Fehleranalyse:

Es gilt für den Interpolationsfehler auf einem Intervall $[a,b]$ mit $x_i \in [a,b]$ $i = 0, \dots, n$

$$\max_{x \in [a,b]} \left| f(x) - P(f | x_0, \dots, x_n) \right| \leq$$

$$\max_{x \in [a,b]} \left| \prod_{j=0}^{n} (x - x_j) \right| \cdot \max_{x \in [a,b]} \frac{\left| f^{(n+1)}(t) \right|}{(n+1)!}$$

(7.13)

Bemerkung 7.14

Fehler (7.13) hängt offenbar von der „Verteilung" der Stützstellen ab (vg. des Terms $\prod_j (x - x_j)$ in 7.13). Eine „geschickte" Wahl für x_j liefern die Nullstellen der sog. Tschebyscheff-Polynome:

$$x_j = 1.5 + 0.5 \cos \frac{(2j+1)}{(2n+2)} \pi$$

$$j = 0, \dots, n$$

§ 7.2 Numerische Differenziation

Idee: Ersetze Ableitung einer Funktion
$f: [a,b] \to \mathbb{R}$ an einer Stelle $x \in [a,b]$
durch die Ableitung eines geeigneten
Interpolationspolynoms:
Wähle Stützstellen $x_0, \dots, x_n$ „nahe bei"
x und berechne für die n-te Ableitung

$$f^{(n)}(x) \approx P(f|x_0,\dots,x_n)^{(n)}(x)$$

$$= n! \, [x_0,\dots,x_n] f$$

(im Newton-Darstellung)

Das führt auf die sog. „finiten Differenzen"

$$f'(x) \approx [x_0, x_1] f = \frac{f(x_1) - f(x_0)}{h}, \quad x \in [x_0, x_1]$$

$$f''(x) \approx 2! [x_0, x_1, x_2] f =$$

$$\frac{f(x_2) - 2f(x_1) + f(x_0)}{h^2}$$

für äquidistante Stützstellen

$$x_j = x_0 + j \cdot h$$

mit der „Schrittweite" h.

Fehleranalyse:

Taylorentwicklung ergibt für

a) $x_0 = x$, $x_1 = x + h$:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(z)$$

für $z \in [x, x+h]$

(Vorwärtsdifferenz)

b) $x_0 = x-h, \quad x_1 = x, \quad x_2 = x+h$

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

$$- \frac{h^2}{12} f^{(4)}(z), \quad z \in [x-h, x+h]$$

(zentrale Differenz)

Problem in der „numer. Praxis“:

Auslöschung für kleine Schrittweiten h.

„moderne Numerik“ verwendet alternativ

„sog. automatische Differenziation (AD)“.

Numerical differentiation

Finite Differences

(folgende Folien von Sebastian Sager, Uni Magdeburg)

Approximate

$$\frac{\partial F(x)}{\partial x} = \frac{F(x \pm \triangle x) - F(x)}{\triangle x} + \mathcal{O}(\triangle x)$$

