Numerical Methods for Ordinary Differential Equations

Numerical Methods for Ordinary Differential Equations

C. Vuik P. van Beek F. Vermolen J. van Kan

VSSD

Related titles published by VSSD:
Numerical methods in scientific computing, J. van Kan, A. Segal and F. Vermolen, xii + 279 pp., hardback, ISBN 978-90-71301-50-6
http://www.vssd.nl/hlf/a002.htm

In Dutch:
Numerieke Wiskunde voor Technici, J.J.I.M. van Kan; 128 pp
ISBN 9 78-90-407-1151-0
http://www.vssd.nl/hlf/a002.htm

Numerieke methoden voor differentiaalvergelijkingen, J. van Kan, P. van Beek, F. Vermolen, K. Vuik, x + 122 pp. (Dutch version of this volume)
http://www.vssd.nl/hlf/a018.htm

First edition 2007

Published by VSSD
Leeghwaterstraat 42, 2628 CA Delft, The Netherlands
tel. +31 15 27 82124, telefax +31 15 27 87585, e-mail: hlf@vssd.nl
internet: http://www.vssd.nl/hlf
URL about this book: **http://www.vssd.nl/hlf/a026.htm**

Printed version: ISBN-13 978-90-6562-156-6
Electronic version: ISBN-13 978-90-6562-170-2
NUR 919

Keywords: numerical analysis, ordinary differential equations

Preface

In this book we discuss several numerical methods for solving ordinary differential equations. We emphasize those aspects that play an important role in practical problems. In this introductory text we confine ourselves to ordinary differential equations with the exception of the last chapter in which we discuss the heat equation, a parabolic partial differential equation. The techniques discussed in the introductory chapters, for e.g. interpolation, numerical quadrature and the solution of nonlinear equations, may also be used outside the context of differential equations. They have been included to make the book self contained as far as the numerical aspects are concerned. Chapters, sections and exercises marked * are not part of the Delft Institutional Package.

This text is an English version of a Dutch original “Numerieke Methoden voor Differentiaalvergelijkingen”. I would like to thank Jos van Kan for translating the Dutch text into English and Hisham bin Zubair for correcting the English of this book.

Delft, July 2007
C. Vuik

Contents

1 Introduction

1.1 Some historical remarks

Modern applied mathematics started in the 17th and 18th century with scholars like Stevin, Descartes, Newton and Euler. Numerical aspects found a natural place in the analysis but the expression ”numerical mathematics” did not exist at that time. However, numerical methods invented by Newton, Euler and at a later stage by Gauss still play an important role even today.

In the 17th and the 18th century fundamental laws were formulated for various subdomains of physics, like mechanics and hydrodynamics. These took the form of simple looking mathematical equations. To the disappointment of many, these equations could be solved analytically in a few special cases only. For that reason technological development has been only loosely connected with mathematics. The introduction and availability of the modern digital computer has changed this. Using a computer it is possible to gain quantitative information with detailed and realistic mathematical models and numerical methods for a multitude of phenomena and processes in physics and technology. Application of computers and numerical methods has become ubiquitous. Statistical analysis shows that non-trivial mathematical models and methods are used in 70% of the papers appearing in the professional journals of engineering sciences.

Computations are often cheaper than experiments; experiments can be expensive, dangerous or downright impossible. Real life experiments can often be performed on a small scale only and that makes their results less reliable.

1.2 What is numerical mathematics?

Numerical mathematics is a collection of methods to approximate solutions of mathematical equations numerically by means of *finite* computational processes.

In large parts of mathematics the most important concepts are mappings and sets. In numerical mathematics we have to add the concept of computability. Computability means that the result can be obtained in a finite number of operations (so the computation time will be finite) on a finite subset of the rational numbers (because a computer has only finite memory).

In general the result will be an approximation of the analytic solution of the mathematical problem, since most mathematical equations contain operators based on infinite processes, like integrals and derivatives. Moreover, solutions are functions whose domain and image may (and usually do) contain irrational numbers.

Because, in general, numerical methods can only obtain approximate solutions, it makes sense to apply them only to problems that are insensitive to small perturbations, in other words to problems that are *stable*. The concept of stability belongs to both numerical and classical mathematics. An important instrument in studying stability is functional analysis. This discipline also plays an important role in error analysis: the difference between numerical approximation and exact solution.

Calculating with only a finite subset of the rational numbers has many consequences. For example: a computer cannot distinguish between two polynomials of sufficiently high degree. Consequently we cannot trust methods based on the main theorem of algebra (i.e. that an n-th degree polynomial has exactly n complex roots). Errors that follow from the use of finitely many digits are called *rounding errors*. We shall pay some attention to rounding errors later on in this chapter.

An important aspect of numerical mathematics is the emphasis on efficiency. Contrary to ordinary mathematics, numerical mathematics considers an increase in efficiency, i.e. a decrease of the number of operations and/or amount of storage needed, an essential improvement. Progress in this aspect is of great practical importance and the end of this development has not been reached yet. Here the creative mind will meet many challenges. On top of that, revolutions in computer architecture will overturn much conventional wisdom.

1.3 Why numerical mathematics?

A big advantage of numerical mathematics is that it can provide answers to problems that do not admit analytical solutions. Consider for example the integral

$$\int_0^\pi \sqrt{1+\cos^2 x}\,dx.$$

This is an expression for the arc length of one arc of the curve $y = \sin x$. There is no solution in closed form for this integral. A numerical method, however, can approximate this integral in a very simple way. An additional advantage is, that a numerical method only uses evaluation of standard functions and the operations: addition, subtraction, multiplication and division. Because these are just the operations a computer can perform, numerical mathematics and computers form a perfect combination.

An analytical method gives the solution as a mathematical formula, which is an advantage. From this we can gain insight in the behavior and the properties of the solution, and with a numerical solution (that gives the function as a table) this is not the case. On the other hand some form of visualization may be used to gain insight in the behavior of the solution. To draw a graph of a function with a numerical method is usually a more useful tool than to evaluate the analytical solution at a great number of points.

1.4 Rounding errors

A computer uses a finite representation of real numbers. These are stored in a computer in the form

$$\pm 0.d_1 d_2 \ldots d_n \cdot \beta^e,$$

in which $d_1 > 0$ and $0 \leq d_i < \beta$. We call this a floating point number (representation) in which $0.d_1 d_2 \ldots d_n$ is called the *mantissa*, β the *base* and e (integer) the *exponent*. Often we have $\beta = 2$ (binary representation) and $n = 24$ (*single* precision). In *double* precision we have $n = 56$. We say that the machine computes with n-bit (or n-digit) precision.

Let for $x \in \mathbb{R}$

$$0.d_1 \ldots d_n \cdot \beta^e \leq x < 0.d_1 d_2 \ldots (d_n + 1) \cdot \beta^e,$$

where for simplicity we assume that x is positive. *Rounding* x means, that x will be replaced with the floating point number closest to x, which we shall call $fl(x)$. The error caused by this process is called *rounding error*. Let us write

$$fl(x) = x(1 + \varepsilon). \tag{1.1}$$

We call $|fl(x) - x| = |\varepsilon x|$ the *absolute error* and $\frac{|fl(x)-x|}{|x|} = |\varepsilon|$ the *relative error*. The difference between the floating point numbers enclosing x is β^{e-n}. Rounding gives $|fl(x) - x| \leq \frac{1}{2}\beta^{e-n}$, so for the absolute error we have

$$|\varepsilon x| \leq \frac{1}{2}\beta^{e-n}.$$

Because $|x| \geq \beta^{e-1}$ (since $d_1 > 0$) we have for the relative error:

$$|\varepsilon| \leq eps \tag{1.2}$$

with the computer's relative precision eps defined by

$$eps = \frac{1}{2}\beta^{1-n}. \tag{1.3}$$

From $\beta = 2$ and $n = 24$ it follows that $eps \approx 6 \times 10^{-8}$, so in single precision we calculate with approximately 7 decimal digits.

