Contents I

Introduction

Goal: representation and evaluation of functions on a computer.

Typical applications:

- Reconstruction of a functional relationship between "measured function values", evaluation for additional arguments
- More efficient evaluation of very expensive functions
- Representation of fonts (2D), structures (3D) in a computer
- Data compression
- Solving differential and integral equations

Introduction

We restrict ourselves to functions of *one* variable, e.g.:

$$f \in C^r[a, b]$$

This is an *infinite dimensional* function space. Computers operate on function classes which are determined through *finitely many* parameters (not necessarily linear subspaces), e.g.:

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n \qquad \text{(polynomals)}$$

$$r(x) = \frac{a_0 + a_1 x + \cdots + a_n x^n}{b_0 + b_1 x + \cdots + b_m x^m} \qquad \text{(rational functions)}$$

$$t(x) = \frac{1}{2} a_0 + \sum_{k=1}^{n} (a_k \cos(kx) + b_k \sin(kx)) \qquad \text{(trigonom. polynomials)}$$

$$e(x) = \sum_{k=1}^{n} a_k \exp(b_k x) \qquad \text{(exponential sum)}$$

Approximation

Basic task of *approximation:*

Given a set of functions P (polynomials, rational functions, ...) and a function f (e.g., $f \in C[a,b]$), find $g \in P$, so that the error $f - g$ is minimized in a suitable fashion.

Examples:

$$\left(\int_a^b (f-g)^2 \, dx \right)^{1/2} \to \min \qquad \text{(2-norm)}$$

$$\max_{a \le x \le b} |f(x) - g(x)| \to \min \qquad (\infty\text{-norm})$$

$$\max_{i \in \{0,\ldots,n\}} |f(x_i) - g(x_i)| \to \min \qquad \text{for } a \le x_i \le b,\ i = 0, \ldots, n$$

Interpolation

Interpolation is a special case of approximation, where g is determined by

$$g(x_i) = y_i := f(x_i) \quad i = 0, \ldots, n$$

Special properties of interpolation:

- The error $f - g$ is only considered on a finite set of *nodes* $x_i, i = 0, \ldots, n$.
- In these finitely many points the deviation must be zero, not just minimal in some weaker sense.

Polynomial Interpolation

Let P_n the set of polynomials on $\mathbb{R}$ of degree smaller or equal $n \in \mathbb{N}_0$:

$$P_n := \{p(x) = \sum_{i=0}^{n} a_i x^i \mid a_i \in \mathbb{R}\}$$

P_n is an $n+1$-dimensional vector space.
The *monomials* $1, x, x^2, \ldots, x^n$ are a basis of P_n.

For given $n+1$ (distinct) nodes $x_0, x_1, \ldots, x_n$ the task of interpolation is

$$\text{Find } p \in P_n: \quad p(x_i) = y_i := f(x_i), \quad i = 0, \ldots, n$$

Polynomial Interpolation

This is equivalent to the linear system

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}}_{=:V[x_0,\ldots,x_n]} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

The matrix $V[x_0, \ldots, x_n]$

- is called *Vandermonde matrix*
- is regular iff all x_i are distinct
- leads to a very ill-conditioned map from values y_i to coefficients a_i
- requires computational effort in $\mathcal{O}(n^3)$ when solving the system

Polynomial Interpolation

Problem:
Assembling the Vandermonde matrix $V[x_0, \ldots, x_n]$ and then solving the linear system is not a good approach due to severe ill-conditioning and high associated cost.

Are there better and simpler approaches?

The problem is caused by the monomial basis $1, x, x^2, \ldots, x^n$, which leads to a particularly unfortunate formulation of the interpolation task. We are going to consider possible alternatives.

Langrange Interpolation

Definition 16 (Lagrange Polynomials)

For $n+1$ distinct nodes $x_i, i = 0, \ldots, n$, define the so-called *Lagrange polynomials*:

$$L_i^{(n)}(x) := \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \quad i = 0, \ldots, n \quad (n+1 \text{ polynomials})$$

The $L_i^{(n)}$ have degree n,

$$L_i^{(n)}(x_k) = \delta_{ik} = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases}$$

holds, and the $L_i^{(n)}$ are a basis of P_n.

Existence and Uniqueness

Using the Lagrange basis, the coefficients a_i are simply the prescribed values of the nodes: $a_i = y_i$. Solving the interpolation problem is therefore trivial in this basis.

Theorem 17 (Uniqueness of Interpolating Polynomial)

For $n+1$ distinct nodes $x_0, \ldots, x_n$ there is exactly one polynomial p of degree n with

$$p(x_i) = y_i \quad i = 0, \ldots, n,\ y_i \in \mathbb{R}$$

Therefore, the interpolation problem is solvable, and its solution is unique.

Newton Representation

Disadvantage of Lagrange polynomials:
Adding a node changes all previous basis polynomials, making this approach unsuitable for "incremental" construction of interpolation polynomials.

In this context, the *Newton representation* with basis polynomials

$$N_0(x) = 1; \quad i = 1, \ldots, n\colon N_i(x) = \prod_{j=0}^{i-1}(x - x_j)$$

is a better choice.

N_i always has degree i,

$$\forall k < i\colon N_i(x_k) = 0$$

holds, and the polynomials $N_0, \ldots, N_n$ are a basis of P_n.

Staggered Computations

In x_0 all the Newton basis polynomials but N_0 are zero, in x_1 all but N_0 and N_1, and so on.

The interpolation task

$$p(x_k) = \sum_{i=0}^{n} a_i N_i(x_k) = \sum_{i=0}^{k} a_i N_i(x_k) = y_k \quad k = 0, \ldots, n$$

leads to the following staggered computations:

$$a_0 = y_0; \quad k = 1, \ldots, n\colon\ a_k = \left[y_k - \sum_{i=0}^{k-1} a_i N_i(x_k) \right] / N_k(x_k)$$

Background

Polynomial interpolation in the language of linear algebra:

- The monomial basis $x^i, i = 0, \ldots, n$ is trivial to construct, but it leads to a dense and very ill-conditioned matrix $V[x_0, \ldots, x_n]$ (Vandermonde matrix).
- The Lagrange basis $L_i^{(n)}, i = 0, \ldots, n$, instead leads to an identity matrix, and therefore the solution is trivial. But an extension of the set of nodes changes all basis functions.
- The Newton basis $N_i, i = 0, \ldots, n$, in turn, results in a lower triangular matrix. The scheme of staggered computations corresponds to forward substitution. Solving requires more effort than with the Langrange basis, but additional nodes simply add additional rows to the matrix.