or

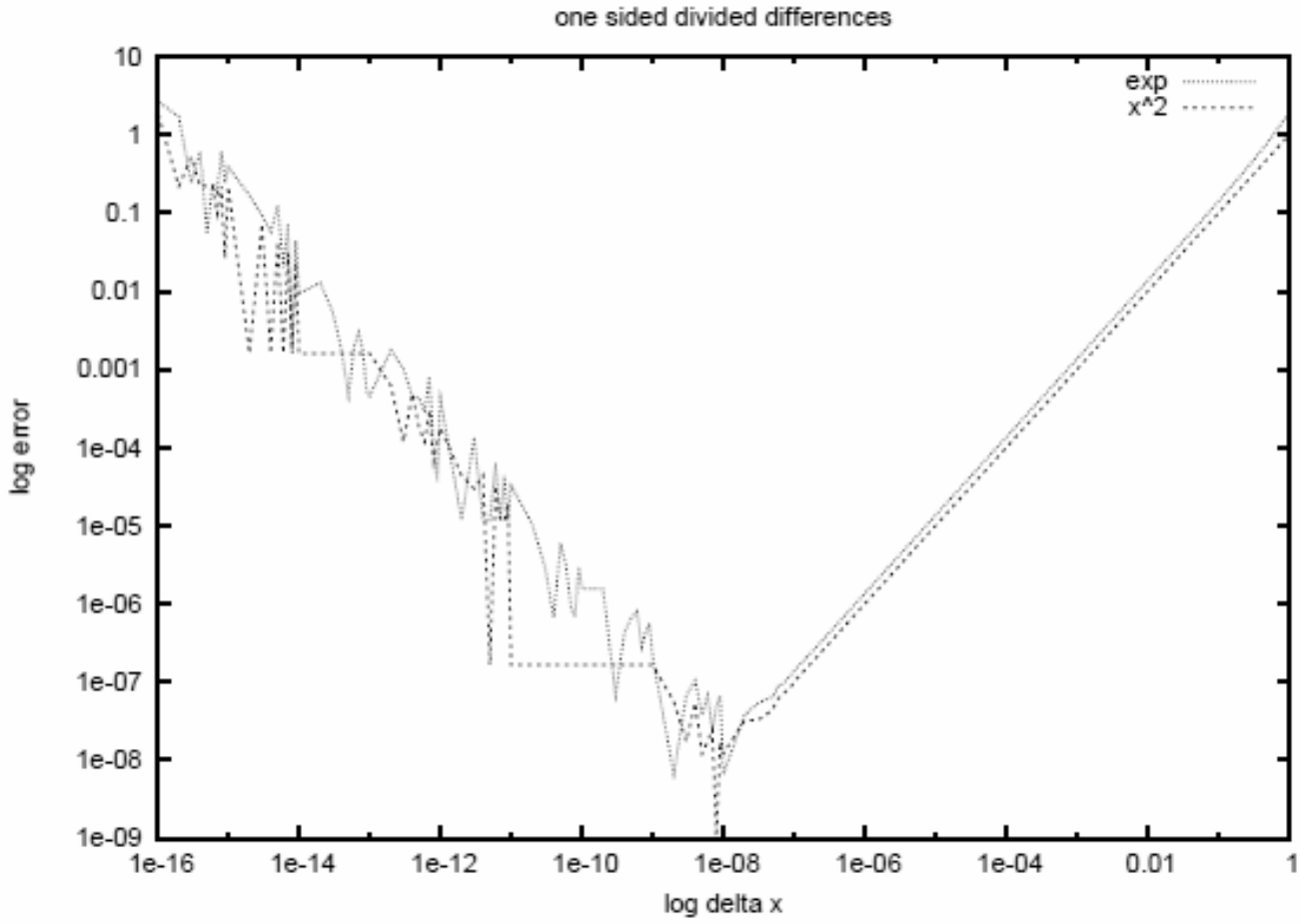
$$\frac{\partial F(x)}{\partial x} = \frac{F(x + \triangle x) - F(x - \triangle x)}{2\triangle x} + \mathcal{O}(\triangle x^2).$$

Choice of increment $\triangle x$ essential.

- $\triangle x$ too small: cancellation error
- $\triangle x$ too big: truncation error, higher order terms become significant

Numerical differentiation

Error analysis, $f'(x)$ evaluated at $x = 0$, machine precision `eps=1e-16`.



Even with optimal choice of $\triangle x$ half of significant digits lost.

But: Effort just 2 function evaluations $\Rightarrow$ cheap.

Symbolic Differentiation

e.g. using Maple, Mathematica

Calculate derivatives symbolically with computer-algebra tools.

- Advantages:
  - high precision (only round-off errors)
  - often easy to use
- Disadvantages:
  - resulting formulae for derivatives often very complex
    $\Rightarrow$ high computational costs
  - need analytical formulation of function to derive

Basic Ideas of Automatic Differentiation

$$y \equiv F(x) = \psi_l(\psi_{l-1}(\ldots(\psi_1(x))))$$

- Function $F(x)$ sequence of elementary operations like $+, -, \times, \frac{1}{x}, \exp, \sin, \cos, \ldots$
- Derivatives of elementary functions known:

$$(u+v)' = u'+v', \quad (uv)' = u'v+uv', \quad exp(u)' = exp(u)u', \ldots$$

- Apply chain-rule to calculate derivative of F
- Differentiation of computer code
- Derivatives evaluated with working precision
- Forward/Reverse mode: effort small multiple of evaluation

Basic Ideas of Automatic Differentiation

Notation

- $y \in \mathbb{R}^m$ „dependent variables“,
  $x \in \mathbb{R}^n$ „independent variables“.