Figure 1.1 shows the distribution of the floating point numbers $0.1 d_2 d_3 \cdot \beta^e$; $e = -1, 0, 1, 2$ in base 2 (binary numbers). These floating point numbers are not uniformly distributed and there is a neighborhood of 0 that contains no floating point number. A computational result lying within this neighborhood is called *underflow*. Most machines give a warning, replace the result with 0 and continue. A computational result larger than the largest floating point number that can be represented is called *overflow*. The machine warns and halts.

How do computers execute arithmetical operations in floating point arithmetic?

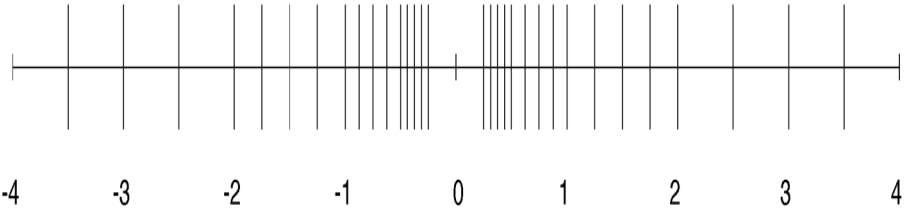

Figure 1.1 Distribution of $\pm 0.1d_2d_3 \cdot \beta^e$, $\beta = 2, e = -1, 0, 1, 2$.

Central processors are very complex and usually the following model is used to simulate reality. Let $\circ$ denote an arithmetic operation ($+, -, \times$ of /) and let x and y be floating point numbers. Then the machine result of the operation $x \circ y$ will be

$$z = fl(x \circ y). \tag{1.4}$$

The exact result of $x \circ y$ will not be a floating point number in general, hence an error results. From formula (1.1) we get

$$z = \{x \circ y\}(1 + \varepsilon), \tag{1.5}$$

for some ε satisfying (1.2) and $z \neq 0$.

Suppose x and y are numbers approximated by the floating point numbers $fl(x)$ and $fl(y)$, so $fl(x) = x(1+\varepsilon_1)$, $fl(y) = y(1+\varepsilon_2)$. We wish to calculate $x \circ y$. The absolute error in the calculated result $fl(fl(x) \circ fl(y))$ satisfies:

$$|x \circ y - fl(fl(x) \circ fl(y))| \leq |x \circ y - fl(x) \circ fl(y)| + |fl(x) \circ fl(y) - fl(fl(x) \circ fl(y))|. \tag{1.6}$$

From this expression we see that the error consists of two terms. The first term is caused by an error in the *data* and the second one by converting the result of an exact calculation to floating point form.

We shall give a few examples to show how rounding errors may affect the result of a calculation. After that we shall give general computational rules regarding the propagation of rounding errors.

Example 1.4.1

Let us take $x = \frac{5}{7}$ and $y = \frac{1}{3}$ and carry out the calculations on a system that uses $\beta = 10$ and a precision of 5 digits. In Table 1.1 you will find the results of various calculations applied to $fl(x) = 0.71429 \times 10^0$ and $fl(y) = 0.33333 \times 10^0$. We shall show how the table has been created. After normalization we find for the addition

$$fl(x) + fl(y) = (.71429 + .33333) \times 10^0 = 0.1047620000... \times 10^1$$

This result has to be rounded to 5 digits:

$$fl(fl(x) + fl(y)) = 0.10476 \times 10^1.$$

Table 1.1 Absolute and relative error for various calculations.

operation	result	exact value	absolute error	relative error
$x+y$	0.10476×10^1	$22/21$	0.190×10^{-4}	0.182×10^{-4}
$x-y$	0.38096×10^0	$8/21$	0.761×10^{-5}	0.200×10^{-4}
$x \times y$	0.23809×10^0	$5/21$	0.523×10^{-5}	0.220×10^{-4}
$x \div y$	0.21429×10^1	$15/7$	0.429×10^{-4}	0.200×10^{-4}

The exact value is $x+y = \frac{22}{21} = 1.0476190518...$ So the absolute error is $1.0476190518... - 0.10476 \times 10^1 \approx 0.190 \times 10^{-4}$ and the relative error is $\frac{0.190 \times 10^{-4}}{22/21} \approx 0.182 \times 10^{-4}$.

The error analysis of the other three operations follows the same lines.

Example 1.4.2

In this example we will use the same numbers x and y and the same precision as in the previous example. Further we use $u = 0.714251$, $v = 98765.1$ and $w = 0.111111 \times 10^{-4}$, so $fl(u) = 0.71425$, $fl(v) = 0.98765 \times 10^5$ and $w = 0.11111 \times 10^{-4}$. These numbers have been chosen in such a way that we can clearly illustrate what problems we may expect with rounding errors. In Table 1.2 $x-u$ has a small absolute error but a large relative error. If we divide $x-u$ by a small number w or multiply it with a large number v, the absolute error increases, whereas the relative error is not affected. On the other hand, adding a larger number u to a small number v results in a large absolute error but only a small relative error. We shall show how the first row has been created. The exact result

Table 1.2 Absolute and relative error for various calculations

operation	result	exact value	absolute error	relative error
$x-u$	0.40000×10^{-4}	0.34714×10^{-4}	0.528×10^{-5}	0.152
$(x-u)/w$	0.36000×10^1	0.31243×10^1	0.476	0.152
$(x-u) \times v$	0.39506×10^1	0.34287×10^1	0.522	0.152
$u+v$	0.98765×10^5	0.98766×10^5	0.814×10^0	0.824×10^{-5}

is $u = 0.714251$ and $x-u = \frac{5}{7} - .714251 = 0.3471428571.... \times 10^{-4}$, whereas $fl(u) = 0.71425 \times 10^0$ and $fl(x) - fl(u) = 0.71429 - 0.71425 = 0.0000400000 \times 10^0$. Normalization gives $fl(fl(x) - fl(u)) = 0.40000 \times 10^{-4}$. From this we obtain the absolute error: $(x-u) - fl(fl(x) - fl(u)) = (.3471428571... - .40000) \times 10^{-4} \approx 0.528 \times 10^{-5}$ and the relative error:
$\frac{0.528... \times 10^{-5}}{0.3471428... \times 10^{-4}} \approx 0.152$.

It is interesting to note, that the large relative error has nothing to do with the limitations of the floating point system (the subtraction of $fl(x)$ and $fl(u)$ is without error in this case) but is due only to the fact that the data is represented in no more than 5 decimal digits. The

zeros that remain after normalization in the single precision result $fl(fl(x) - fl(u)) = 0.40000$ have no significance, only the digit 4 is significant; the zeros that have been substituted are a mere formality and represent no information. This phenomenon is called *loss of significant digits*. The loss of significant digits has a large impact on the relative error, because of division by the small result.

A large relative error sooner or later will have some unpleasant consequences in later stages of the process, also for the absolute error. If we multiply for example $x - u$ by a large number, then we immediately also generate a large absolute error, together with the large relative error we already had. As an example we look at the third row of the table. The exact result is $(x - u) \times v = 3.4285594526000....$ Calculating $fl(fl(x) - fl(u)) \times fl(v)$ gives:

$$fl(fl(x) - fl(u)) \times fl(v) = 0.4 \times 10^{-4} \times 0.98765 \times 10^{5} = 0.3950600000 \times 10^{1}.$$

After rounding we get: $fl(fl(fl(x) - fl(u)) \times fl(v)) = 0.39506 \times 10^{1}$. This yields the absolute error: $3.42855990000460..... - 0.39506 \times 10^{1} \approx 0.522$ and the relative error: $\frac{0.522..}{3.4285....} \approx 0.152$. Suppose we add something to $(x - u) \times v$, for example: y^2; because $y = \frac{1}{3}$ and therefore $y^2 = \frac{1}{9}$, the result of this operation due to the large absolute error is indistinguishable. In other words, for the reliability of the result it does not make a difference whether we would omit the last operation and by doing that alter the numerical process. So we conclude that something is fundamentally wrong in this case.