Divided Differences

Theorem 18 (Divided Differences)

The divided differences are recursively defined as

$$\forall i = 0, \ldots, n: \quad y[x_i] := y_i \quad \text{(values in nodes)}$$

$$\forall k = 1, \ldots, n - i:$$

$$y[x_i, \ldots, x_{i+k}] := \frac{y[x_{i+1}, \ldots, x_{i+k}] - y[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$

Then

$$p(x) = \sum_{i=0}^{n} y[x_0, \ldots, x_i] N_i(x)$$

holds.

Divided Differences

The divided differences are usually arranged in a tableau:

$$\begin{array}{lllll}
y_0 = y[x_0] & \to & y[x_0, x_1] & \to & y[x_0, x_1, x_2] \quad \to \quad y[x_0, x_1, x_2, x_3] \\
 & \nearrow & & \nearrow & \nearrow \\
y_1 = y[x_1] & \to & y[x_1, x_2] & \to & y[x_1, x_2, x_3] \\
 & \nearrow & & \nearrow & \\
y_2 = y[x_2] & \to & y[x_2, x_3] & & \\
 & \nearrow & & & \\
y_3 = y[x_3] & & & &
\end{array}$$

Then, the first row contains the desired coefficients $a_i, i = 0, \ldots, n$ of the basis polynomials. This form of computation doesn't need the values of the $N_i(x_k)$ and additionally is more stable.

Example I

Example 19

Consider the following pairs of nodes and values:

$$(x_0 = 0, y_0 = 1), \quad (x_1 = 1, y_1 = 4), \quad (x_2 = 2, y_2 = 3)$$

The monomial basis results in the system of equations

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$$

with solution $[1, 5, -2]^T$, therefore

$$p(x) = 1 + 5 \cdot x - 2 \cdot x^2$$

Example II

Example 19

The Lagrange basis leads to

$$\begin{aligned} p(x) &= 1 \cdot L_0^{(2)}(x) + 4 \cdot L_1^{(2)}(x) + 3 \cdot L_2^{(2)}(x) \\ &= 1 \cdot \frac{x-1}{0-1} \cdot \frac{x-2}{0-2} + 4 \cdot \frac{x-0}{1-0} \cdot \frac{x-2}{1-2} + 3 \cdot \frac{x-0}{2-0} \cdot \frac{x-1}{2-1} \\ &= \frac{1}{2} \cdot (x-1) \cdot (x-2) - 4 \cdot x \cdot (x-2) + \frac{3}{2} \cdot x \cdot (x-1) \end{aligned}$$

The individual basis polynomials can be precomputed for a given set of nodes $x_0, \ldots, x_n$, afterwards additional interpolation tasks on the same set of nodes are trivial to solve.

Example III

Example 19

For the Newton basis we obtain the tableau

$$\begin{array}{ccccc} y_0 = a_0 = 1 & \to & a_1 = \frac{4-1}{1-0} = 3 & \to & a_2 = \frac{(-1)-3}{2-0} = -2 \\ & \nearrow & & \nearrow & \\ y_1 = 4 & \to & \frac{3-4}{2-1} = -1 & & \\ & \nearrow & & & \\ y_2 = 3 & & & & \end{array}$$

and therefore

$$\begin{aligned} p(x) &= 1 \cdot N_0(x) + 3 \cdot N_1(x) - 2 \cdot N_2(x) \\ &= 1 + 3 \cdot x - 2 \cdot x \cdot (x-1) \end{aligned}$$

Here it is easy to add an additional node if required.

Evaluating Polynomials

The standard algorithm for the evaluation of a polymial of the form

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + \cdots + a_n x^n$$

at a point x is the *Horner scheme*

$$b_n := a_n; \quad k = n-1, \ldots, 0 \colon b_k := a_k + x \cdot b_{k+1}; \quad p(x) = b_0 \,,$$

since it is particularly efficient and stable.

For a polynomial in Newton representation

$$p(x) = \sum_{i=0}^{n} a_i N_i(x) = a_0 + a_1 N_1(x) + \cdots + a_n N_n(x)$$

this scheme leads to the recursion

$$b_n := a_n; \quad k = n-1, \ldots, 0 \colon b_k := a_k + (x - x_k) \cdot b_{k+1}; \quad p(x) = b_0$$

Interpolation Error

Let $y_i = f(x_i), i = 0, \ldots, n$, the evaluation of a function f in $n+1$ distinct nodes, and $p(x)$ the resulting interpolation polynomial of degree n.

By construction, the difference

$$e(x) := f(x) - p(x)$$

fulfills the condition

$$e(x_i) = 0 \quad \text{für} \quad i = 0, \ldots, n$$

Question: How large can this difference become at other locations?

