smoothness q and grid Δ. The junction points x_1 to x_{n-1} between the subintervals are called the **knots** of the spline.
Most used splines are those from $S(\Delta, 3, 2)$ (cubic spline) and from $S(\Delta, 3, 1)$ (cubic Hermite spines). Before we address these we discuss the simpler ones from $S(\Delta, 1, 0)$.

3.3.1 Piecewise linear interpolation

Interpolation Let $\Delta : a = x_0 < x_1 < \cdots < x_n = b$ be a partition of the interval $[a, b]$. Then

$$S(\Delta, 1, 0) = \left\{ s \in C[a, b] : s_{|[x_i, x_{i+1}]} \in \Pi_1, i = 0, \ldots, n-1 \right\}$$

is the set of continuous piecewise linear functions with breakpoints at the "inner points" of Δ.
On each interval the spline is a linear function, such that the overall number of parameters to be determined is $2n$. The continuity condition $s \in C^0$ produces the $n-1$ conditions

$$s(x_i-) = s(x_i+), i = 1, \ldots, n-1$$

at the knots of the spline, such that there may be imposed still $n+1$ interpolation conditions, e.g. Lagrangean conditions of the form

$$s(n_i) = y_i, i = 0, \ldots, n.$$

The points n_i are usually called the **nodes** of the spline.
It is an interesting task to investigate, how nodes an knots must be interlaced to guarantee unique solvability of the interpolation problem.
We will henceforth confine ourselves to the important case, where the nodes are the knots together with the boundary points $x_0 = a$ and $x_n = b$.
Actually, we only mentioned the general case to explain the difference between **nodes** and **knots.**
If now the values $y_0, \ldots, y_n \in \mathbb{R}$ are used to fix the values of s at the points of Δ, then - obviously - there is exactly one $s \in S(\Delta, 1, 0)$ fulfilling $s(x_i) = y_i, i = 0, \ldots, n$ and

$$s(x) = \frac{1}{x_i - x_{i-1}} \left(y_i(x - x_{i-1}) + y_{i-1}(x_i - x) \right), \quad x \in [x_{i-1}, x_i]. \tag{3.26}$$

Interpolation Error: If $y_i = f(x_i)$ for some $f \in C^2[a,b]$ then the error estimate (3.23) provides us with the result

$$f(x) - s(x) = \frac{1}{2}(x - x_{i-1})(x - x_i)f''(\zeta_i), \quad \text{for } x \text{ and } \zeta_i = \zeta_i(x) \text{ from } [x_{i-1}, x_i].$$

Hence

$$|f(x) - s(x)| = \frac{1}{8}(x_i - x_{i-1})^2 \cdot |f''(\zeta_i)|, \text{ for all } x \text{ and } \zeta_i = \zeta_i(x) \text{ from } [x_{i-1}, x_i].$$

Letting (see above)

$$|\Delta| := \max_{i=0,\ldots,n-1}(x_i - x_{i-1}) \text{ and } \|f\|_\infty := \max_{x\in[a,b]} |f(x)|$$

it follows that

$$\|f - s\|_\infty \leq \frac{1}{8}\|\Delta\|^2 \cdot \|f''\|_\infty. \tag{3.27}$$

Hence, if Δ_n is a sequence of partitions with $\lim_{n\to\infty} |\Delta_n| = 0$, then for $f \in C^2([a,b])$ the sequence $\{s_n\}_{n\in\mathbb{N}}$ converges uniformly to f on $[a,b]$.

Lagrangean Elements (Hat functions): For piecewise linear interpolation of the function values at the points of the partition the Lagrangian elements $\varphi_j(x)$ from $S(\Delta, 1, 0)$ with

$$\varphi_j(x_i) = \delta_{ij}, \quad i,j = 0,\ldots,n$$

are easily written up as

$$\varphi_i(x) := \begin{cases} (x - x_{i-1})/(x_i - x_{i-1}), & \text{for } x \in [x_{i-1}, x_i], \\ (x_{i+1} - x)/(x_{i+1} - x_i), & \text{for } x \in [x_i, x_{i+1}], \quad i = 1,\ldots,n-1 \\ 0, & \text{for } x \notin [x_{i-1}, x_{i+1}], \end{cases}$$

and

$$\varphi_0(x) = \begin{cases} (x - x_1)/(x_0 - x_1), & \text{for } x \in [x_0, x_1], \\ 0 & \text{sonst} \end{cases}$$

as well as

$$\varphi_n(x) = \begin{cases} (x - x_{n-1})/(x_n - x_{n-1}), & \text{for } x \in [x_{n-1}, x_n], \\ 0 & \text{sonst.} \end{cases}$$

These function are called **hat functions**, cf. Figure 3.16. They have a local support (= the set of arguments, where they are different from zero), and if the interpolation has to be calculated at some x only two of them have to be calculated, since all the other ones vanish there. This is an comforting behaviour, which is used in the Theory of Finite Elements with great success.

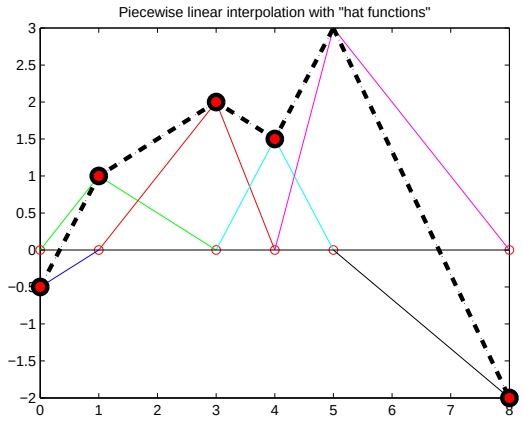

Figure 3.16: Piecewise linear interpolation

3.3.2 Cubic Spline Interpolation

While the theory of the $S(\delta, 1, 0)$-splines is simple the resulting interpolates are often not fully satisfying, since they lack in smoothness at the interpolation points.
$S(\Delta, 3, 2)$-splines by contrast are smooth up to the second derivatives but are not calculated that easily and not defined locally.
Within each subinterval $[x_i, x_{i+1}]$ a spline $s \in S(\Delta, 3, 2)$ is a polynomial of degree three, such that overall s is determined by $4n$ parameters, four in every interval. The smoothness-condition $s \in C^2$ imposes three conditions at every knot,

$$s^{(j)}(x_i - 0) = s^{(j)}(x_i + 0), \quad j = 0, 1, 2; i = 1, \ldots, n-1,$$

there remain $n + 3$ degrees of freedom to be determined.
As for the piecewise linear interpolation we only consider the case, where we interpolate values at the points from Δ, i.e. we consider the case "nodes=knots":

$$s(x_i) = y_i, \quad i = 0, \ldots, n.$$

But even though we included the endpoints of the interval into the set of nodes, there remain $n + 3 - (n + 1) = 2$ degrees of freedom to be determined.
These may be determined in several ways by so called **boundary conditions**.