Almost all numerical processes exhibit loss of significant digits for a certain set of input data; one might call such a set *ill conditioned*. There also are numerical processes that exhibit these phenomena for all possible input data. Such processes are called *unstable*. One of the objectives of numerical analysis is to identify unstable processes and classify them as useless. Or improve them in such a way that they become stable.

Computational rules for error propagation

In the analysis of a complete numerical process, in each subsequent step we have to interpret the accumulated error of all previous steps as a perturbation of the original data. Moreover, in the result of this step we have to take into account the propagation of these perturbations together with the floating point error. After a considerable number of steps this error source will be more important than the floating point error most of the time. (In the previous example of $(x - u) \times v$ even after two steps!) In that stage the error in a numerical process will be largely determined by the 'propagation' of the accumulated errors. The computational rules to calculate numerical error propagation are the same as those to calculate propagation of error in measurements in physical experiments. There are two rules: one for addition and subtraction and one for multiplication and division.

The approximations of x and y will be denoted by $\tilde{x}$ and $\tilde{y}$ and the (absolute) perturbations $\delta x = x - \tilde{x}$ and $\delta y = y - \tilde{y}$.

a) Addition and subtraction.
   $(x + y) - (\tilde{x} + \tilde{y}) = (x - \tilde{x}) + (y - \tilde{y}) = \delta x + \delta y$, in other words, the absolute error in the sum of two perturbed terms is equal to the sum of the absolute perturbations.

A similar rule holds for differences: $(x-y)-(\tilde{x}-\tilde{y}) = \delta x - \delta y$. Often the rule is presented in the form of an inequality (also called an *error estimate*.): $|(x \pm y) - (\tilde{x} \pm \tilde{y})| \leq |\delta x| + |\delta y|$.

b) This rule does not hold for multiplication and division. Efforts to derive a rule for *absolute* error will lead nowhere. But one may derive a similar rule for the *relative* error.

The *relative* perturbations ε_x and ε_y are defined by $\tilde{x} = x(1-\varepsilon_x)$, and similarly for y. For the relative error in a product xy we have: $\frac{xy-\tilde{x}\tilde{y}}{xy} = \frac{xy-x(1-\varepsilon_x)y(1-\varepsilon_y)}{xy} = \varepsilon_x + \varepsilon_y - \varepsilon_x\varepsilon_y \approx \varepsilon_x + \varepsilon_y$, assuming ε_x and ε_y are negligible compared to 1. In words: the relative error in a product of two perturbed factors is approximately equal to the sum of the two relative perturbations. A similar rule can be derived for division. Formulated as an error estimate we have $|\frac{xy-\tilde{x}\tilde{y}}{xy}| \leq |\varepsilon_x| + |\varepsilon_y|$.

Identification of $\tilde{x}$ with $fl(x)$ and $\tilde{y}$ with $fl(y)$ enables us to explain clearly various phenomena in floating point computations using these two simple rules.

1.5 Landau's O-symbol

In the analysis of numerical methods estimating the error is of prime importance. It is often more important to have an indication of the order of magnitude of the error than a precise expression. To save ourselves some tedious work we use Landau's O-symbol.

Definition 1.5.1 *Let f and g be given functions. We say $f(x) = O(g(x))$ ("$f(x)$ is big Oh of $g(x)$") for $x \to 0$, if there exist positive r and finite M such that*

$$|f(x)| \leq M|g(x)| \quad \textit{for all} \quad x \in [-r,r] \,.$$

To estimate errors we often use the following computational rules.

Computational rules
If $f(x) = O(x^p)$ and $g(x) = O(x^q)$ for $x \to 0$, with $p \geq 0$ and $q \geq 0$ then

a) $f(x) = O(x^s)$ for all s with $0 \leq s \leq p$.

b) $\alpha f(x) + \beta g(x) = O(x^{\min\{p,q\}})$ for all $\alpha, \beta \in \mathbb{R}$.

c) $f(x)g(x) = O(x^{p+q})$.

d) $\frac{f(x)}{|x|^s} = O(x^{p-s})$ if $0 \leq s \leq p$.

1.6 Some important theorems from analysis

In this section we recollect some important theorems from analysis that are often used in numerical analysis. In this book we use the notation $C[a,b]$ for the set of all functions continuous on the interval $[a,b]$ and $C^p[a,b]$ for the set of all functions of which all derivatives up to the p-th exist and are continuous.

Theorem 1.6.1 (Intermediate value theorem) *Assume $f \in C[a,b]$. Let $f(a) \neq f(b)$ and let F be a number between $f(a)$ and $f(b)$. Then there exists a number $c \in (a,b)$ such that $f(c) = F$.*

Theorem 1.6.2 (Rolle's theorem) *Assume $f \in C[a,b]$ and f differentiable on (a,b). If $f(a) = f(b)$, then there exists a number $c \in (a,b)$ such that $f'(c) = 0$.*

Theorem 1.6.3 (Mean value theorem) *Assume $f \in C[a,b]$ and f differentiable on (a,b), then there exists a number $c \in (a,b)$ such that $f'(c) = \frac{f(b)-f(a)}{b-a}$.*

Theorem 1.6.4 (Taylor polynomial) *Assume $f : (a,b) \to \mathbb{R}$ is $(n+1)$ times differentiable. Then for all $c, x \in (a,b)$ there exists a number ξ between c and x such that*

$$f(x) = P_n(x) + R_n(x),$$

in which the Taylor polynomial $P_n(x)$ *is given by*

$$P_n(x) = f(c) + (x-c)f'(c) + \frac{(x-c)^2}{2!}f''(c) + \ldots + \frac{(x-c)^n}{n!}f^{(n)}(c)$$

and the remainder term $R_n(x)$ *is:*

$$R_n(x) = \frac{(x-c)^{n+1}}{(n+1)!}f^{(n+1)}(\xi).$$

Proof
Take $c, x \in (a,b)$ with $c \neq x$ and let K be defined by:

$$f(x) = f(c) + (x-c)f'(c) + \frac{(x-c)^2}{2!}f''(c) + \ldots + \frac{(x-c)^n}{n!}f^{(n)}(c) + K(x-c)^{n+1}. \quad (1.7)$$

Consider the function

$$F(t) = f(t) - f(x) + (x-t)f'(t) + \frac{(x-t)^2}{2!}f''(t) + \ldots + \frac{(x-t)^n}{n!}f^{(n)}(t) + K(x-t)^{n+1}.$$

By (1.7) we have $F(c) = 0$ and $F(x) = 0$. Hence, by Rolle's theorem there exists a number ξ between c and x such that $F'(\xi) = 0$. Further elaboration gives

$$\begin{aligned} F'(\xi) &= f'(\xi) + \{f''(\xi)(x-\xi) - f'(\xi)\} + \{\frac{f'''(\xi)}{2!}(x-\xi)^2 - f''(\xi)(x-\xi)\} + \\ &\quad + \cdots + \{\frac{f^{(n+1)}(\xi)}{n!}(x-\xi)^n - \frac{f^{(n)}(\xi)}{(n-1)!}(x-\xi)^{(n-1)}\} - K(n+1)(x-\xi)^n = \\ &= \frac{f^{(n+1)}(\xi)}{n!}(x-\xi)^n - K(n+1)(x-\xi)^n = 0. \end{aligned}$$