- Introduce variables for intermediate values and rename linearly:

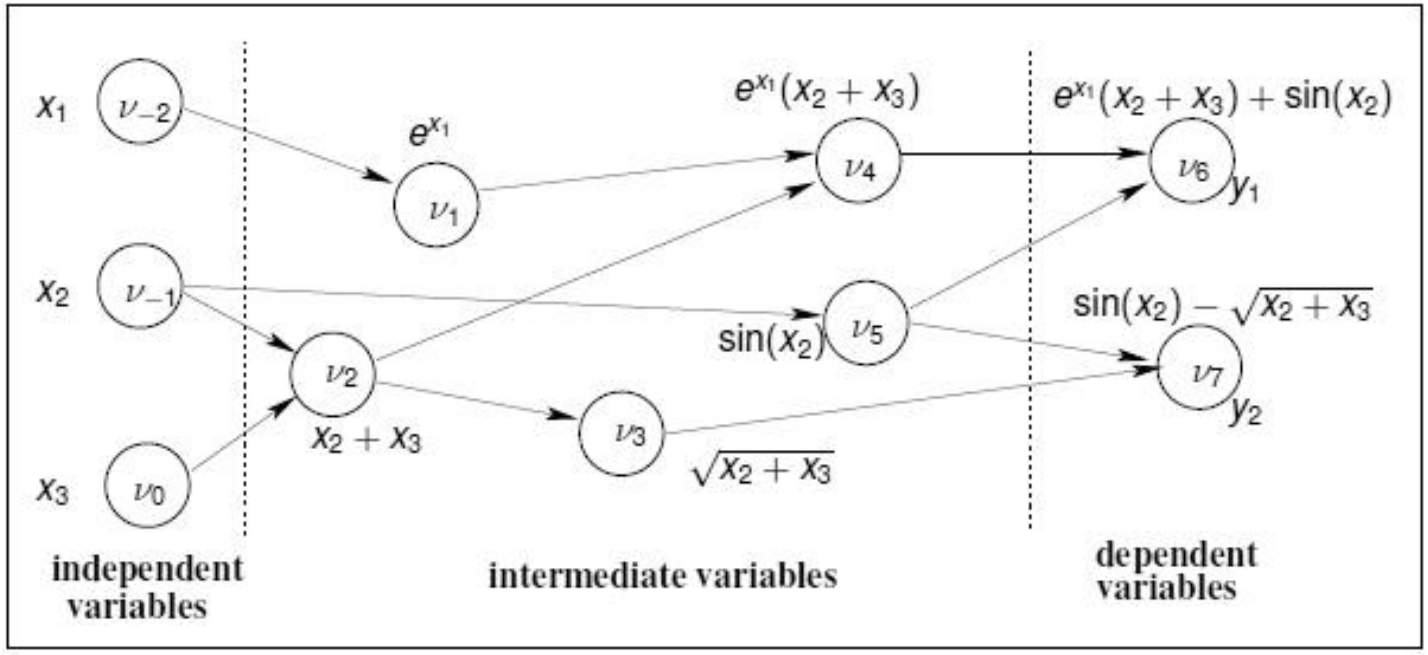
$$\begin{aligned} \nu_{i-n} &\equiv x_i, & 1 \le i \le n \\ \nu_i &\equiv \phi_i((\nu_j)_{j \prec i}), & 1 \le i \le l \\ y_i &:= \nu_{l-m+i}, & 1 \le i \le m \end{aligned}$$

$j \prec i$ indicates that ν_i depends directly on ν_j.