Interpolation Error

Theorem 20 (Interpolation Error)

Let $f(x)$ $n+1$-times continuously differentiable on $[a,b]$ and

$$a \leq x_0 < x_1 < \cdots < x_n \leq b.$$

Then there is for each $x \in [a,b]$ an $\xi_x \in \overline{(x_0,\ldots,x_n,x)}$ (smallest interval containing all nodes), so that

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j)$$

Remarks

For the special case of *equidistant* nodes, i.e.,

$$x_{k+1} - x_k = h \quad \text{für} \quad k = 0, \ldots, n-1,$$

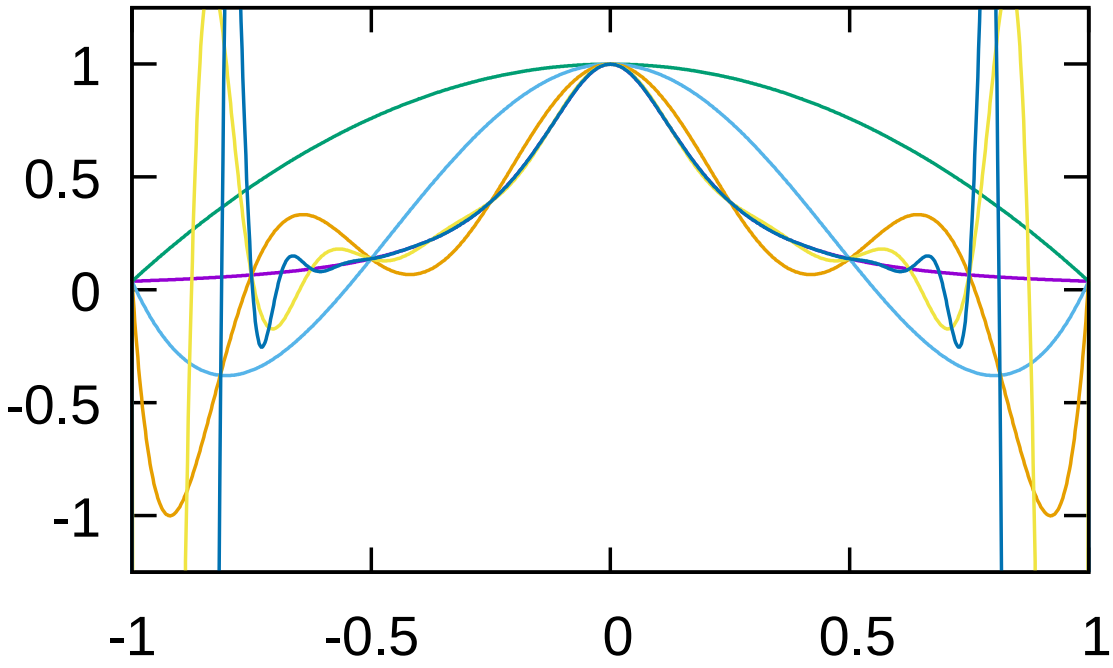
we therefore have

$$|f(x) - p(x)| \leq |f^{(n+1)}(\xi_x)| \cdot h^{n+1},$$

and for $|f^{(n+1)}|$ bounded and $n \to \infty$ it follows $|f(x) - p(x)| \to 0$.

Unfortunately, the higher derivatives of functions, even of simple ones, are often *not* bounded for $n \to \infty$, but grow very fast instead.

Runge's Counter Example



Polynomial interpolation of Runge's function $f(x) = (1 + 25x^2)^{-1}$ with equidistant nodes (3, 5, 9, 17 resp. 33 node/value pairs).
The minima / maxima of the last two polynomials are $-14.35/1.40$ resp. $-5059/2.05$ (!).

Remarks

Remark 21

According to the *Weierstraß Approximation Theorem*, any function in $C^0([a, b])$ can be approximated uniformly by polynomials.
The phenomena we observe are no contradiction, since:

- The approximation need not be based on interpolation (the proof uses Bernstein polynomials).
- Using non-equidistant nodes one can already achieve significantly improved results (if one knows how to choose these non-equidistant nodes...).

Remark 22

In general "methods of higher (polynomial) order" require sufficient differentiability.

Condition Analysis

With $p(x; y)$ the interpolation polynomial to the values $(y_0, \ldots, y_n)^T$ at *fixed* nodes $(x_0, \ldots, x_n)^T$, we have

$$\begin{aligned} p(x; y + \Delta y) - p(x; y) &= \sum_{i=0}^{n} (y_i + \Delta y_i) L_i^{(n)}(x) - \sum_{i=0}^{n} y_i L_i^{(n)}(x) \\ &= \sum_{i=0}^{n} \Delta y_i L_i^{(n)}(x) \end{aligned}$$

This implies

$$\frac{p(x; y + \Delta y) - p(x; y)}{p(x; y)} = \sum_{i=0}^{n} \frac{L_i^{(n)}(x) y_i}{p(x; y)} \cdot \frac{\Delta y_i}{y_i}$$

For large n, $L_i^{(n)}$ can become very large, then the interpolation task is ill-conditioned!

Contents I

Numerical Differentiation

Problem:
Compute the derivative (of some order n) of a function that is given as a table or implemented as a function (in the computer science sense of the word).

Idea:
Assemble interpolation polynomial for certain nodes, differentiate it and evaluate result to obtain (approximation of) derivative.

We assume order of derivative = degree of polynomial.

Numerical Differentiation

The Lagrange polynomials are

$$L_i^{(n)}(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} = \underbrace{\prod_{j \neq i} (x_i - x_j)^{-1}}_{=:\lambda_i \in \mathbb{R}} \cdot x^n + \alpha_{n-1} x^{n-1} + \cdots + \alpha_0,$$

therefore taking the n-th derivative produces

$$\frac{d^n}{dx^n} L_i^{(n)}(x) = n! \cdot \lambda_i$$

which gives us the n-th derivative of an interpolation polynomial of degree n:

$$\frac{d^n}{dx^n} \left(\sum_{i=0}^{n} y_i L_i^{(n)}(x) \right) = n! \cdot \sum_{i=0}^{n} y_i \lambda_i \quad \text{(independent of } x\text{)}$$

Numerical Differentiation

We have the following statement about the resulting error:

Theorem 23
Let $f \in C^n([a,b])$ and $a = x_0 < x_1 < \cdots < x_n = b$. Then there is $\xi \in (a,b)$, so that

$$f^{(n)}(\xi) = n! \cdot \sum_{i=0}^{n} y_i \lambda_i$$

Therefore, the derivative from the interpolation polynomial coincides in at least one point with the true derivative of f.