So $K = \frac{f^{(n+1)}(\xi)}{(n+1)!}$, which proves the theorem. ⊠

Theorem 1.6.5 (Taylor polynomial of two variables) *Let $f : D \subset \mathbb{R}^2 \mapsto \mathbb{R}$ be continuous with continuous partial derivatives up to and including order $n+1$ in a ball $B \subset D$ with center $\mathbf{c} = (c_1, c_2)$ and radius ρ. Then for each $\mathbf{x} = (x_1, x_2) \in B$ there exists a $\theta \in (0,1)$, such that*

$$f(\mathbf{x}) = P_n(\mathbf{x}) + R_n(\mathbf{x}),$$

in which the Taylor polynomial $P_n(\mathbf{x})$ is given by

$$P_n(\mathbf{x}) = f(\mathbf{c}) + (x_1 - c_1)\frac{\partial f}{\partial x_1}(\mathbf{c}) + (x_2 - c_2)\frac{\partial f}{\partial x_2}(\mathbf{c}) + \frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{2}(x_i - c_i)(x_j - c_j)\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{c}) + \ldots$$

$$+\frac{1}{n!}\sum_{i_1=1}^{2}\sum_{i_2=1}^{2}\cdots\sum_{i_n=1}^{2}(x_{i_1} - c_{i_1})(x_{i_2} - c_{i_2})\ldots(x_{i_n} - c_{i_n})\frac{\partial^n f}{\partial x_{i_1}\partial x_{i_2}\ldots\partial x_{i_n}}(\mathbf{c})$$

and the remainder term is

$$R_n(\mathbf{x}) = \frac{1}{(n+1)!}\sum_{i_1=1}^{2}\sum_{i_2=1}^{2}\cdots$$

$$\sum_{i_{n+1}=1}^{2}(x_{i_1} - c_{i_1})(x_{i_2} - c_{i_2})\ldots(x_{i_{n+1}} - c_{i_{n+1}})\frac{\partial^{n+1} f}{\partial x_{i_1}\partial x_{i_2}\ldots\partial x_{i_{n+1}}}(\mathbf{c} + \theta(\mathbf{x} - \mathbf{c}))$$

Proof
Let for fixed $\mathbf{x}$ and $\mathbf{h}$ with $\|\mathbf{h}\| < \rho$, the function $F : (-1,1) \mapsto \mathbb{R}$ be defined by:

$$F(s) = f(\mathbf{x} + s\mathbf{h}).$$

Because of the differentiability conditions satisfied by f in the ball B, F is $(n+1)$ times continuously differentiable on the interval $(-1,1)$ and $F^k(s)$ is given by (check this!)

$$F^k(s) = \sum_{i_1=1}^{2}\sum_{i_2=1}^{2}\cdots\sum_{i_k=1}^{2}\frac{\partial^k f(\mathbf{x} + s\mathbf{h})}{\partial x_{i_1}\partial x_{i_2}\ldots\partial x_{i_k}}h_{i_1}h_{i_2}\ldots h_{i_k}.$$

Expand F into a Taylor polynomial about 0. This yields:

$$F(s) = F(0) + sF'(0) + \ldots + \frac{s^n}{n!}F^n(0) + \frac{s^{n+1}}{(n+1)!}F^{n+1}(\theta s),$$

for some $\theta \in (0,1)$. Now substitute $s = 1$ into this expression and into the expressions for the derivatives of F and the result follows. ⊠

Example
For $n = 1$ we get:

$$P_1(\mathbf{x}) = f(c_1, c_2) + (x_1 - c_1)\frac{\partial f}{\partial x_1}(c_1, c_2) + (x_2 - c_2)\frac{\partial f}{\partial x_2}(c_1, c_2),$$

and for the remainder term: $R_1(\mathbf{x})$ is $O(\|\mathbf{x} - \mathbf{c}\|^2)$.

Theorem 1.6.6 (Power series of $\frac{1}{1-x}$) *Let $x \in \mathbb{R}$ with $|x| < 1$. Then:*

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k.$$

Theorem 1.6.7 (Power series of e^x) *Let $x \in \mathbb{R}$. Then:*

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

1.7 Summary

In this chapter the following subjects have been discussed

- Numerical mathematics
- Rounding errors
- Landau's O-symbol
- Some important theorems from analysis

1.8 Exercises

1. Let $f(x) = x^3$. Determine the second order Taylor polynomial of f about the point $x = 1$. Compute the value of this polynomial in $x = 0.5$. Give an error estimate and compare this with the actual error.

2. Let $f(x) = e^x$. Give the n-th order Taylor polynomial about the point $x = 0$ and also give the remainder term. How large should n be chosen in order to make the error less than 10^{-6} in the interval $[0, 0.5]$?

3. We use the polynomial $P_2(x) = 1 - \frac{1}{2}x^2$ to approximate $f(x) = \cos(x)$ in the interval $[-\frac{1}{2}, \frac{1}{2}]$. Give an upper bound for the error in this approximation.

4. Let $x = \frac{1}{3}$, $y = \frac{5}{7}$. We calculate with a precision of 3 (decimal) digits. Express x and y as floating point numbers. Compute $fl(fl(x) \circ fl(y))$, $x \circ y$ and the rounding error taking $\circ = +, -, *, /$ respectively.

2 Interpolation

2.1 Introduction

In practice we often have to determine intermediate values from a limited number of measurements (interpolation) or to predict values outside the range of measurements (extrapolation) Let us take as an example the number of chickens in Dutch chicken farms. In Table 2.1 the numbers have been tabulated (in millions) from 1970 every fifth year up to 1995. How can we use these numbers to estimate the number of chickens in intermediate years, e.g. in 1992 or predict the number in the year 2000? In this chapter we shall consider a number of interpolation and extrapolation methods to tackle this problem. Also in visualizing images on a computer screen it is possible to save much memory by

Table 2.1 Number of chickens (in millions) in the Netherlands (source: NRC 09-12-1998).

year	1970	1975	1980	1985	1990	1995
number	53	68	82	92	94	92

not storing every pixel but only a limited number. Through these points a curve can be constructed to render a more or less realistic image on the screen.

As a final application consider computing the values of a trigonometric function on a computer. Calculating such a value is time consuming. A solution: store a number of precalculated function values in memory and determine from these the values at intermediate points in a cheap way.

2.2 Linear interpolation

The simplest way to interpolate is zeroth degree interpolation. Suppose the function value at a certain point is known. We choose the value of the approximation in a neighborhood of this point equal to this known value. A well known example is the prediction that tomorrow's weather will be the same as today's. This prediction appears to be correct in 80% of all cases. (In the Sahara this percentage is even higher.)