Basic Ideas of Automatic Differentiation

Computational graph

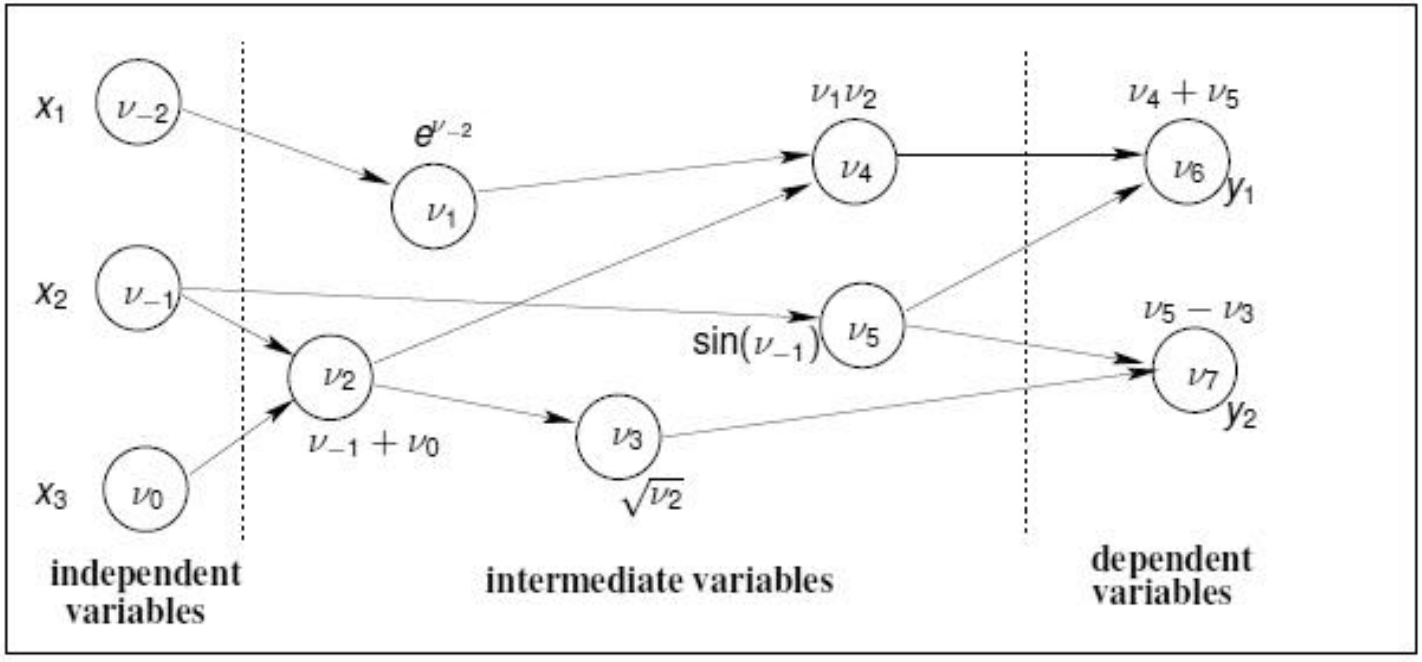
$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv F(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1}(x_2 + x_3) + \sin(x_2) \\ \sin(x_2) - \sqrt{x_2 + x_3} \end{pmatrix}$$



Basic Ideas of Automatic Differentiation

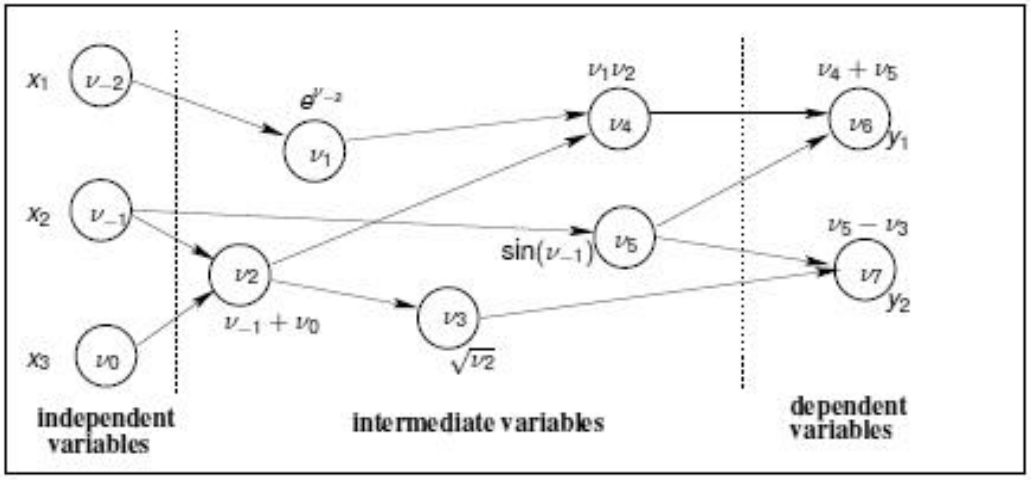
Computational graph

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv F(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1}(x_2 + x_3) + \sin(x_2) \\ \sin(x_2) - \sqrt{x_2 + x_3} \end{pmatrix}$$



Basic Ideas of Automatic Differentiation

Evaluation trace



$$
\begin{array}{lcl}
v_{-2} & \equiv & x_1 \\
v_{-1} & \equiv & x_2 \\
v_0 & \equiv & x_3 \\
\hline
v_1 & \equiv & e^{(v_{-2})} \\
v_2 & \equiv & v_{-1} + v_0 \\
v_3 & \equiv & \sqrt{v_2} \\
v_4 & \equiv & v_1 v_2 \\
v_5 & \equiv & \sin(v_{-1}) \\
v_6 & \equiv & v_4 + v_5 \\
v_7 & \equiv & v_5 - v_3 \\
\hline
y_1 & := & v_6 \\
y_2 & := & v_7
\end{array}
$$

Forward Mode

Generate directional derivative

$$\dot{y} = \frac{\partial F}{\partial x}\dot{x} = \dot{F}(x, \dot{x})$$

with direction $\dot{x} \in \mathbb{R}^n$ and $\dot{F}(x, \dot{x}) : \mathbb{R}^{2n} \mapsto \mathbb{R}^m$.