Theorem 5: [Unique existence of interpolating cubic splines]
Given the partitioning Δ from (3.25) and the interpolation data $y_0, \ldots, y_n \in \mathbb{R}$ there is a unique spline $s \in S(\Delta, 3, 2)$ with

$$s(x_i) = y_i, \quad i = 0, \ldots, n$$

for each of the following five "boundary conditions":

(i) $s'(a) = y'_a, \quad s'(b) = y'_b, \qquad y'_a, y'_b \in \mathbb{R}$ prescribed,

(ii) $s''(a) = y''_a, \quad s''(b) = y''_b, \qquad y''_a, y''_b \in \mathbb{R}$ prescribed,

(iii) $s''(a) = s''(b) = 0$,

(iv) $s'(a) = s'(b), \quad s''(a) = s''(b)$,

(v) $s^{(3)}(x_1-) = s^{(3)}(x_1+), \quad s^{(3)}(x_{n-1}-) = s^{(3)}(x_{n-1}+)$,

Remark 4: The spline s can be seen as solution of the beam-equation

$$s^{(4)}(x) = 0$$

with point loads at the knots to fulfill the interpolation conditions.
The boundary conditions (i) to (iii) are boundary conditions that may be found in beam theory.
(i) corresponds to the case, where the beam is fixed at the end, prescribing the value (by the interpolation conditions at a and b)as well as the slope.
(ii) would specify the moments instead of the slopes.
(iii) corresponds to a simply supported beam, where the "natural boundary conditions" (iii) appear automatically.
(iv) is a condition for periodicity, which one would chose if functions with period $T = (b - a)$ are interpolated. (See section 3.4.5.)
(v) is the so called "not-a-knot-condition". This is chosen, if one does nor know anything about the boundary behavior of the function. It is applied when you use the normal MATLAB routine `spline`.

Remark 5: For the boundary conditions (i), (ii), (iv) the interpolating cubic spline is in the following sense the smoothest function, which interpolates the given data. Among all C^2-functions $y(x)$ which interpolate the data $y(x_i) = y_i, i = 0, \ldots, y_n$ and fulfill the boundary condition the cubic spline is the unique function, which minimizes the following mean-curvature

$$C(y) := \int_a^b \left(y''(x)\right)^2 dx.$$

The cubic spline with boundary condition (iii) is the minimizing interpolator of $C(y)$ without imposing any additional boundary condition.

Remark 6: The conditions (i) to (iii) can of course be "mixed", i.e, one could clamp the beam on the a-side by $s'(a) := y'(a)$ and have a natural boundary condition $s''(b) = 0$ on the other, and so forth.

Proof of the theorem (Idea): The proof is done by explicit construction. To this end one assumes to know the values

$$s''(x_i) = M_i, \quad i = 0, \ldots, n.$$

of the second derivatives at the knots, which are called "moments". Since $s''(x)$ is a piecewise continuous linear function, it is determined by the moments[18] $M_0 \ldots, M_n$.

[18] As will be explained after the proof that the function $s(x)$ can be interpreted as the deflexion line of a homogeneous beam, with point loads at the knots to induce the fulfillment of the interpolation conditions at he nodes. Notice, that the second derivative of the deflexion line is called "moment" in beam theory.

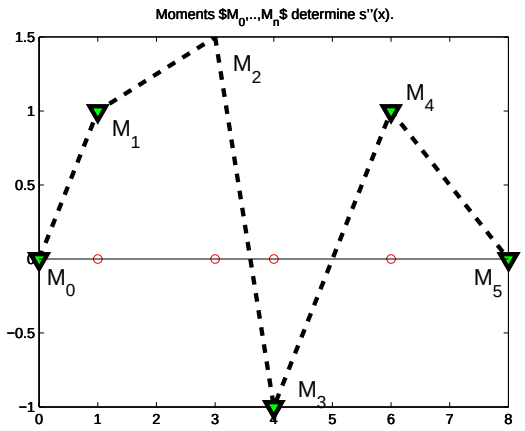

Figure 3.17: Piecewise linear interpolation

$$s''(x) = \frac{1}{h_{j+1}} \left(M_j(x_{j+1} - x) + M_{j+1}(x - x_j) \right), x \in [x_j, x_{j+1}], \quad h_{j+1} := x_{j+1} - x_j.$$

Integrating twice delivers

$$s'(x) = -M_j \frac{(x_{j+1} - x)^2}{2h_{j+1}} + M_{j+1} \frac{(x - x_j)^2}{2h_{j+1}} + \alpha_j \tag{3.28}$$

and

$$s(x) = M_j \frac{(x_{j+1} - x)^3}{6h_{j+1}} + M_{j+1} \frac{(x - x_j)^3}{6h_{j+1}} + \alpha_j(x - x_j) + \beta_j. \tag{3.29}$$

The constants of integration α_j and β_j can be determined from

$$s(x_j) = y_j \text{ an } s(x_{j+1}) = y_{j+1}$$

to be

$$\beta_j = y_j - M_j \frac{h_{j+1}^2}{6}$$

and

$$\alpha_j = \frac{y_{j+1} - y_j}{h_{j+1}} - \frac{h_{j+1}}{6}(M_{j+1} - M_j).$$

Inserting these constants into (3.29) gives the description of s solely by the moments.
To derive equations for the moments one makes use of the up to now neglected fact, that $s'(x)$ is continuous at the knots. Inserting α_j into(3.28) and using

$$s(x_j-) = s(x_j+), \quad \text{for } j = 1, \ldots, n-1$$

leads to the $n-1$ equations

$$\frac{h_j}{6} M_{j-1} + \frac{h_j + h_{j+1}}{3} M_j + \frac{h_{j+1}}{6} M_{j+1} = \frac{y_{j+1} - y_j}{h_{j+1}} - \frac{y_j - y_{j-1}}{h_j}, j = 1, \ldots, n-1 \tag{3.30}$$

for the $n+1$ unknowns $M_0, \ldots, M_n$.
The missing two equations result from the boundary conditions.

(i) $s'(a) = y'_a, \quad s'(b) = y'_b, \qquad y'_a, y'_b \in \mathbb{R}$ prescribed, lead to

$$\frac{h_1}{3}M_0 + \frac{h_1}{6}M_1 = \frac{y_1 - y_0}{h_1} - y'_0 \text{ and } \frac{h_n}{6}M_{n-1} + \frac{h_n}{3}M_n = y'_n - \frac{y_n - y_{n-1}}{h_n}.$$

(ii) $s''(a) = y''_a, \quad s''(b) = y''_b, \qquad y''_a, y''_b \in \mathbb{R}$ prescribed, lead to

$$M_0 = y''_a \text{ and } M_n = y''_b.$$

(iii) $s''(a) = s''(b) = 0$, lead to

$$M_0 = 0 \text{ and } M_n = 0.$$

(iv) $s'(a) = s'(b), \quad s''(a) = s''(b)$, lead to

$$M_0 = M_n \text{ and } \frac{h_n}{6}M_{n-1} + \frac{h_n + h_1}{3}M_0 + \frac{h_1}{6}M_1 = \frac{y_1 - y_0}{h_1} - \frac{y_0 - y_{n-1}}{h_n},$$

(v) $s^{(3)}(x_1-) = s^{(3)}(x_1+), \quad s^{(3)}(x_{n-1}-) = s^{(3)}(x_{n-1}+)$, lead to

$$-h_2M_0 + (h_2 + h_1)M_1 - h_1M_2 = 0 \text{ and } -h_nM_{n-2} + (h_{n-1} + h_n)M_{n-1} - h_{n-1}M_n = 0.$$

Together with the equations (3.30) each of these sets of extra conditions leads to a strictly diagonal dominant set of equations. The subsequent Lemma will show, that the matrices of such equations are non singular, which concludes the proof. □

Lemma 1: [Strictly diagonally dominant matrices]
A real (or complex) $n \times n$-matrix $A = (a_{ij})_{i,j=1}^n$ is called strictly (row-) diagonally dominant, iff

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, \ldots, n.$$

i.e. if for every row the sum of absolute values of the off-diagonal-elements is strictly less then the absolute value of the corresponding diagonal element.
Such matrices are non singular.