A better way of interpolation is a straight line between two points (see Figure 2.1). Suppose we know the value of a function f at the points x_0 and x_1: $f(x_0)$ and $f(x_1)$. If we lack any further information it seems plausible to take as function value in x the value of the linear function (the graph of which is a straight line) through $(x_0, f(x_0))$, $(x_1, f(x_1))$. It is easily shown, that this function is given by

$$p(x) = f(x_0) + \frac{x - x_0}{x_1 - x_0}(f(x_1) - f(x_0))$$

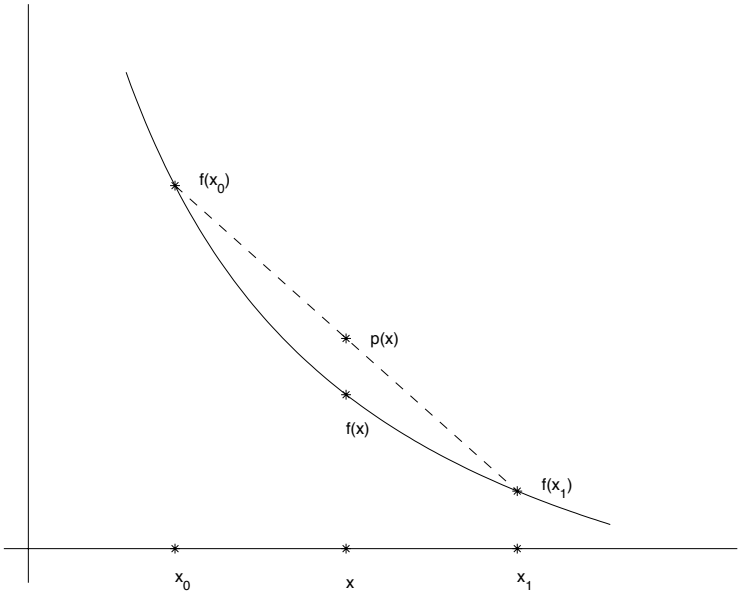

Figure 2.1 Linear interpolation.

or

$$p(x) = \frac{x-x_1}{x_0-x_1} f(x_0) + \frac{x-x_0}{x_1-x_0} f(x_1) \,.$$

The function p is a linear interpolation polynomial that is equal to $f(x_0)$ in x_0 and $f(x_1)$ in x_1. An obvious question is of course: how large is the error of linear interpolation and on what does it depend? We can say something about this error if we know that the interpolated function is at least twice continuously differentiable. Note: by $[a,b,x_0]$ we mean the interval spanned by the extremes of the three values a, b, and x_0.

Theorem 2.2.1 *Let x_0 and x_1 be points in $[a,b]$, $x_0 \neq x_1$ and $f \in C[a,b] \cap C^2(a,b)$. The linear interpolation polynomial p of f in the nodes x_0, x_1 satisfies: for each $x \in [a,b]$ there exists a $\xi \in (x_0,x_1,x)$ such that*

$$f(x) - p(x) = \frac{1}{2}(x-x_0)(x-x_1) f''(\xi) \,. \tag{2.1}$$

Proof
If $x = x_0$ or $x = x_1$, then $f(x) - p(x) = 0$ and ξ can be chosen arbitrarily. Assume $x \neq x_0$ and $x \neq x_1$. For each x there exists a number q such that

$$f(x) - p(x) = q(x-x_0)(x-x_1) \,.$$

To find an expression for q consider the function

$$\varphi(t) = f(t) - p(t) - q(t-x_0)(t-x_1) \,.$$

φ satisfies $\varphi(x_0) = \varphi(x_1) = \varphi(x) = 0$. By Rolle's theorem, there exist at least two points y and z in (x_0, x_1, x) such that $\varphi'(y) = \varphi'(z) = 0$. Again by Rolle's theorem there is a $\xi \in (y,z)$ and hence $\xi \in (x_0, x_1, x)$ such that $\varphi''(\xi) = 0$. Because $\varphi''(t) = f''(t) - 2q$ this means that $q = \frac{1}{2} f''(\xi)$. ⊠

If $x \notin [x_0, x_1]$ we use the polynomial to extrapolate. Relation (2.1) is still the correct expression for the error.

From this theorem an upper bound for *linear interpolation* follows:

$$|f(x) - p(x)| \leq \frac{1}{8}(x_1 - x_0)^2 \max_{\xi \in [x_0, x_1]} |f''(\xi)| \,.$$

In many practical applications the values $f(x_0)$ and $f(x_1)$ are a result of measurements or calculations. Hence these values may contain errors. Suppose that the absolute error is at most ε. The difference between the exact polynomial p and the perturbed polynomial $\hat{p}$ is bounded by

$$|p(x) - \hat{p}(x)| \leq \frac{|x_1 - x| + |x - x_0|}{x_1 - x_0} \varepsilon \,.$$

For interpolation this error is always bounded by ε. For extrapolation the error may be larger than ε. Suppose $x \geq x_1$ then the additional inaccuracy is bounded by

$$|p(x) - \hat{p}(x)| \leq (1 + 2\frac{x - x_1}{x_1 - x_0})\varepsilon \,.$$

The total error is the sum of the interpolation/extrapolation error and the measurement error.

Example 2.2.1 (linear interpolation)

The value of the sine function has been given for 36° and 38° (see table). The linear interpolation approximation for 37° gives a result of 0.601723. The difference with the exact value is only 0.9×10^{-4}.

Table 2.2 The value of $\sin\alpha$.

α	$\sin\alpha$
36°	0.58778525
37°	0.60181502
38°	0.61566148

2.3 Lagrangian interpolation

If there are more than two data points it makes sense to use those extra points too. An obvious method is to use higher degree interpolation. Because a polynomial of degree

n contains $n+1$ independent parameters we need $n+1$ data points to construct an n-th degree interpolation polynomial.

As a generalization of linear interpolation we consider the approximation of a function f by a polynomial $L_n(x)$) of degree at most n, such that the values of f and L_n at $n+1$ different points $x_0, x_1, \ldots, x_n$ coincide. This we call n-th order Lagrangian interpolation.

The polynomial L_n satisfying these constraints is easy to find. It has a shape that is a clear generalization of Equation (2.1):

$$L_n(x) = \sum_{k=0}^{n} f(x_k) L_{kn}(x) \,,$$

in which

$$L_{kn}(x) = \frac{(x-x_0)....(x-x_{k-1})(x-x_{k+1})....(x-x_n)}{(x_k-x_0)....(x_k-x_{k-1})(x_k-x_{k+1})....(x_k-x_n)} \,.$$

That this is the right polynomial will be clear from the following observations:

- Each L_{kn} is a polynomial of degree n,
- From $L_{kn}(x_j) = \delta_{kj}$ it follows that $L_n(x_k) = f(x_k), k \in \{0, \ldots, n\}$.

The polynomial L_n is called *Lagrangian interpolation polynomial*. The polynomials L_{kn} are called *Lagrangian coefficients*. They can also be written as:

$$L_{kn}(x) = \frac{\omega(x)}{(x-x_k)\omega'(x_k)} \quad \text{with} \quad \omega(x) = \prod_{i=0}^{n} (x-x_i) \,.$$

Theorem 2.2.1 now can be generalized:

Theorem 2.3.1 *Let $x_0, \ldots, x_n$ be different node points in $[a,b]$. Let $f \in C^n[a,b] \cap C^{n+1}(a,b)$ and let L_n be the Lagrangian polynomial generated by f and these node points. Then for each $x \in [a,b]$ there exists a $\xi \in (x_0, x_1, \ldots, x_n, x)$ such that*

$$f(x) - L_n(x) = (x-x_0)....(x-x_n)\frac{f^{n+1}(\xi)}{(n+1)!} \,.$$

Proof:
The proof is completely analogous to that of Theorem 2.2.1. ⊠

If we use Lagrangian interpolation on tabular values, the best results are obtained by choosing the node points in such a way that x is in the (or an) innermost interval. Explain why.

We want to know which errors may occur in higher order interpolation besides the interpolation error if the function values or tabulated values are not exact. Assume that the error in the values is at most ε. Then the error in the perturbed interpolation polynomial is at most

$$\sum_{k=0}^{n} |L_{kn}(x)| \varepsilon \,.$$

Table 2.3 Upper bounds for $\sum_{k=0}^{n} |L_{kn}(x)|$.