Numerical Differentiation

For equidistant nodes, $x_i = x_0 + ih, 0 \le i \le n$, there is an explicit formula based on the node values y_i:

$$f^{(n)}(x) \approx h^{-n} \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} y_i, \quad \text{e.g.,} \quad f^{(1)}(x) \approx \frac{y_1 - y_0}{h},$$

$$f^{(2)}(x) \approx \frac{y_2 - 2y_1 + y_0}{h^2}, \quad f^{(3)}(x) \approx \frac{y_3 - 3y_2 + 3y_1 - y_0}{h^3}$$

Based on Taylor expansion one can show

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2) \qquad \text{for} \quad f \in C^3,$$

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \mathcal{O}(h^2) \quad \text{for} \quad f \in C^4$$

Numerical Differentiation

These approximations of function derivatives are called *centered difference quotients*.

One can also place the nodes off-center to obtain the *forward difference quotient*

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h) \qquad \text{for} \quad f \in C^2$$

and *backward difference quotient*

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h) \qquad \text{for} \quad f \in C^2$$

Note that forward and backward difference quotients have lower approximation order than centered variants.

Difference quotients play an important role in the derivation of methods for differential equations.

Extrapolation to the Limit I

Let some quantity $a(h)$ be computable for $h > 0$, but not for $h = 0$. We are interested in computing

$$a(0) = \lim_{h \to 0} a(h)$$

with good accuracy.

Example 24

Possible applications:

1. L'Hospital's rule:

$$a(0) = \lim_{h \to 0} \frac{\cos(x) - 1}{\sin(x)} (= 0)$$

Extrapolation to the Limit II

Example 24

2. Numerical Differentiation:

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

(small h cause cancellation)

3. Numerical Integration:

$$\int_a^b f(x) = \lim_{N \to \infty} \sum_{i=1}^{n} N^{-1} f\left(a + (i - \frac{1}{2}) \frac{b-a}{N}\right)$$

(set $h := N^{-1}$)

Extrapolation to the Limit III

Example 24

4. Numerical solution of initial value problem

$$y'(t) = f(t, y(t)) \quad \text{on} \quad [0, T]; \quad y(0) = y_0$$

Set

$$h = N^{-1}; \quad y_n = y_{n-1} + h \cdot f(t, y_n - 1); \quad y(T) \approx y_N$$

Here $h \to 0$ is equivalent to $N \to \infty$ and therefore increasing computational cost.

Central Idea of Extrapolation

Idea of extrapolation:

For $h_0 > h_1 > \cdots > h_n > 0$ construct interpolation polynomial

$$p(h_i) = a(h_i) \quad i = 0, \ldots, n$$

and compute

$$a(0) \approx p(0)$$

(Extrapolation instead of interpolation, since $0 \notin [h_n, \ldots, h_0]$)

Example I

Example 25

For $a(h) = (\cos(h) - 1) \cdot (\sin(h))^{-1}$ we have

$$\begin{aligned} h_0 = 1/8: &\quad a(h_0) = -6.258151 \cdot 10^{-2} \\ h_1 = 1/16: &\quad a(h_1) = -3.126018 \cdot 10^{-2} \\ h_2 = 1/32: &\quad a(h_2) = -1.562627 \cdot 10^{-2} \end{aligned}$$

(i.e., $a(h)$ is directly proptional to h), and with extrapolation using p_2 of degree 2:

$$a(0) \approx p_2(0) = -1.02 \cdot 10^{-5}$$

which is significantly better than the initial approximations or a possible direct evaluation for $h \ll 1$ (cancellation)!

Example II

Example 25

Why does this work so well?

Let $h_i = h \cdot r^i$ with $r < 1$ (geometric distribution), e.g., $r = 1/2$, and let p the interpolation polynomial of a to the nodes h_i. Then we have

$$|p(0) - a(0)| \leq \|V^{-T}\|_\infty |a^{(n+1)}(\xi)| \frac{h^{n+1}}{(n+1)!}(1 + r^{n+1})$$

for the extrapolation error, with Vandermonde matrix V and $\xi \in (0, h)$, as long as a is sufficiently differentiable.

Extrapolation in the Case of Derivatives

The Taylor expansion of a at zero (Mclaurin expansion) is crucial. For the usual difference quotient for the second derivative we obtain

$$\begin{aligned} a(h) &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \\ &= f''(x) + \frac{h^2}{2 \cdot 4!} f^{(4)}(x) + \cdots + \frac{h^{2n}}{2 \cdot (2n+2)!} f^{(2n+2)}(x) \\ &\qquad + \frac{h^{2n+2}}{2 \cdot (2n+4)!} [f^{(2n+4)}(\xi_+) + f^{(2n+4)}(\xi_-)] \\ &= p_x(h^2) + \mathcal{O}(h^{2(n+1)}) \end{aligned}$$

This means one gains two powers of h per evaluation (if f is sufficiently smooth)!

Newton-Cotes Formulas

The *Newton-Cotes formulas* are interpolatory quadrature (integration) formulas.

Idea: Construct the interpolation polynomial p of a function f for certain nodes and evaluate the integral of p exactly to approximate the integral of f.

Formally: nodes and values $(x_i, f(x_i)), i = 0, \ldots, n$, Lagrange representation:

$$p_n(x) = \sum_{i=0}^{n} f(x_i) L_i^{(n)}(x), \quad L_i^{(n)}(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

and therefore

$$I_{[a,b]}(f) \approx I_{[a,b]}^{(n)}(f) = \int_a^b p_n(x)\, dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_i^{(n)}(x)\, dx$$

Order of Quadrature

Definition 26 (Order of Quadrature)

A quadrature formula $I^{(n)}(f)$ has at least order m, if it is able to integrate polynomials of degree $m-1$ exactly.

For example, a second order formula integrates linear functions exactly.

The Newton-Cotes formulas use polynomial interpolation and therefore they have at least order $n+1$ for $n+1$ nodes. But there are other formulas that can achieve even higher orders with the same number of nodes.