Proof: Assume that A is singular. Then there is a non vanishing vector x such that

$$Ax = 0.$$

Let $i \in \{1, \ldots, n\}$ be determined such that $|x_i| = \max\{|x_j| : j = 1, \ldots, n\} > 0$ and consider the ith equation of $Ax = 0$:

$$a_{i1}x_1 + \cdots + a_{i,i-1}x_{i-1} + a_{ii}x_i + a_{i,i+1}x_{i+1} + \cdots + a_{i,n}x_n = 0.$$

Solving for $a_{ii}x_i$ gives

$$a_{ii}x_i = -\left(a_{i1}x_1 + \cdots + a_{i,i-1}x_{i-1} + 0 + a_{i,i+1}x_{i+1} + \cdots + a_{i,n}x_n\right).$$

Deviding by x_i and going over to absolute values leads to

$$|a_{ii}| \leq |a_{i1}\frac{x_1}{x_i}| + \cdots + |a_{i,i-1}\frac{x_{i-1}}{x_i}| + |a_{i,i+1}\frac{x_{i+1}}{x_i}| + \cdots + |a_{i,n}\frac{x_n}{x-i}| \leq \sum_{k\neq i} |a_{i,k}|$$

contradicting

$$|a_{ii}| > \sum_{k\neq i} |a_{i,k}| \quad \text{for all } i = q1, \ldots, n.$$

This concludes the proof. □

Convergence of Cubic Spline Interpolants

What concerns convergence results for sequences of interpolating cubic splines similar to the estimation (3.27) for piecewise linear splines, there are lots and lots of them. Since the cubic splines are not locally defines as the linear ones are, all the proofs are rather involved. Hence we just cite for the cubic interpolation with boundary conditions of the sort (i) the following theorem, which was first proved by Hall an Meyer in J. Appr. Theory 16 in 1976:

Theorem 6: [Convergence of cubic spline interpolants]
Let $f \in C^4[a,b]$ and let $s \in S(\Delta, 3, 2)$ fulfill

$$s(x_i) = f(x_i), i = 1, \ldots, n; \quad s'(a) = f'(a),\ s'(b) = f'(b).$$

Then

$$\begin{aligned} \|f - s\|_\infty &\leq \tfrac{5}{384}\,|\Delta|^4\,\|f^{(4)}\|_\infty, \\ \|f' - s'\|_\infty &\leq \tfrac{1}{24}\,|\Delta|^3\,\|f^{(4)}\|_\infty, \\ \|f'' - s''\|_\infty &\leq \tfrac{3}{8}\,|\Delta|^2\,\|f^{(4)}\|_\infty. \end{aligned}$$

Remark 7: The last inequalities demonstrate, that cubic splines may be used to derive from a discrete set of function values not only approximate values of the function in the full interval but as well values of the derivatives.
We will need this in Section **??**.

3.4 Further Issues on Interpolation

3.4.1 Trigonometric Interpolation

Most often in trigonometric interpolation equidistributed Lagrangean data

$$((t_i), f_i) := \left(\frac{i}{n}T, f(\frac{i}{n}T)\right) i = 0, 1, \ldots, n-1$$

in an interval of periodicity $[0, T]$ are interpolated by trigonometric polynomials of the kind discussed in subsection 3.1.4.
The formulae are more easily derived by a least-squares approximation approach (see Section **??**) than in the interpolation regime. Thus we delay the explanation until the and merely list the result here:
Letting $\lceil \frac{n}{2} \rceil$ denote the largest integer less than $\frac{n}{2}$ and $\omega := \frac{2\pi}{T}$, put

$$a_k = \frac{2}{n} \sum_{i=0}^{n-1} f_i \cdot \cos\left(\frac{2\pi k i}{n}\right), \quad k = 0, 1, \ldots, m, \tag{3.31}$$

$$b_k = \frac{2}{n} \sum_{i=0}^{n-1} f_i \cdot \sin\left(\frac{2\pi k i}{n}\right), \quad k = 0, 1, \ldots, m, \tag{3.32}$$

the trigonometric interpolation $p(t)$ is given by

$$p(t) = \begin{cases} \frac{a_0}{2} + \sum_{k=1}^{m} \left(a_k \cos(\omega k t) + b_k \sin(\omega k t)\right) & \text{if } n = 2m+1, \\ \frac{a_0}{2} + \sum_{k=1}^{m-1} \left(a_k \cos(\omega k t) + b_k \sin(\omega k t)\right) + \frac{a_m}{2} \cos(\omega m t) & \text{if } n = 2m. \end{cases}$$

3.4.2 Exponential Interpolation

Will be filled soon

3.4.3 Rational Interpolation

Will be filled soon

3.4.4 Extrapolation

Will be filled soon

3.4.5 Interpolation of curves in the plane and in space

Rather often one wants to fit a (closed) curve in $\mathbb{R}^2$ past a number of measured points (see Figure 3.18).

$$P_1 := \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, P_2 := \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}, \ldots, P_n := \begin{pmatrix} x_n \\ y_n \end{pmatrix} = P_1.$$

Such a curve is mathematically given by a function

$$C : [a, b] \longrightarrow \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

where closedness of the curve is usually described by a periodicity condition

$$C(a) = C(b). \tag{3.33}$$

If the curve is thought of as being a smooth curve, then one will certainly assume that $C \in C^1([a, b], \mathbb{R}^1)$ (normally with $C'(t) \neq 0$ everywhere) and that the periodicity of the function value (3.19) is complemented by (at least)

$$C'(a) = C'(b). \tag{3.34}$$

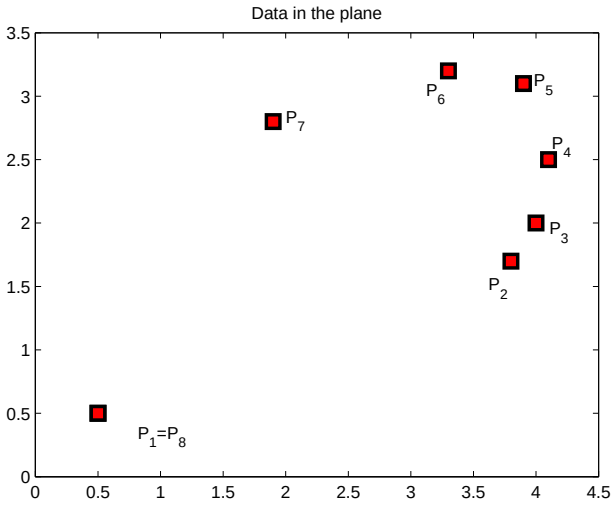

Figure 3.18: Data in the Plane

Thus the curve consists of two functions $x(t)$ and $y(t)$, which one has to replace by substitutes. If one decides to do this by interpolation, there arises as a first problem , that only values $x_1, \ldots, x_n = x_1$ and $y_1, \ldots, y_n = y_1$ are given but no corrsponding argument values $t_1, \ldots, t_n$.
A first idea could be to use $t_i = i, i = 1, \ldots, n$, but it turns out that this is not a good idea if the distances of consecutive points P_i and P_{i+1} vary considerably. In such a case one would use as a "quasi-arclength-approach":

$$t_1 = 0, t_{i+1} = t_i + \|P_{i+1} - P_i\|_2, i = 1, \ldots n - 1. \tag{3.35}$$