	$x \in [x_0, x_1]$	$x \in [x_1, x_2]$	$x \in [x_2, x_3]$	$x \in [x_3, x_4]$	$x \in [x_4, x_5]$
$n = 1$	1				
$n = 2$	1.25	1.25			
$n = 3$	1.63	1.25	1.63		
$n = 4$	2.3	1.4	1.4	2.3	
$n = 5$	3.1	1.6	1.4	1.6	3.1

If the nodes are equidistant, $x_k = x_0 + kh$, the value of $\sum_{k=0}^{n} |L_{kn}(x)|$ increases slowly with n. In Table 2.3 the reader will find a number of upper bounds.

In general one would expect the approximation error to decrease with increasing polynomial degree. However, this is not always the case as the next example shows.

Example 2.3.1 (interpolation)

Consider the function $\frac{1}{1+x^2}$ on the interval $[-5,5]$. We use the points $x_k = -5 + \frac{10k}{n}, k = 0, \ldots, n$ as interpolation nodes. In Figure 2.2 you will find a graph of the function, together with the 6th and 14th degree interpolation polynomials. Note that, on the interval [-3, 3], the 14th degree polynomial approximation is better than that of degree 6. In the neighborhood of the end points, however, the 14th degree polynomial exhibits large aberrations.

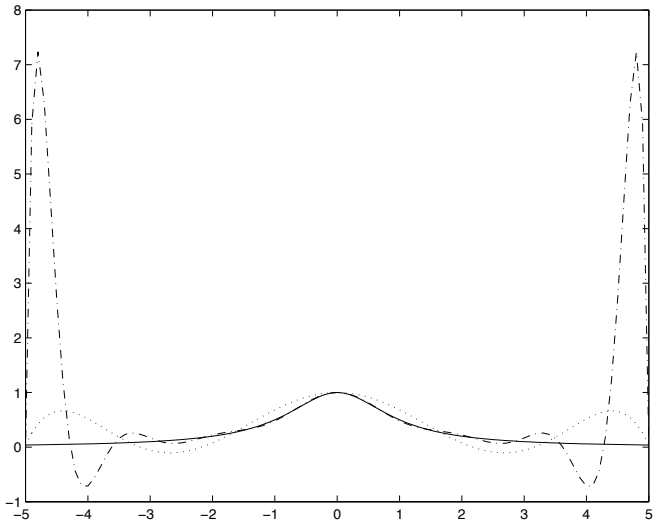

Figure 2.2 Interpolation of function $\frac{1}{1+x^2}$ $(-)$ *with a 6-th degree Lagrangian polynomial* $(\cdots)$ *and a 14-th degree Lagrangian polynomial* $(- \cdot - \cdot -)$.

Example 2.3.2 (extrapolation)

A similar phenomenon may occur in extrapolation. Consider the function $\frac{1}{x}$. The n-th degree interpolation polynomial is determined with nodes $x_k = 0.5 + \frac{k}{n}$, $k = 0, ..., n$. Let us plot the graphs of the function, the 6-th degree and the 10-th degree interpolation polynomial on the interval [0.5, 1.8] (Figure 2.3). The polynomials and the function itself are indistinguishable on the interval [0.5, 1.5]. With extrapolation $(x \geq 1.5)$, however, large errors occur. Again the largest error occurs in the 10-th degree polynomial.

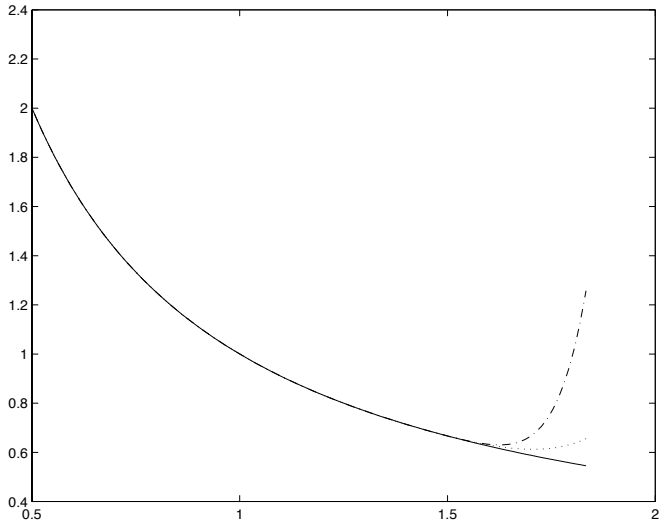

Figure 2.3 Extrapolation of function $\frac{1}{x}$ $(-)$ *with a 6-th degree Lagrangian polynomial* $(\cdots)$ *and a 10-th degree Lagrangian polynomial* $(-\cdot-\cdot-)$.

2.4 Interpolation with function values and derivatives *

2.4.1 Taylor polynomial

A well known method to approximate a function by a polynomial is Taylor's method. As an approximation method it is not used very often in numerical mathematics, but instead *is* used quite often for the analysis of numerical processes.

In many cases the approximation of the Taylor polynomial gets better with increasing polynomial degree. But this is not always true. We show this in the following example:

Example 2.4.1 (Taylor polynomial)

We want to approximate the function $f(x) = \frac{1}{x}$ in $x = 3$ by a Taylor polynomial of degree n about the node point $x = 1$. (Note that $x = 3$ is outside the region of convergence of the Taylor series.) The derivatives are given by: $f^{(k)}(x) = (-1)^k k! x^{-(k+1)}$. The n-th degree

Taylor polynomial is:

$$p_n(x) = \sum_{k=0}^{n} f^{(k)}(1)(x-1)^k/k! = \sum_{k=0}^{n} (-1)^k (x-1)^k \,.$$

The values of $p_n(3)$ as an approximation of $f(3) = \frac{1}{3}$ are tabulated in Table 2.4. The approximation gets less accurate with increasing polynomial degree.

Table 2.4 Value of the Taylorpolynomial in $x = 3$ with increasing degree (n).

n	0	1	2	3	4	5	6	7
$p_n(3)$	1	-1	3	-5	11	-21	43	-85

2.4.2 Interpolation in general

In general, given a function f we look for a polynomial, such that at a number of different node points $x_0, \ldots, x_n$ not only the values of f and p coincide, but also the value of their derivatives up to and including order m_i at x_i. (m_i could be different for different nodal points)

Suppose that $f \in C^m[a,b]$ with $m = \max\limits_{0 \le i \le n} m_i$. Then p is the polynomial of lowest degree such that

$$\frac{d^k p}{dx^k}(x_i) = \frac{d^k f}{dx^k}(x_i) \quad \text{for each} \quad i = 0, 1, \ldots, n \quad \text{and} \quad k = 0, 1, \ldots, m_i \,.$$

Remarks

1. This polynomial is at most of degree $M = \sum\limits_{i=0}^{n} m_i + n$.
2. If $n = 0$ then p is the Taylor polynomial about x_0 of degree m_0.
3. If $m_i = 0$ then p is the n-th degree Lagrangian polynomial of f in the nodes $x_0, x_1, \ldots, x_n$.

As an example of general interpolation we consider in the next section the choice $m_i = 1$. The resulting polynomials are called Hermitian interpolation polynomials.

2.4.3 Hermitian interpolation

If besides the value of a function we also know the value of its derivatives at certain points, we may use this data to construct an approximation polynomial of higher degree.

For instance assume that the function values and the value of the first derivative at two different points are known. Then we have in fact 4 independent data items and we may construct a 3-rd degree polynomial using this data.