Closed and Open Formulas

The Newton-Cotes formulas use *equidistant* nodes. There are two variants:

Closed formulas:
The interval bounds a and b are nodes, i.e.,

$$x_i = a + iH, \quad i = 0, \ldots, n, \quad \text{with } H = \frac{b-a}{n}$$

Open formulas:
The bounds a and b aren't nodes, i.e.,

$$x_i = a + (i+1)H, \quad i = 0, \ldots, n, \quad \text{with } H = \frac{b-a}{n+2}$$

Examples

Closed formulas for $n = 1, 2, 3$ and $H = (b - a)/n$:

The trapezoidal rule:

$$I^{(1)}(f) = \frac{b-a}{2} \cdot [f(a) + f(b)]$$

The Simpson rule resp. Kepler's barrel rule:

$$I^{(2)}(f) = \frac{b-a}{6} \cdot [f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)]$$

The 3/8 rule resp. "pulcherrima":

$$I^{(3)}(f) = \frac{b-a}{8} \cdot [f(a) + 3f(a+H) + 3f(b-H) + f(b)]$$

Examples

Open formulas for $n = 0, 1, 2$ and $H = (b - a)/(n + 2)$:

The midpoint rule:

$$I^{(0)}(f) = (b - a) \cdot f\left(\frac{a + b}{2}\right)$$

The second open rule (no special name):

$$I^{(1)}(f) = \frac{b - a}{2} \cdot [f(a + H) + f(b - H)]$$

The third open rule (also no special name):

$$I^{(2)}(f) = \frac{b - a}{3} \cdot \left[2f(a) - f\left(\frac{a + b}{2}\right) + 2f(b)\right]$$

Remarks

Remark 27

From $n = 7$ on for closed rules, resp. from $n = 2$ on for open rules, negative weights appear in the sums. This is detrimental, because:

- Strictly non-negative functions f can have $I^{(n)}(f) < 0$ (solute concentration, mass conservation,...).
- There is increased risk of cancellation.
- Condition can become worse, while it is bounded for strictly positive weights.

Estimates for Remainder Terms I

Theorem 28 (Remainder Terms)

The resulting error can be estimated as follows:

1. Trapezoidal rule: $n = 1$, order 2, we have

$$I(f) - \frac{b-a}{2} \cdot [f(a) + f(b)] = -\frac{(b-a)^3}{12} f''(\xi), \quad \xi \in [a, b]$$

for $f \in C^2([a, b])$. This means polynomials up to degree 1 are integrated exactly, because for those $f''(x) = 0$ holds on $[a, b]$.

In general: the order of the odd formulas is the number of nodes, while the order of the even formulas is one higher.

Estimates for Remainder Terms II

Theorem 28 (Remainder Terms)

2. Simpson rule: n = 2, order 4, for $f \in C^4([a,b])$ we have

$$I(f) - \frac{b-a}{6} \cdot \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] = -\frac{(b-a)^5}{2880} f^{(4)}(\xi)$$

3. Midpoint rule: n = 0, order 2, for $f \in C^2([a,b])$ we have

$$I(f) - (b-a) \cdot f\left(\frac{a+b}{2}\right) = \frac{(b-a)^3}{24} f''(\xi)$$

so "half the error of trapezoidal rule" at just one function evaluation!

Summed Quadrature Rules

Increasing the polynomial degree doesn't make much sense, since

- negative weights appear early
- Lagrange interpolation with equidistant nodes doesn't converge pointwise
- f has to be sufficiently regular for the estimates to hold

Idea of *summed quadrature rules*:

- Subdivide interval $[a, b]$ into N smaller intervals

$$[x_i, x_{i+1}], \quad x_i = a + ih, \ i = 0, \ldots, N-1, \ h = \frac{b-a}{N}$$

- Apply one of the above formulas on each subinterval and sum the results.

Examples

For N subintervals of stepsize h we arrive at:

The summed trapezoidal rule:

$$I_h^{(1)}(f) = \sum_{i=0}^{N-1} \frac{h}{2}[f(x_i) + f(x_{i+1})] = h \cdot \left[\frac{f(a)}{2} + \sum_{i=1}^{N-1} f(x_i) + \frac{f(b)}{2} \right]$$

The summed Simpson rule:

$$I_h^{(2)}(f) = \frac{h}{3} \cdot \left[\frac{f(a)}{2} + \sum_{i=1}^{N-1} f(x_i) + 2 \sum_{i=1}^{N-1} f\left(\frac{x_i + x_{i+1}}{2} \right) + \frac{f(b)}{2} \right]$$

The summed midpoint rule:

$$I_h^{(0)}(f) = h \cdot \sum_{i=0}^{N-1} f\left(\frac{x_i + x_{i+1}}{2} \right)$$

Contents I

Vectors and Matrices I

A *vector* $v \in \mathbb{R}^n$ is a finite sequence of real numbers:

$$v = [v_1, v_2, \ldots, v_n]^T, \quad v_i \in \mathbb{R}$$

A *matrix* $A \in \mathbb{R}^{n \times m}$ is defined similarly, but uses two independent indices i and j, one for columns and one for rows:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}, \quad a_{ij} \in \mathbb{R}$$

Each matrix $A \in \mathbb{R}^{n \times m}$ corresponds to a linear mapping $\varphi_A \colon \mathbb{R}^m \to \mathbb{R}^n$ given by

$$\varphi_A(v) = A \cdot v, \quad v \in \mathbb{R}^m$$

Vectors and Matrices II

For a sufficiently regular scalar function $f(x), x = [x_1, \ldots, x_m]$, we have the following special vectors and matrices:

$$\nabla f := \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_m}\right]^T \qquad (\textit{gradient})$$

$$\mathrm{H}_f = \nabla^2 f := \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{ij} \qquad (\textit{Hessian})$$

For vector-valued functions $f(x) = [f_1(x), \ldots, f_n(x)]^T$, the gradient generalizes to the *Jacobian*:

$$\mathrm{J}_f = \left[\frac{\partial f_i}{\partial x_j}\right]_{ij}$$

If $f = \varphi_A$ is a linear function, then $\mathrm{J}_f = A$, i.e., the constituent matrix A and the Jacobian coincide.