If one wants to construct the substitutes of $x(t)$ and $y(t)$ as spline interpolants, then the second question is which boundary conditions to take. The most obvious ones would be the periodic boundary conditions (iv).
If we do not want to program the full spline algorithm ourselves but instead want to use MATLAB, we will discover, that the "spline toolbox" does not belong to our student edition package and that the available command `spline` only offers natural boundary conditions and boundary conditions (1), where we can specify the slopes. To guarantee approximate periodicity, we could implement condition (**??**) by using condition (1) with the same values at the starting and at the end point. Of course one has to experiment then a little to find appropriate slope-values.
In Figure **??** we used $x'(t_1) = x'(t_8) = 1$ and $y'(t_1) = y'(t_8) = -1$ to ensure approximate periodicity. Notice the natural boundary conditions will lead to a corner at P_1. Notice further that $t_i = i, i = 1, \ldots, 8$ leads to nasty wriggles of the curve.

3.4.6 Further Hints

Will befilled soon

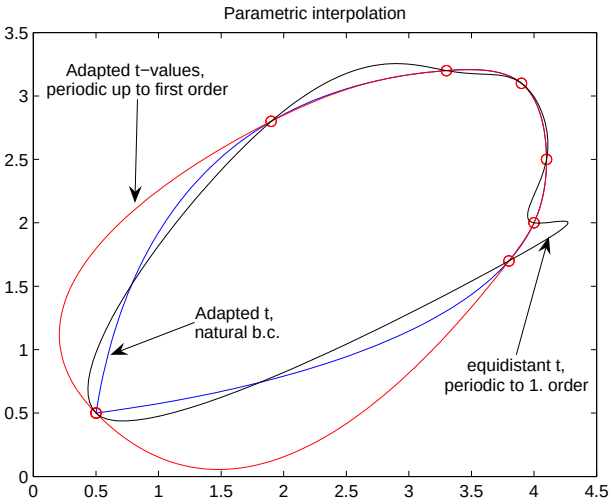

Figure 3.19: Parametric Spline-Interpolation

Chapter 4

Approximation of Linear Functionals

4.1 Introduction

If scientists or engineers collect information on a subject of interest they try to measure that thing. According to Wikipedia "measurement is the process of obtaining the magnitude of a quantity, such as length or mass, relative to a unit of measurement, such as a meter or a kilogram."
Mathematically, measurement is to apply a function, which associates to every object in the domain of the function a real number. (Since the real numbers are totally ordered measuring allows the comparison of things through comparing the measures.
As we heard before functions with values from the real numbers are called functionals in mathematics[1] Hence the mathematical issue of a functional is of fundamental importance for a scientist and an engineer.

When introducing functions at school, one usually starts with linear functions, which are characterized by

$$\textit{Additivity: } f(x+y) = f(x) + f(y)$$
and
$$\textit{Homogeneity: } f(\lambda x) = \lambda f(x), \lambda \in \mathbb{R}.$$

and both ars connected with

proportionality.

Actually, it is a severe problem for teachers of mathematics at the universities to convince students, that not every function is linear: 50 % of our freshmen use to calculate $\sqrt{x^2 + a^2} = x + a$, but this is definitely wrong in general.

A lot of important functionals are linear, however:

[1] Normally the arguments of such functional would be more complicated entities than real numbers. In our actual section these arguments will be functions.

Integrals:

$$f \longmapsto \int_0^1 f(x)dx$$

$$f \longmapsto \int_0^1 e^x f(x)dx$$

$$f \longmapsto \int_0^\infty e^{-x} f(x)dx$$

Evaluations at points:

$$f \longmapsto f(0)$$

$$f \longmapsto f'(3)$$

$$f \longmapsto f''(4)$$

Linear combinations of these:

$$f \longmapsto 3f(0) - 4f(1) + 6f(2)$$

$$f \longmapsto \int_0^3 f(x)dx - (f(0.5) + f(1.5) + f(2.f))$$

$$f \longmapsto f'(0) - (f(1) - f(0))$$

Quite often, one is unable to compute a certain functional. To give an example, the integral

$$Si(x) = \int_0^x \frac{\sin(s)}{s}\, ds$$

can not be described by functions already available. Actually, $Si(x)$ defines a new function, named "the sine integral". Thus the value

$$V := \int_1^2 \frac{\sin(s)}{s}\, ds \tag{4.1}$$

can not be calculated analyticly by usual routines known from analysis.
Certainly other functionals can be computed for the function[2] $f(x) = \frac{\sin(x)}{x}$, e.g. function values. Thus one could try to approximate V by linear combinations of computable functionals .
If values of f (and/or its derivatives) are available then a usual approach is to approximate f by an interpolating function and then to apply the desired functional to this approximation of f instead.

If a function value $f(p)$ at a certain is known, e.g., then the constant interpolation $c(x) = f(p)$ could be used to approximate any integral

$$Inte(f) = \int_a^b w(x)f(x)\, dx; \quad w \text{ a given weight-function}$$

by putting

$$Inte(f) \approx Inte(c) = \int_a^b w(x)c(x)\, dx = f(p) \cdot \int_a^b w(x)\, dx.$$

[2]To mention in passing $f(x) = \frac{\sin(x)}{x}$ is known as the sinc-Function.

Notice that substitutes of function derives from certain data may be of different usefulness for the approximation of different functionals. While the above function $c(x) = f(p)$ might deliver a first guess of an integral, it does certainly not incorporate enough information on f to derive a useful differentiation formula since

$$f'(z) \approx c'(x) = 0.$$

To estimate a slope you surely need at least two function values if you want to do it by linear interpolation. Similarly, if you use an interpolating polynomial for c you certainly will have to use at least $n+1$ function values to estimate an n-th derivative. Otherwise, the interpolating polynomial $p(x)$ would be of degree less than n and the estimate

$$f^{(n)}(z) \approx p^{(n)}(z)$$

will always deliver the value 0.
Notice however, that this "rule" is only true as long as you use polynomial interpolation.
If you have additional advance information about the nature of the functins which you are dealing with, e.g. that your function is an exponential function $f(x) = c\exp(x)$, then you can determine derivatives at all places of all orders just from one function value[3].

We shall be dealing only with the standard approximation via polynomial interpolation.

4.2 Quadrature Formulae for Simple Integral

4.2.1 Introductory Ideas

Replacing f in $\int_a^b f(x)dx$ by its Lagrangian interpolation from (**??**) leads to

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i) \underbrace{\int_a^b \prod_{\substack{k=1 \\ k\neq i}}^{n} \frac{x - x_k}{x_i - x_k} dx}_{w_i} = \sum_{i=1}^{n} w_i f(x_i). \tag{4.2}$$

If $f \in \Pi_{n-1}$, the polynomial interpolation of f with the n pairwise distinct data points will reproduce f. Hence the above **interpolatory quadrature rule** $\int_a^b f(x) \approx \sum_{i=1}^{n} w_i f(x_i)$ will deliver exact integrals for these functions. One says that the formula is exact of (polynomial) order $n-1$.