So assume we have been given

$$\begin{array}{ll} (x_0, f(x_0)) & (x_1, f(x_1)) \,, \\ (x_0, f'(x_0)) & (x_1, f'(x_1)) \,. \end{array}$$

The third degree polynomial p_3 has to satisfy:

$$\begin{aligned} p_3(x_i) &= f(x_i) \qquad i = 0,1\,, \\ p_3^{'}(x_i) &= f^{'}(x_i) \qquad i = 0,1\,. \end{aligned}$$

In the same vein as the Lagrangian interpolation polynomials we may write this polynomial as:

$$p_3(x) = \sum_{i=0}^{1} [f(x_i)H_{i1}(x) + f^{'}(x_i)\hat{H}_{i1}(x)]\,,$$

with

$$\begin{aligned} H_{i1}(x_j) &= \delta_{ij}\,, \qquad \hat{H}_{i1}(x_j) = 0\,, \\ H_{i1}^{'}(x_j) &= 0\,, \qquad \hat{H}_{i1}^{'}(x_j) = \delta_{ij}\,. \end{aligned}$$

Polynomials based both on given function values and given derivatives are called *Hermitian interpolation polynomials*.

The general expression for Hermitian interpolation polynomials containing function values and first derivatives is:

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j)H_{jn}(x) + \sum_{j=0}^{n} f^{'}(x_j)\hat{H}_{jn}(x)\,,$$

in which

$$H_{jn}(x) = [1 - 2(x - x_j)L_{jn}^{'}(x_j)]L_{jn}^2(x),$$

and

$$\hat{H}_{jn}(x) = (x - x_j)L_{jn}^2(x)\,.$$

We may show the correctness of this polynomial as follows: H_{jn} and $\hat{H}_{jn}$ are polynomials of degree $2n+1$. In node x_k we have $H_{jn}(x_k) = 0$ if $k \neq j$. If $k = j$:

$$H_{jn}(x_j) = [(1 - 2(x_j - x_j)L_{jn}^{'}(x_j)]L_{kn}^2(x_j) = 1\,.$$

From $\hat{H}_{jn}(x_k) = 0$ we have $H_{2n+1}(x_j) = f(x_j)$.

To show that the derivatives coincide at the node points we remark that $H_{jn}^{'}$ is given by:

$$\begin{aligned} H_{jn}^{'}(x) \quad = \quad & -2L_{jn}^{'}(x_j)L_{jn}^2(x) + \\ & [1 - 2(x - x_j)L_{jn}^{'}(x_j)]2L_{jn}(x)L_{jn}^{'}(x)\,. \end{aligned}$$

By substitution we may simply check that $H_{jn}^{'}(x_k) = 0$ for $k = 0, 1, \ldots, n$. The derivative of $\hat{H}_{jn}$ is given by

$$\hat{H}_{jn}(x) = L_{jn}^2(x) + 2(x - x_j)L_{jn}(x)L_{jn}^{'}(x)\,.$$

Hence $\hat{H}_{jn}^{'}(x_k) = \delta_{jk}$ and therefore $H_{2n+1}^{'}(x_j) = f^{'}(x_j)$.

Theorem 2.4.1 *If $f \in C^{2n+2}[a,b]$ then there exists a $\xi \in (x_0, \ldots, x_n, x)$ such that*

$$f(x) - H_{2n+1}(x) = \frac{(x - x_0)^2 \ldots (x - x_n)^2}{(2n+2)!} f^{(2n+2)}(\xi)\,.$$

Proof:
The proof of this theorem is analogous to that of Theorem 2.2.1. The auxiliary function is chosen as follows:

$$\varphi(t) = f(t) - H_{2n+1}(t) - \frac{(t-x_0)^2 \ldots (t-x_n)^2}{(x-x_0)^2 \ldots (x-x_n)^2}[f(x) - H_{2n+1}(x)] \; .$$

It can be proved that $\varphi^{'}(t)$ has $2n+2$ different zeros in $(x_0, \ldots, x_n, x)$.

⊠

We shall clarify the choice for Hermitian interpolation instead of Lagrangian in the following two examples.

Example 2.4.2 (seismic waves)

In oil exploration one often uses seismic waves. A simple model for wave propagation is described by the following set of ordinary differential equations:

$$\begin{aligned} \frac{dx}{dt} &= c \sin\theta \; , \\ \frac{dz}{dt} &= -c \cos\theta \; , \\ \frac{d\theta}{dt} &= -\frac{dc}{dz} \sin\theta \; . \end{aligned}$$

The position is denoted by (x,z) while θ is the angle between wave front and x-axis. We suppose that the propagation speed c depends on the vertical position only and that it is known at a finite number of measuring points. In solving this system we need an approximation of $c(z)$ in the intermediate points. Piecewise linear interpolation in each interval has the consequence, that the derivative $\frac{dc}{dz}$ does not exist in the nodes. This may lead to large errors in the solution. If both c and $\frac{dc}{dz}$ are known in the nodes we may use the third degree Hermitian interpolation in each interval. Then the first derivative exists also at the nodes.

Example 2.4.3 (visualization)

Suppose a finite number of points of a figure is known and we want to draw a smooth curve through them. Piecewise linear interpolation has sharp angles in the graph in the nodes and this often leads to an unrealistic representation. We get a better result using Hermitian interpolation.

Suppose the graph of the function $f(x) = \frac{1}{1+x^3}$, $x \in [0,4]$ is used to visualize half of a symmetric hill. To save memory and computation visualization programs utilize simple building blocks. Assume that third degree polynomials are such simple blocks. In Figure 2.4 the graph has been approximated with several interpolation polynomials. The resulting figure using piecewise linear polynomials does not resemble a hill even remotely. Hermitian interpolation on the same nodes gives a much better result. This however is not an honest comparison, because third degree interpolation uses twice as many datapoints. Therefore we also consider the third degree Lagrangian interpolation on the intervals [0,2] and [2,4]. Note that on the interval [2,4] the function and the polynomial coincide. However, due to the large jump in the derivative at $x = 2$ this result is also unusable.

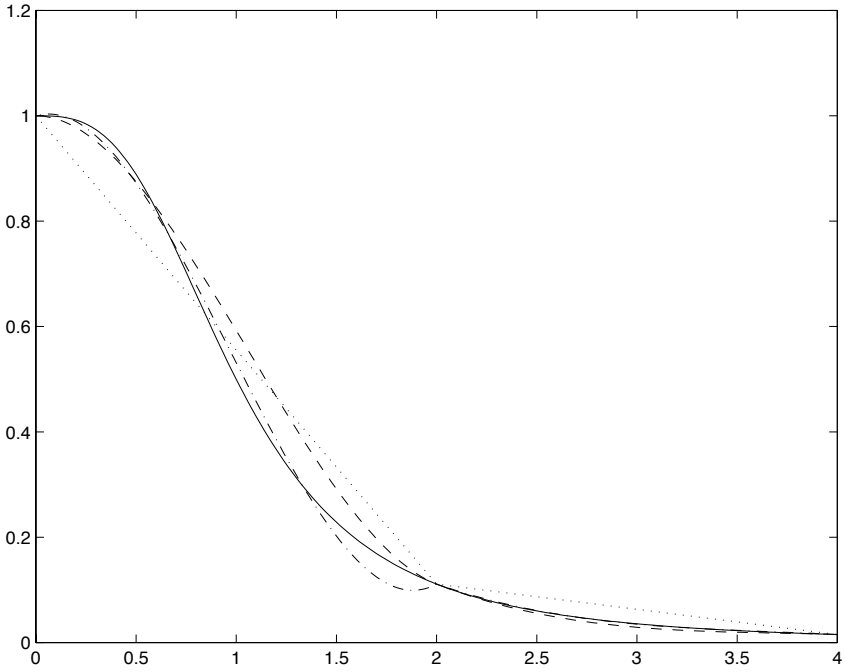

Figure 2.4 Interpolation van $\frac{1}{1+x^3}$. *—— function,* $\cdots$ *piecewise linear interpolation,* $- - -$ *Hermitian interpolation,* $- \cdot - \cdot -$ *piecewise third degree Lagrangian interpolation.*

2.5 Interpolation with splines

In the previous sections we saw that approximation of a function on an interval may lead to several problems. Often it is better to divide the interpolation interval into subintervals and to construct an interpolation polynomial on each subinterval. One problem with this approach is the lack of smoothness at the interface of two subintervals. Hermitian interpolation is a remedy for this problem, but for this to work one has to know the derivative at the interface. If the data is gathered by measurement, the derivative at the nodes are often unknown. A way to circumvent this problem is to use splines.