Natural Matrix Norms

Definition 29 (Associated Matrix Norm)

Let $\|\cdot\|$ an arbitrary vector norm on $\mathbb{R}^n$. Then

$$\|A\| := \sup_{x\in\mathbb{R}^n\setminus\{0\}} \frac{\|Ax\|}{\|x\|} = \sup_{x\in\mathbb{R}^n,\|x\|=1} \|Ax\|$$

is called the *matrix norm associated with* $\|\cdot\|$, or *natural matrix norm*. It is compatible with the matrix norm, i.e.,

$$\|Ax\| \leq \|A\| \cdot \|x\| \quad A \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n,$$

and submultiplicative, i.e.,

$$\|AB\| \leq \|A\| \cdot \|B\| \quad A, B \in \mathbb{R}^{n\times n}$$

(compare with triangle inequality / subadditivity).

Linear and Nonlinear Systems of Equations

Many important problems, e.g., the solution of ordinary and partial differential equations, can be framed as solving a given linear system of equations:

$$A \cdot x = b \quad \text{resp.} \quad A \cdot x - b = 0$$

or nonlinear system of equations:

$$F(x) = 0$$

where F is a possibly vector-valued function, A is a matrix, and x and b are vectors of the right dimensions.

Perturbation Theorem

Theorem 30 (Perturbation Theorem)

Let $A \in \mathbb{R}^{n\times n}$ regular and $\|\Delta A\| \leq \|A^{-1}\|^{-1}$. Then $\widetilde{A} = A + \Delta A$ is also regular, and for the relative error of the perturbed system

$$(A + \Delta A) \cdot (x + \Delta x) = b + \Delta b$$

the equation

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\operatorname{cond}(A)}{1 - \operatorname{cond}(A) \cdot \frac{\|\Delta A\|}{\|A\|}} \cdot \left\{ \frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right\}$$

holds, where $\operatorname{cond}(A) := \|A\| \cdot \|A^{-1}\|$ is the *condition number* of the matrix A. For the special case $\Delta A = 0$ we have

$$\frac{\|\Delta x\|}{\|x\|} \leq \operatorname{cond}(A) \cdot \frac{\|\Delta b\|}{\|b\|}.$$

Triangular Systems

Let $A \in \mathbb{K}^{n \times n}$ an upper triangular matrix:

$$\begin{aligned} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n &= b_1 \\ a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n &= b_2 \\ \ddots \quad \vdots \quad & \vdots \\ a_{nn} \cdot x_n &= b_n \end{aligned}$$

This system permits a unique solution iff $a_{ii} \neq 0, i = 1, \ldots, n$.

Because of the simple structure this can be solved using "backward substitution".

Triangular Systems

Solution using backward substitution:

$$\begin{aligned} x_n &= b_n / a_{nn} \\ x_{n-1} &= \left(b_{n-1} - a_{(n-1)n} \cdot x_n\right) / a_{(n-1)(n-1)} \\ &\vdots \\ x_i &= \left(b_i - \sum_{k=i+1}^{n} a_{ik} \cdot x_k\right) / a_{ii} \end{aligned}$$

Required number of operations:

$$N_\Delta(n) = \sum_{i=0}^{n-1} (2i+1) = n^2$$

(Of course there is an analogous "forward substitution" for upper triangular matrices.)

Direct Methods for Linear Systems

Let $A \in \mathbb{K}^{n \times n}$ regular, but with arbitrary structure.

Goal: Transform A into (upper) triangular form, then use backward substitution.

This can be done using:

- exchange of two equations / rows
- addition of a multiple of one equation to another

This is a standard technique known as *Gauss elimination*.

Gauss Elimination

Perform the following steps until an upper triangular matrix is obtained, starting with $k = 1$:

1. For $i > k$, define $l_{ik} = a_{ik} \cdot a_{kk}^{-1}$.
2. For $i > k$, set
$$a_{ij} \longleftarrow a_{ij} - l_{ik} a_{kj}$$
(subtract a multiple of the k-th row to eliminate the first k entries of the i-th row).
3. Increase k by one: $k \longleftarrow k + 1$.
4. Repeat.

After (at most) $n - 1$ loop iterations the matrix has become upper right triangular.

Cost of Gauss Elimination

Lemma 31

The cost of transforming A into an upper right triangular matrix by Gauss elimination is

$$N_{\text{Gauß}}(n) = \frac{2}{3}n^3 + \mathcal{O}(n^2)$$

Since the cost for backward substitution is negligible ($N_\Delta(n) = n^2$), this is also the cost for solving a linear equation system using Gauss elimination.

A Note on Stability

The classic Gauss elimination is *unstable* for general matrices (because we divide by diagonal elements of the original matrix, which can become arbitrarily small).

The algorithm can be made significantly more stable through a process called *row pivotisation*. In each iteration, we search for the largest subdiagonal element in the k-th column and swap its row with the k-th row, remembering resulting row permutations.

Total pivotisation instead searches for the largest element in the lower right corner of the matrix, and employs both row and column permutations. This is more expensive, but leads to further improvements in terms of stability.

LU Decomposition I

Storing the factors l_{ik} of the elimination steps is a good idea:

Theorem 32 (LU Decomposition)

Let $A \in \mathbb{R}^{n\times n}$ regular, then there exists a decomposition $PA = LU$, where

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{11} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n1} & \cdots & l_{n(n-1)} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \ddots & u_{2n} \\ \vdots & \ddots & \ddots & u_{(n-1)n} \\ 0 & \cdots & 0 & u_{nn} \end{bmatrix},$$

and P is a permutation matrix. For $P = I$ this decomposition is unique.

LU Decomposition II

Solving a linear system via LU decomposition:

1. For given A, compute LU decomposition and matrix P
2. For given b, calculate $b' = Pb$
3. Solve triangular system $Ly = b'$ (forward substitution)
4. Solve triangular system $Rx = y$ (backward substitution)

LU decomposition is equivalent to Gauss elimination, and therefore has the same cost $N_{LU}(n) = \frac{2}{3}n^3 + \mathcal{O}(n)$.