[3] To know, that f is a multiple of the exponential function is, of course, an information of consequence,which you normally don't have.

Especially, putting $f = 1, x, x^2, \ldots, x^{n-1}$ results in a system of n equations for the n unknowns $w_1, \ldots, w_n$:

$$\begin{aligned} b - a = \int_a^b 1; dx &= \sum_{i=1}^n w_i, \qquad (4.3) \\ \frac{1}{2}(b^2 - a^2) = \int_a^b x; dx &= \sum_{i=1}^n w_i x_i, \\ \frac{1}{6}(b^3 - a^3) = \int_a^b x^2; dx &= \sum_{i=1}^n w_i x_i^2, \\ \vdots & \quad \vdots \\ \frac{1}{n}(b^n - a^n) = \int_a^b x^{n-1}; dx &= \sum_{i=1}^n w_i x_i^{n-1}, \qquad (4.4) \end{aligned}$$

Since the matrix of the system is a Vandermonde system (cf. Section 3.2), it is non singular and the uniquely determined weights $w_1, \ldots, w_n$ of the interpolatory integration formula (4.2) can be computed equally well from this set of equation.
More than this, one does have to determine interpolatory quadrature formulae only for some standard integration interval. $[0, 1]$ or $[-1, 1]$ are usually used as such reference intervals.
If, e.g. an interpolatory quadrature formula

$$\int_0^1 f(x)\, dx \approx Q_{[0,1]}(f) := \sum_{i=1}^n w_i f(x_i)$$

is known for the interval $[0, 1]$, then through

$$\int_a^b f(x)\, dx = (b-a) \int_0^1 \underbrace{f(a + t(b-a))}_{g(t)}\, dt \approx (b-a) Q_{[0,1]} g = \sum (b-a) w_i f(a + x_i(b-a))$$

the method is being transformed to the corresponding formula

$$Q_{[a,b]} f := \sum_{i=1}^n W_i f(y_i)$$

with

$$W_i = (b-a) \cdot w_i \text{ and } y_i := a + x_i(b-a). \qquad (4.5)$$

Since $g(t) := f(a + t(b-a))$ is a polynomial of degree k if f is, the transformed formula $Q_{[a,b]}$ will have the same degree of exactness as $Q_{[0,1]}$ and since the interpolatory formulae with nodes $y_1, \ldots, y_n$ for thee integral from a to b is uniquely determined, $Q_{[a,b]}$ is this formula.

Example 13: The **Simpson formula** is the interpolatory quadrature formula using the endpoints and the midpoint of the integration interval as quadrature nodes. We can calculate its weights in

$$\int_{-1}^1 f(x)\, dx \approx S_{[-1,1]} f = w_1 f(-1) + w_2 f(0) + w_3 f(1)$$

for the case of the reference interval $[-1,1]$ from the three equations

$$\begin{aligned} \int_{-1}^{1} 1\,dx = 1 &= w_1 + w_2 + w_3, \\ \int_{-1}^{1} x\,dx = 0 &= -w_1 + w_3, \\ \int_{-1}^{1} x^2\,dx = \tfrac{2}{3} &= w_1 + w_3, \end{aligned}$$

to be

$$w_1 = w_3 = \frac{1}{3}, w_2 = \frac{4}{3},$$

such that

$$\int_{-1}^{1} p(x)\,dx = S_{[-1,1]}p := \frac{1}{3}p(-1) + \frac{4}{3}p(0) + \frac{1}{3}p(1) \tag{4.6}$$

for all polynomials of degree three, such that the degree of the formula is at least three.
Notice, that this formula has in fact one order of exactness higher than expected, since for $f(x) = x^3$ the equation

$$\int_{-1}^{1} f(x)\,dx = S_{[-1,1]}f := w_1 f(-1) + w_2 f(0) + w_3 f(1)$$

is automatically fulfilled. Due to symmetry this equations will hold even for all uneven power-functions $m_{2n-1}(x) := x^{2n-1}, n \in \mathbb{N}$.

From the Simpson-formula (4.6) for the reference interval $[-1,1]$ the corresponding formula for a general interval is easily found. Due to the transformation (4.5) the new nodes have to lie in the new interval in "the same way as the old ones did in there old interval". Hence the nodes for the interval $[a,b]$ have to lie at the ends of the interval and in the middle. The weights have just to be multiplied by a constant, and the first equation (4.3) of the Vandermonde system above tells us, that the factor has to secure that the sum of the weights is always the length of the integration interval. Hence we have

$$\int_a^b f(x)\,dx \approx S_{[a,b]}f = (b-a)\left\{\frac{1}{6}f(a) + \frac{2}{3}f(\frac{a+b}{2}) + \frac{1}{6}f(b)\right\}.$$

4.2.2 The Newton-Cotes Formulae

The Simpson rule is the second one of a sequence of quadrature rules called the "Newton-Cotes quadrature formulae". These are the interpolatory quadrature formulae using the points of an equidistant mesh

$$a =: x_1 < x_2 = a + k < x_3 = a + 2k < \cdots < a + nk = x_{n+1} = b; \quad k = (b-a)/n$$

as integration nodes.
For $n = 2$ we just derived Simpsons formula. For $n = 1$ we have the Trapezoidal Rule

$$\int_a^b f(x)\,dx \approx T_{[a,b]}f := \frac{b-a}{2} \cdot (f(a) + f(b)),$$

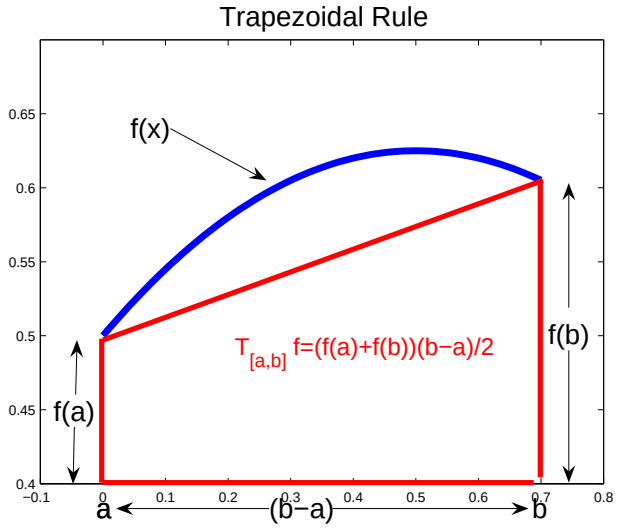

Figure 4.1: Trapezoidal integration rule

which bears its name due to the fact, that the integral of the interpolating polynomial is the area of a trapezoidal region.
If we apply the Trapezoidal rule to

$$V := \int_1^2 \frac{\sin(s)}{s}\, ds = 0.6593299064355....$$

from (4.1) we get the approximation

$$T = 0.64805984911037$$

with an error of $E_T = -0.01127005732514$.
The Simpson formula gives the far better result

$$S = 0.65935105486081$$

with an error of $E_S = 0.00002114842530187566$.
From this experience one might infer that to increase exactness further one will have to use higher order Newton-Cotes formulae, i.e. the $(n = 3)$-rule

$$\int_a^b f(x)\, dx \approx R_{[a,b]}f = (b-a)\left\{\frac{1}{8}f(a) + \frac{3}{8}f(a+(b-a)/3) + \frac{3}{8}f(b-(b-a)/3)\frac{1}{8}f(b)\right\}.$$

leading to

$$R = 0.65933928753017$$

with the error $E_R = 0.0000093810946609$ and then the $(n = 4)$-Milne-rule

$$\int_a^b f(x)\, dx \approx Milne_{[a,b]}f = (b-a)\left\{\frac{7}{90}f(a) + \frac{32}{90}f(a+(b-a)/4) + \frac{12}{90}38f(\frac{a+b}{2}) + \frac{32}{90}f(b-(b-a)/4) + \frac{7}{90}f(b)\right\}.$$

giving

$$Milne = 0.65932988801751$$

with an error of $E_{Mi} = -0.00000001841800678$, and so forth.