A *spline* is a piecewise polynomial that is connected smoothly in the nodes. We shall only consider first and third order splines.

Definition
For $f \in C[a,b]$ the interpolation spline s of degree 1 is a function $s \in C[a,b]$ such that for a partitioning $a = x_0 < x, \ldots < x_n = b$ of the interval $[a,b]$, s is linear on each subinterval $[x_i, x_{i+1}]$ and $s(x_i) = f(x_i)$.

Note that an interpolation spline of degree 1 consists just of piecewise linear interpolation polynomials. But a spline of degree three has several additional properties.
A spline of degree 3, also called a cubic spline, consists of piecewise third degree polynomials. In the nodes the value is equal to the function value and the first and second

derivative are continuous.

Definition
For $f \in C[a,b]$ the interpolating spline s of degree 3 has the following properties:

a. s is a third degree polynomial s_j on each subinterval $[x_j, x_{j+1}]$, $j = 0, \ldots, n-1$,

b. $s(x_j) = f(x_j)$, $j = 0, \ldots, n$,

c. $s_j(x_{j+1}) = s_{j+1}(x_{j+1})$, $j = 0, \ldots, n-2$,
$s_j'(x_{j+1}) = s_{j+1}'(x_{j+1})$, $j = 0, \ldots, n-2$,
$s_j''(x_{j+1}) = s_{j+1}''(x_{j+1})$, $j = 0, \ldots, n-2$,

d. $s_0''(x_0) = s_{n-1}''(x_n) = 0$.

Note that $s \in C^2[a,b]$. The conditions under d could be replaced with other conditions.

We shall demonstrate how to determine such an interpolating spline. We express s_j as:

$$s_j(x) = a_j(x-x_j)^3 + b_j(x-x_j)^2 + c_j(x-x_j) + d_j \,. \tag{2.2}$$

Define $h_j = x_{j+1} - x_j$ and $f_j = f(x_j)$. From b we have

$$d_j = f_j. \tag{2.3}$$

Next we use the various conditions from c:

1. $\mathbf{s_j''(x_{j+1}) = s_{j+1}''(x_{j+1})}$
From (2.2) we have $s_j''(x) = 6a_j(x-x_j) + 2b_j$. On substitution we obtain:

$$\begin{array}{llll} s_j''(x_j) = 2b_j & \text{for} & x = x_j & \text{and} \\ s_j''(x_{j+1}) = 6a_jh_j + 2b_j & \text{for} & x = x_{j+1} & . \end{array}$$

We may now express a_j in b_j:

$$a_j = \frac{1}{3h_j}(b_{j+1} - b_j) \,. \tag{2.4}$$

2. $\mathbf{s_j(x_{j+1}) = s_{j+1}(x_{j+1})}$
From this $a_jh_j^3 + b_jh_j^2 + c_jh_j + d_j = f_{j+1}$. Substitution of a_j, b_j and d_j yields the following expression for c_j:

$$c_j = \frac{f_{j+1} - f_j}{h_j} - h_j\frac{2b_j + b_{j+1}}{3} \,. \tag{2.5}$$

3. $\mathbf{s_j'(x_{j+1}) = s_{j+1}'(x_{j+1})}$

   Therefore $3a_j h_j^2 + 2b_j h_j + c_j = c_{j+1}$. On substitution:

$$h_j(b_{j+1} - b_j) + 2b_j h_j + \frac{f_{j+1} - f_j}{h_j} - h_j \frac{2b_j + b_{j+1}}{3}$$
$$= \frac{f_{j+2} - f_{j+1}}{h_{j+1}} - h_{j+1} \frac{2b_{j+1} + b_{j+2}}{3} .$$

   After simplification:

$$h_j b_j + 2(h_j + h_{j+1})b_{j+1} + h_{j+1} b_{j+2} = 3\left(\frac{f_{j+2} - f_{j+1}}{h_{j+1}} - \frac{f_{j+1} - f_j}{h_j}\right) .$$

   This relation holds from $j = 0$ up to $j = n-2$. So we get $n-1$ equations in $n+1$ unknowns $b_0, \ldots, b_n$. From d we determine $b_0 = b_n = 0$. The resulting system then becomes:

$$\begin{pmatrix} 2(h_0+h_1) & h_1 & & \\ h_1 & 2(h_1+h_2) & h_2 & \\ \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix} =$$

$$\begin{pmatrix} 3\left(\frac{f_2-f_1}{h_1} - \frac{f_1-f_0}{h_0}\right) \\ \vdots \\ \vdots \\ 3\left(\frac{f_n-f_{n-1}}{h_{n-1}} - \frac{f_{n-1}-f_{n-2}}{h_{n-2}}\right) \end{pmatrix}$$

   From this system we may calculate the values b_j. From this we may determine a_j from (2.4) and c_j from (2.5).

Example 2.5.1 (visualization)

We consider the same example as in the previous section. In Figure 2.5 we have used a cubic spline. We divided the interval into 6 subintervals. Note that the interpolation quality is better than that of Hermitian interpolation. Another advantage is that function values at the nodes give sufficient information already. It is not necessary to know the value of the derivatives at the nodes.

2.6 Summary

In this chapter the following subjects have been discussed:

- Linear interpolation
- Lagrangian interpolation

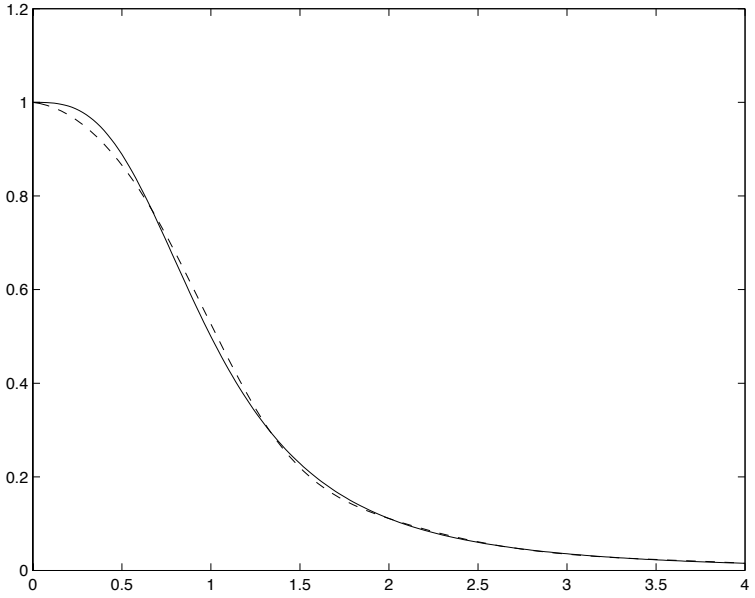

Figure 2.5 Interpolation of $\frac{1}{1+x^3}$*: — function, - - cubic spline.*

- Interpolation with derivatives
    - Taylor polynomial
    - Interpolation in general
    - Hermitian interpolation
- Spline interpolation.

2.7 Exercises

1. Determine the second degree Lagrangian polynomial of $f(x) = 1/x$ using nodes $x_0 = 2, x_1 = 2.5$ and $x_2 = 4$. Approximate $f(3)$ with this polynomial.
2. Determine $s(\frac{1}{2})$, in which s is the cubic spline with nodes 0, 1 and 2 for the function $f(x) = x$. Do this also for $f(x) = x^2$.