Important difference:
The LU decomposition can be reused for new righthand sides $A\widetilde{x} = \widetilde{b}$, while Gauss elimination has to start over from the beginning!

Symmetric Positive Definite Matrices

Theorem 33

For a symmetric positive definite (s.p.d.) matrix $A \in \mathbb{R}^{n\times n}$ the LU decomposition is always stable, even *without* pivotisation. The equation

$$a_{ii}^{(k)} \geq \lambda_{\min}(A), \quad k \leq i \leq n$$

holds for the diagonal elements, where $\lambda_{\min}(A)$ is the smallest eigenvalue of A.

The symmetric structure of the matrix can be used to reduce the cost of LU decomposition.

Cholesky Decomposition

With $D = \mathrm{diag}(R)$ we have

$$A = LD(D^{-1}R) = LDR \quad \text{with} \quad R := D^{-1}U,$$

and because of symmetry $R = L^T$, therefore $A = LDL^T$. Since all diagonal elements of D are positive, the matrix $D^{1/2}$ with

$$(D^{1/2})_{ii} = d_{ii}^{1/2}, \quad (D^{1/2})_{ij} = 0 \text{ for } i \neq j$$

is well-defined, and

$$A = LD^{1/2} \cdot D^{1/2}L^T = \widetilde{L}\widetilde{L}^T \quad \text{with} \quad \widetilde{L} := LD^{1/2}$$

holds.

This special form of the decomposition is called *Cholesky decomposition*, it has half the cost of the general version.

Iterative Methods for Linear Systems I

We consider a second approach for the solution of

$$Ax = b, \quad A \in \mathbb{R}^{n\times n} \text{ regular}, \quad b \in \mathbb{R}^n.$$

Definition 34 (Sparse Matrices)

A sequence of matrices $\{A^{(n)} \mid n \in \mathbb{N}\}$ is called *sparse*, iff

$$|\{a_{ij}^{(n)} \mid a_{ij}^{(n)} \neq 0\}| =: \mathrm{nnz}(A^{(n)}) = \mathcal{O}(n)$$

(nnz = "number of non-zeros").

Because of "fill in" Gauss elimination is often unsuited for sparse matrices, and: for large systems the cost in $\mathcal{O}(n^3)$ makes the solution intractable.

Iterative Methods for Linear Systems II

We have

$$\text{Solving } Ax = b \iff \text{“root search” for } f(x) := b - Ax = 0.$$

With given matrix C, define the iteration

$$\begin{aligned} x^{(t+1)} &= g(x^{(t)}) = x^{(t)} + C^{-1} f(x^{(t)}) \\ &= x^{(t)} + C^{-1}(b - Ax^{(t)}) \\ &= \underbrace{(I - C^{-1}A)}_{=:B} x^{(t)} + C^{-1} b \end{aligned}$$

with “iteration matrix” B. The choice $C = A$ would be optimal in theory, but that requires solving the problem itself.

$\Longrightarrow$ look for easily invertible C “similar” to A

Iterative Methods for Linear Systems III

For the solution $x := A^{-1}b$,

$$g(x) := (I - C^{-1}A)x + C^{-1}b = x - (C^{-1}A) \cdot (A^{-1}b) + C^{-1}b = x$$

holds, therefore x is a fixpoint of g.

The Lipschitz constant of the function g fulfills

$$\|g(x) - g(y)\| = \|B(x - y)\| \leq \|B\| \cdot \|x - y\|,$$

i.e., if $\|B\| < 1$ (for suitable matrix norm $\|\cdot\|$), then g is a contraction on $\mathbb{R}^n$, and repeated application of g defines a sequence that converges to the solution (consequence of Banach's fixpoint theorem).

Examples for Iterative Methods

Decompose $A = L + D + U$ with L strict lower triangular matrix, D diagonal matrix and U strict upper triangular matrix.

Jacobi method:
Set $C = D$, i.e.,

$$x^{(t+1)} = x^{(t)} + D^{-1}(b - Ax^{(t)})$$

Gauß–Seidel method:
Set $C = L + D$, i.e., (forward substitution)

$$x^{(t+1)} = x^{(t)} + (L + D)^{-1}(b - Ax^{(t)})$$

Such iterative methods typically converge only for special classes of matrices (since we need $\|B\| < 1$).

Convergence of Jacobi Method

A matrix is called *strictly diagonally dominant*, iff

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

Theorem 35
The Jacobi method converges for strictly diagonally dominant matrices.

There are many similar statements for symmetric positive definite matrices, weakly diagonally dominant matrices, so-called M-matrices, ...

Costs

Let $\alpha(n)$ be the cost for one iteration, typically with $\alpha(n) = \mathcal{O}(n)$. Since

$$\|x^{(t)} - x\| \leq \|B\|^t \|x^{(0)} - x\|,$$

a total of $t \geq \frac{\log(\epsilon)}{\log(\|B\|)}$ iterations are necessary for a reduction of the error by a factor $\epsilon \ll 1$, leading to a total cost of

$$T_{\text{fix}}(n) = \frac{\log(\epsilon)}{\log(\|B\|)} \alpha(n).$$

Problem: high costs if $\|B\|$ is close to one, $\|B\|$ is problem dependent and often grows with n.

Conjugate Gradients

For symmetric positive definite matrices A one can instead use the method of Conjugate Gradients. It uses an initial guess x_0, the initial residuum $r_0 := b - Ax_0$, and $d_0 := r_0$ to iteratively compute

$$\alpha_t = \frac{r_t^T r_t}{d_t^T A d_t}$$
$$x_{t+1} = x_t + \alpha_t d_t$$
$$r_{t+1} = r_t - \alpha_t A d_t$$
$$\beta_t = \frac{r_{t+1}^T r_{t+1}}{r_t^T r_t}$$
$$d_{t+1} = r_{t+1} + \beta_t d_t$$

- The CG method converges in at most n steps in exact arithmetic.
- For $n \gg 1$ it can be used as an iterative method, and often displays good convergence properties after the first few steps.