Actually one would not further pursue this plan due to at least four reasons:

First: The computation of the subsequent Newton-Cotes weights gets nastier and nastier with increasing n and it becomes more and more difficult to avoid rounding errors.

Second: The sequence of sums of absolute values of the weights $\sum_{i=1}^{n} |w_i^n|$ is not bounded, wherefrom one can prove, that there are continuous functions for which the sequence of Newton-Cotes-approximations of an integral will not converge to this integral.

Third: There are quadrature formulae (the Gauss-formulae, s.b.), which have far better approximation properties, which have all positive weights, such that their sums are bounded, because they equal the length of the integration interval.

Fourth: The techniques of "compound rules" (see (2.15) and below) and of "adaptive integration" can be used with general formulae to approximate an integral to a desired precision.

4.2.3 Gauss-Formulae

To increase the precision of quadrature formulae

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

Gauss had the splendid idea to drop the prescription of the quadrature nodes $x_1, \ldots, x_n$. Introducing them as additional degrees of freedom he could complement the n equations (4.3) to (4.4) by another n equations demanding exact integration of the additional functions $x^n, x^{n+1}, \ldots, x^{2n-1}$.
From (4.3) to (4.4) we readily see, that these set of equations will not be easily solved, because they are nonlinear equations[4].
Hence it is by no means a trivial fact, that for every $n \in \mathbb{N}$ there is exactly one Gaussian quadrature rule, with all its nodes belonging to the integration interval.
Gaussian formulae are generally formulated for the reference interval $[-1, 1]$. The first three of them are

$$\begin{aligned} G1_{[-1,1]}f &= 2f(0), \\ G2_{[-1,1]}f &= f\left(-\frac{1}{\sqrt{2}}\right) + f\left(\frac{1}{\sqrt{2}}\right), \\ G3_{[-1,1]}f &= \frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9}f(0) + \frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right). \end{aligned}$$

As can be seen from the following table

[4] From $x^2 + 1 = 0$ one sees that nonlinear equations must not be solvable at all.

Formula	Result	Error
G_1	0.66499665773604	0.00566675130052
G_2	0.65931579113093	-0.00001411530458517163
G_3	0.65932992413122	$-0.000000017695711...$

the formulae are really much better than the Newton-Cotes ones. With the same number of three function evaluations the G_3 formula delivers nearly the double number of correct digits as the Simpson rule does, and with just one function evaluation G_1 gives better results than the trapezoidal rule with two evaluations does.

Example 14: The first Gaussian formula G_1 is easily computed from the system of equations

$$\int_{-1}^{1} 1 \, dx = 2 = w_1 x_1^0 = w_1, \tag{4.7}$$

$$\int_{-1}^{1} x \, dx = 0 = w_1 x_1, . \tag{4.8}$$

The first equation (4.7) states again that the sum of weights (here just one weight) must be the length of the integration interval. Knowing $w_1 \neq 0$ the second equation delivers $x_1 = 0$, such that G_1 is the midpoint rule, which we already know from (2.14).
The second Gaussian formula is not as easily found from its set of equations.

$$\begin{aligned}
\int_{-1}^{1} 1 \, dx = 2 = w_1 x_1^0 + w_2 x_2^0 = w_1, \\
\int_{-1}^{1} x \, dx = 0 = w_1 x_1 + w_2 x_2, \\
\int_{-1}^{1} x^2 \, dx = \frac{2}{3} = w_1 x_1^2 + w_2 x_2^2, \\
\int_{-1}^{1} x^3 \, dx = 0 = w_1 x_1^3 + w_2 x_2^3.
\end{aligned}$$

Actually, an elegant way to derive the existence of Gaussian formulae and to compute their nodes an weights is cleared by the following Theorem.

Theorem 7: [Gaussian quadrature] The nodes $x_1^n, \ldots, x_n^n$ of the nth Gaussian quadrature rule

$$Gn_{[-1,1],w} := \sum_{i=1}^{n} w_i^n f(x_i^n)$$

to approximate the weighted integral

$$\int_{-1}^{1} w(x) f(x) \, dx$$

for a positive continuous weight function $w(x)$ are the zeros of the nth polynomial $p_n(x) \in \Pi_n$ of the set of polynomials, which are orthogonal in the sense that

$$< p_r, p_s >:= \int_{-1}^{1} w(x) p_r(x) p_s(x)\, dx = 0, \text{ for } r \neq s$$

and normed through $p_n(1) = 1$, e.g.

Remark 8: On can prove a three term recurrence for the orthogonal polynomials. For the case of constant weight-function $w(x) \equiv 1$ the orthogonal polynomials are the well know Legendre polynomials, which can be derived from

$$p_0(x) = 1, \quad p_1(x) = x, \quad (n+1)p_{n+1}(x) = (2n+1)xp_n(x) - np_{n-1}(x), n \geq 1.$$

Remark 9: The interesting theory of orthogonal polynomials has lots of useful applications both in practical mathematics and in theoretical investigations. Properties of Gaussian quadrature play a major role in this theory. Unfortunately we do not have the time to dwell on this subject.

A third approach to explain the extraordinary approximation potential of Gaussian quadrature is connected with osculatory interpolation and due to the Russian mathematician Markov. He proposed to investigate the quadrature formulae which arise from the integration of Hermitian interpolation polynomial interpolation both the function values $f(x_i)$ as well as the derivatives $f'(x_i)$:

$$H(x) = \sum_{i^1}^{n} (f(x_i)\varphi_i(x) + f'(x_i)\psi_i(x))$$

with

$$\varphi_i(x_j) = \delta_{ij}, \varphi'(x_j) = 0, i, j = 1 \ldots, n$$

and

$$\psi_i(x_j) = 0, \psi'(x_j) = \delta_{ij}, i, j = 1 \ldots, n.$$

Then he proves that for the Gaussian points the coefficients of the derivatives in

$$\int_{-1}^{1} H(x)\, dx = \sum_{i=1}^{n} (f(x_i) \int_{-1}^{1} \varphi(x)\, dx + f'(x_i) \int_{-1}^{1} \psi_i(x)\, dx)$$

vanish.

For the Midpoint rule this is just the observation, that the value of area of the rectangle with width $(b-a)$ and height $f(\frac{a+b}{2})$ is the same as that of the trapezium which describes the integral of the first order Taylor-expansion of f at the midpoint.
From the next Figure 4.2 it is clear why the midpoint rule gives better results in the above example than the trapezoidal rule.

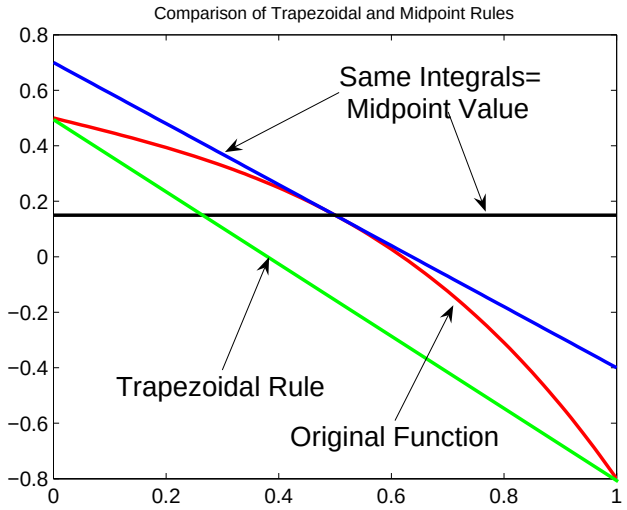

Figure 4.2: Trapezoidal and Midpoint integration rules

4.2.4 Compound Formulae

Just as we did already above in (2.15) for the midpoint rule (2.14) we will define a compound rule for a general quadrature rule

$$\int_{-1}^{1} f(x(\, dx \approx Q_{[-1,1]} = \sum_{i=1}^{n} w_i f(x_i) \tag{4.9}$$

by first fixing the corresponding rule for a general interval through

$$\int_a^b f(x)dx \approx Q_{[a,b]} = \frac{b-a}{2} \sum_{i=1}^{n} w_i f(a + \frac{b-a}{2} x_i). \tag{4.10}$$

Then - by subdividing the actual integration interval $[c, d]$ into $m \in \mathbb{N}$ equal intervals of length $h = 1/m$

$$[c, d] = \bigcup_{k=1}^{m} [c + (k-1)h, c + kh]$$

and applying the quadrature rule to each of the integrals over the subintervals- one defines the "compound rule" :

$$\int_c^d f(x)dx \approx Q_{h,[c,d]} f := \sum_{k=1}^{n} Q_{[c+(k-1)h, c+kh]} f. \tag{4.11}$$

In order to know how fine the grid has to be to guarantee a sufficiently small error, we need error estimates and we turn to them next. For the compound rule with equally sized subintervals of length h we will show, that for functions in $C^m[a, b]$ the error may be bounded by a constant times h^m where $m - 1$ is order of polynomial exactness of the formula.

Before we do this, we investigate the error of $Q_{[a,b]}f$ on a general interval $[a,b]$ where $f \in C^m[a,b]$.
This estimate will be applied both for estimating the behaviour of the compound rule as well as for adaptive integration.

4.2.5 Error-Estimation

Here we will estimate the error of quadrature rules. We start with the investigation of the error of rules for some reference interval, say

$$E_{[-1,1]}(f) := \int_{-1}^{1} f(x)\ dx - Q_{[-1,1]}(f).$$

Then we will show how the error estimate of the same rule, adapted to a general interval $[a,b]$,

$$E_{[a,b]}(f) := \int_{a}^{b} f(x)\ dx - Q_{[a,b]}(f).$$

is connected to $E_{[-1,1]}$.
Finally we will show, how the error estimate for the compound rule

$$E_{h,[c,d]} = \int_{c}^{f} f(x)dx - Q_{h,[c,d]}f$$

is derived from $E_{[a,b]}(f)$.

Quadratur-Error

In preparation

Error of adapted formula

In preparation

Error of a compound rule

In preparation

4.2.6 Adaptive Quadrature

In preparation

4.2.7 Quadrature for Integrals with Weight Functions

In preparation

4.3 Differentiation Formulae

In preparation

4.4 Extrapolative Improvement of Functional Approximations

In preparation

Chapter 5

Solution of systems of linear equations

5.1 Introductory Example

5.1.1 A Simple Truss

tSystems of linear equations arise all over in engineering and natural sciences. Figure (5.2) shows a truss-Model if a bridge with a total number of seven straight members, whose ends are connected at five joints called nodes and denoted by K_1 to K_5. The whole bridge is supported by two bearings, a hinged joint at $K1$ and rollers at K_5. Furthermore an external force E is exerted at node K_5, which results in stresses $x_3, x_4, \ldots, x_9$ in the members, which lead to tensile or compressive forces in the nodes. Since the whole bridge does not move, all the forces including the forces exerted by the internal stresses as well as the forces of sizes x_1, x_2 and x_{10} of the bearings and the external force E have to sum up at their respective nodes of application to give zero.
Whereas the external force E is to be thought of as know, the ten stresses $x_1, \ldots, x_{10}$ are unknown and have to be determined, and we hope and expect, the the balance of forces will endow us with a suitable set of 10 equations.
At node K_1 we introduced two unknown bearing forces of sizes x_1 and x_2 since this hinged joint can exert both horizontal and vertical force components. The right support on rollers has no capacity to bear horizontal loads. Hence there is only one vertical unknown force to be determined[1]. Putting one bearing on rollers has both physical an mathematical reasons. Choosing both bearings as hinged joints would produce physical problems, since the iron members would typically change their lengths with changing temperatures. What would be the right length in summer would be too short in winter and the other way round. Hence such bearings would soon be destroyed by the bridge itself. From a mathematical point of view two hinged bearings would allow for several solutions. It would not at all be clear, who of them would be responsible to cope with a horizontal components of external forces.
We do not have to discuss two roller-bearings, since we usually do not want to go roller-skating with bridges.
To set up equations for x_1 to x_{10} we multiply these by suitable direction vectors to get force vectors which can be summed. One has to observe in this process that if one sets up a force vector at one end of a member by multiplying x_k times a vector v_k with the direction of the member one has to set up

[1] More precisely the bearing can only exert forces with go vertically up. But that has not to be included expressively in our model since bridged do usually not try to fly away.

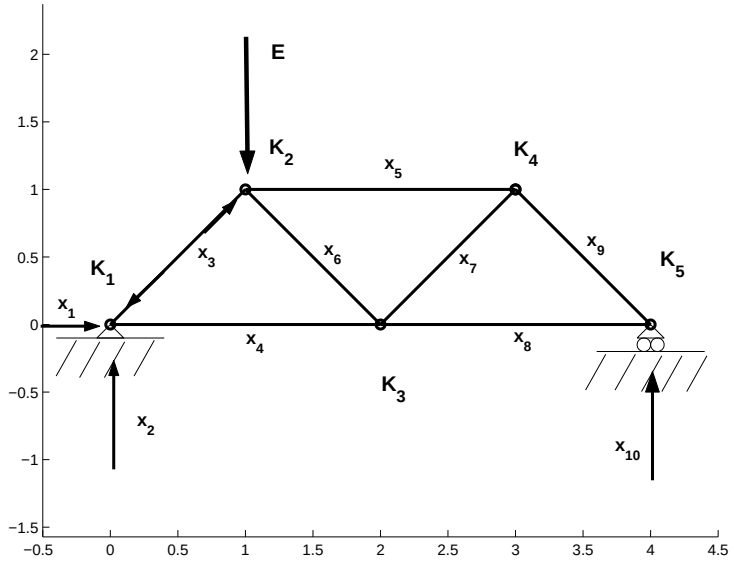

Figure 5.1: Einfaches Stabwerk: Modell einer Brücke

the force vector at the opposite end of the member by multiplying x_k by $-v_k$. If an internal stress leads to a tension force at one node of a member, which points from the node into the member, then there is tension at the other node as well. The same with pressure stresses. If e.g. the impact of x_3 on node K_1 is modeled by