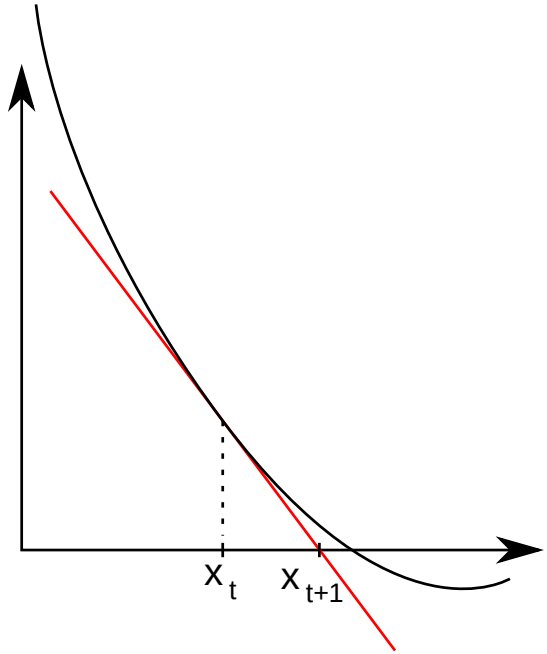
Newton's Method

Let f a differentiable function in one variable. For given x_t we have the "tangent"

$$T_t(x) = f'(x_t)(x - x_t) + f(x_t)$$

with root

$$T_t(x) = 0 \iff x = x_t - \frac{f(x_t)}{f'(x_t)}.$$



Using this root as an estimate for the root of f leads to the iteration

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}.$$

Obviously we need $|f'(x_t)| > 0$, i.e., we assume that the root of f is a *simple root*.

Newton's Method in Multiple Dimensions

Newton's method can be extended to systems $f : \mathbb{R}^n \to \mathbb{R}^n$:

Assume that the Taylor expansion of f exists:

$$f_i(x) = f_i(x_t+\Delta x) = f_i(x_t)+\sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(x_t)\Delta x_i+R_i(x_t,\Delta x) \quad i = 1,\ldots,n$$

or in vector notation

$$f(x_t + \Delta x) = f(x_t) + J_f(x_t)\Delta x + R(x_t, \Delta x)$$

with "Jacobian" matrix

$$[J_f(x_t)]_{ij} = \frac{\partial f_i}{\partial x_j}(x_t)$$

Newton's Method in Multiple Dimensions

Ignoring the remainder term is equivalent to "linearization of f".
Find approximate root of f:

$$f(x) \approx f(x_t) + J_f(x_t)\Delta x = 0 \iff \Delta x = -J_f^{-1}(x_t)f(x_t)$$

This leads to the iteration

$$x_{t+1} = x_t - J_f^{-1}(x_t)f(x_t)$$

Every single step requires the solution of a linear system based on the local Jacobian!

Convergence of Newton's Method I

Theorem 36 (Newton's Method)

Let $f \in C^2([a,b])$ have a root z in (a,b) (interior!), and let

$$m := \min_{a \le x \le b} |f'(x)| > 0, \quad M := \max_{a \le x \le b} |f''(x)|.$$

Let $\rho > 0$ chosen thus, that

$$q := \frac{M}{2m}\rho < 1, \quad K_\rho(z) := \{x \in \mathbb{R} \mid |x - z| \le \rho\} \subset [a,b]$$

Then for every initial value $x_0 \in K_\rho(z)$ the Newton iterations $x_t \in K_\rho(z)$ are defined and converge to the root z.

Convergence of Newton's Method II

Theorem 36 (Newton's Method)

Additionally, the a priori error erstimate

$$|x_t - z| \leq \frac{2m}{M} q^{(2^t)}, \quad t \in \mathbb{N}$$

and the a posteriori error estimate

$$|x_t - z| \leq m^{-1}|f(x_t)| \leq \frac{M}{2m}|x_t - x_{t-1}|^2, \quad t \in \mathbb{N}.$$

hold (a priori: only uses prerequisites, a posteriori: also uses iterations that were computed up to that point)

Example: Roots of Real Numbers

Example 37 (Computing Roots with Newton's Method)

Let $a > 0$ and $n \geq 1$. Solve $x^n = a$, i.e.,

$$f(x) = x^n - a = 0, \quad f'(x) = n \cdot x^{n-1}.$$

This leads to iteration

$$x_{t+1} = n^{-1} \cdot [(n-1) \cdot x_t + a \cdot x_t^{1-n}].$$

According to Thm. 36 this converges, if x_0 is close enough to $a^{1/n}$. However, in this special case it converges *globally*, i.e., for all $x_0 > 0$ (but not necessarily quadratically in the beginning).

Remarks I

Remark 38

- Newton's method converges only *locally*, i.e., when $|x_0 - z| \leq \rho$ ("basin of attraction"). Here ρ is normally not known and potentially very small.
- Newton's method exhibits quadratic convegence,

$$|x_t - z| \leq c \cdot |x_{t-1} - z|^2,$$

in contrast to alternatives like, e.g., bisection, which converges only linearly.

Remarks II

Remark 38

- Damped Newton's method:
  Convergence *outside* of the basin of attraction can be improved by setting

$$x_{t+1} = x_t - \lambda_t \frac{f(x_t)}{f'(x_t)}$$

  with the choice of some sequence $\lambda_t \in (0, 1]$ as "dampening strategy".

Remarks III

Remark 38

- Multiple roots:
  If z is a p-fold root, with $p > 1$, Newton's method will still converge, but only linearly. One can show that the modified iteration

$$x_{t+1} = x_t - p \cdot \frac{f(x_t)}{f'(x_t)}$$

  reestablishes quadratic convergence if p is known a priori.

Summary

In the last few days, we have discussed the fundamentals of the following topics:

1. Numerical programming in C++
2. Numbers and calculations with finite precision
3. Condition analysis of numerical problems and tasks
4. Error propagation and stability of numerical algorithms
5. Numerical differentiation and integration
6. Numerical solution of linear and nonlinear equation systems

The introduced concepts will form the basis of the lectures and exercises next week.