$$V_3 := x_3 \cdot \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

then the impact of x_3 on K_2 has to be modeled by $-V_3$.
The usual question how accompanying direction vectors of the x_k have to be chosen when setting up the system of equation, whether they have to point to the inner of the members or to the outer can be answered (from the mathematical standpoint) that you do not have to care about it. If you change the vector by a factor of -1 then the solution process will put this -1 back on your solution vector by multiplying x_k by the same factor. You can't escape from getting the right answer.
From the engineering standpoint you would rather take direction vectors that point into the members. Tension forces are usually seen as positive whereas pressure forces are negative for engineers[2].

5.1.2 Governing Equations

After this short introduction everyone should be able to put up five vector equations for the unknowns x_1 to x_{10}. We repeat the picture for the convenience of the reader.
Balance of forces in node 1: Balance of forces in node 4:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 \\ 1 \end{pmatrix} x_2 + \begin{pmatrix} -1 \\ -1 \end{pmatrix} x_3 + \begin{pmatrix} -1 \\ 0 \end{pmatrix} x_4 = 0 \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} x_5 + \begin{pmatrix} 1 \\ 1 \end{pmatrix} x_7 + \begin{pmatrix} -1 \\ 1 \end{pmatrix} x_9 = 0$$

[2]Might be there is too much pressure in an engineers education?

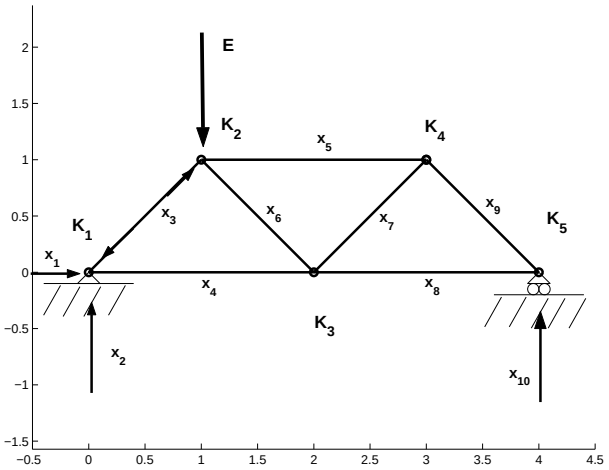

Figure 5.2: Truss of a planar bridge

Balance of forces in node 2:
$$\begin{pmatrix}1\\1\end{pmatrix} x_3 + \begin{pmatrix}0\\-1\end{pmatrix} E + \begin{pmatrix}-1\\0\end{pmatrix} x_5 + \begin{pmatrix}-1\\1\end{pmatrix} x_6 = 0$$

Balance of forces in node 5:
$$\begin{pmatrix}1\\0\end{pmatrix} x_8 + \begin{pmatrix}1\\-1\end{pmatrix} x_9 + \begin{pmatrix}0\\1\end{pmatrix} x_{10} = 0$$

Balance of forces in node 3:
$$\begin{pmatrix}1\\0\end{pmatrix} x_4 + \begin{pmatrix}1\\-1\end{pmatrix} x_6 + \begin{pmatrix}-1\\-1\end{pmatrix} x_7 + \begin{pmatrix}-1\\0\end{pmatrix} x_8 = 0$$

5.1.3 System of Equations

Writing these vector-equations componentwise and collecting all equations we arrive at the system.

$$\begin{array}{rrrrrrrrrrrr}
x_1 & & -x_3 & -x_4 & & & & & & & = & 0 \\
 & x_2 & -x_3 & & & & & & & & = & 0 \\
 & & x_3 & & -x_5 & -x_6 & & & & & = & 0 \\
 & & x_3 & & & +x_6 & & & & & = & E \\
 & & & x_4 & & +x_6 & -x_7 & -x_8 & & & = & 0 \\
 & & & & & -x_6 & -x_7 & & & & = & 0 \\
 & & & & x_5 & & +x_7 & & -x_9 & & = & 0 \\
 & & & & & & x_7 & & +x_9 & & = & 0 \\
 & & & & & & & x_8 & +x_9 & & = & 0 \\
 & & & & & & & & -x_9 & +x_{10} & = & 0
\end{array} \tag{5.1}$$

This can be seen as ten single pieces of information to determine the ten unknowns.
We call this the row-wise interpretation of a linear system.

5.1.4 A Combination Problem

Collecting the coefficients of each of the ten unknown as well as the entities of the right hand sides into vectors from R^{10}, one can write the whole problem as the following task to combine one right-hand-side vector from 10 given vectors:

$$\begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix} x_1 + \begin{pmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix} x_2 + \begin{pmatrix}-1\\-1\\1\\1\\0\\0\\0\\0\\0\\0\end{pmatrix} x_3 + \begin{pmatrix}-1\\0\\0\\0\\-1\\0\\0\\0\\0\\0\end{pmatrix} x_4 + \begin{pmatrix}0\\0\\-1\\0\\0\\0\\-1\\0\\0\\0\end{pmatrix} x_5$$
$$+ \begin{pmatrix}0\\0\\-1\\1\\1\\-1\\0\\0\\0\\0\end{pmatrix} x_6 + \begin{pmatrix}0\\0\\0\\0\\-1\\-1\\1\\1\\0\\0\end{pmatrix} x_7 + \begin{pmatrix}0\\0\\0\\0\\-1\\0\\0\\0\\1\\0\end{pmatrix} x_8 + \begin{pmatrix}0\\0\\0\\0\\0\\0\\-1\\1\\1\\-1\end{pmatrix} x_9 + \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\1\end{pmatrix} x_{10} = \begin{pmatrix}0\\0\\0\\E\\0\\0\\0\\0\\0\\0\end{pmatrix}.$$

5.1.5 Sparse Matrix Formulation

If we put the coefficient-vectors of the last representation columnwise into a 10×10-matrix A and use matrix-vector multiplication, the problem can be written as

$$Ax = b,$$

where x is the vectors of unknowns, b is the vector of the right hand side with $b_4 = E$ and $b_i = 0$ for $i \neq 4$ and finally th matrix A is given by:

$$A = \begin{pmatrix}1&0&-1&-1&0&0&0&0&0&0\\0&1&-1&0&0&0&0&0&0&0\\0&0&1&0&-1&-1&0&0&0&0\\0&0&1&0&0&1&0&0&0&0\\0&0&0&1&0&1&-1&-1&0&0\\0&0&0&0&1&0&1&0&-1&0\\0&0&0&0&0&0&1&0&1&0\\0&0&0&0&0&0&0&1&1&0\\0&0&0&0&0&0&0&0&-1&1\end{pmatrix}$$

5.2 Solvability of General Linear Systems