Algorithm:

$$\begin{aligned} \dot{\nu}_{1-n} &\equiv \dot{x}_i \\ \dot{\nu}_{i-n} &\equiv \sum_{j \prec i} \frac{\partial \phi_i}{\partial \nu_j} \dot{\nu}_j \\ \dot{y}_i &:= \dot{\nu}_{l-m+i} \end{aligned}$$

Forward Mode

Evaluation and Forward Trace

ν_{-2}	$\equiv$	x_1	$\dot{\nu}_{-2}$	$\equiv$	$\dot{x}_1$
ν_{-1}	$\equiv$	x_2	$\dot{\nu}_{-1}$	$\equiv$	$\dot{x}_2$
ν_0	$\equiv$	x_3	$\dot{\nu}_0$	$\equiv$	$\dot{x}_3$
ν_1	$\equiv$	e^{ν_0}	$\dot{\nu}_1$	$\equiv$	$\nu_1\dot{\nu}_0$
ν_2	$\equiv$	$\nu_{-1} + \nu_0$	$\dot{\nu}_2$	$\equiv$	$\dot{\nu}_{-1} + \dot{\nu}_0$
ν_3	$\equiv$	$\sqrt{\nu_2}$	$\dot{\nu}_3$	$\equiv$	$\frac{1}{2\nu_3}\dot{\nu}_2$
ν_4	$\equiv$	$\nu_1\nu_2$	$\dot{\nu}_4$	$\equiv$	$\dot{\nu}_1\nu_2 + \nu_1\dot{\nu}_2$
ν_5	$\equiv$	$\sin(\nu_{-1})$	$\dot{\nu}_5$	$\equiv$	$\cos(\nu_{-1})\dot{\nu}_{-1}$
ν_6	$\equiv$	$\nu_4 + \nu_5$	$\dot{\nu}_6$	$\equiv$	$\dot{\nu}_4 + \dot{\nu}_5$
ν_7	$\equiv$	$\nu_5 - \nu_3$	$\dot{\nu}_7$	$\equiv$	$\dot{\nu}_5 - \dot{\nu}_3$
y_1	$:=$	ν_6	$\dot{y}_1$	$:=$	$\dot{\nu}_6$
y_2	$:=$	ν_7	$\dot{y}_2$	$:=$	$\dot{\nu}_7$

Automatic Differentiation

Summary

- Differentiation of computer code.
- Derivatives evaluated with working precision (no truncation errors).
- Computational effort (p Number of directions $\dot{x}/\bar{y}$):
  - Forward Mode: $\leq c$-times function evaluation, $c \in [1 + p, 1 + 1.5p]$.
  - Reverse Mode: $\leq d$-times function evaluation, $d \in [1 + 2p, 1.5 + 2.5p]$.
    Needs to store intermediate values during function evaluation.

Automatic Differentiation

Implementations

Two main concepts:

- Operator overloading:
  - Redefine the basic operators $+, -, \times, \ldots$ to calculate derivatives in parallel with functional evaluation or tape function evaluation at runtime.
  - Tools: ADOL-C, FADBAD++, ...
- Sourcecode transformation:
  - Generate from sourcecode for function evaluation new sourcecode for evaluating the derivatives.
  - Tools: ADIFOR, ADIC, TAPENADE, TAF, ...

Literature

- Andreas Griewank & Andrea Walther. *Evaluating Derivatives*. SIAM, 2008.
- AutoDiff Homepage `www.autodiff.org`.

Kap. 8 Numerische Integration (Quadratur)

Sei $f : [a, b] \to \mathbb{R}$, stetig.

Aufgabe: Berechne $\int_a^b f(x)\,dx$ numerisch.

Idee: Ersetze Integral durch Summe über „Stützstellen"

$$t_j := a + jh,\ j = 0, \dots, n,\ h = \frac{b-a}{n}$$

und f durch „einfach" zu integrierende Funktionen $g_k : [t_{k-1}, t_k] \to \mathbb{R}$

$$\rightarrow \int_a^b f(x)\,dx = \sum_{k=1}^{n} \int_{t_{k-1}}^{t_k} f(x)\,dx$$

$$\approx \sum_{k=1}^{n} \int_{t_{k-1}}^{t_k} g_k(x)\,dx \qquad (8.1)$$

Trapezregel (8.2)

$$g_k(x) := \frac{x - t_{k-1}}{h} f(t_k) + \frac{t_k - x}{h} f(t_{k-1})$$

$$\text{also} \int_{t_{k-1}}^{t_k} g_k(x)\,dx = \frac{h}{2}\left[f(t_{k-1}) + f(t_k)\right]$$

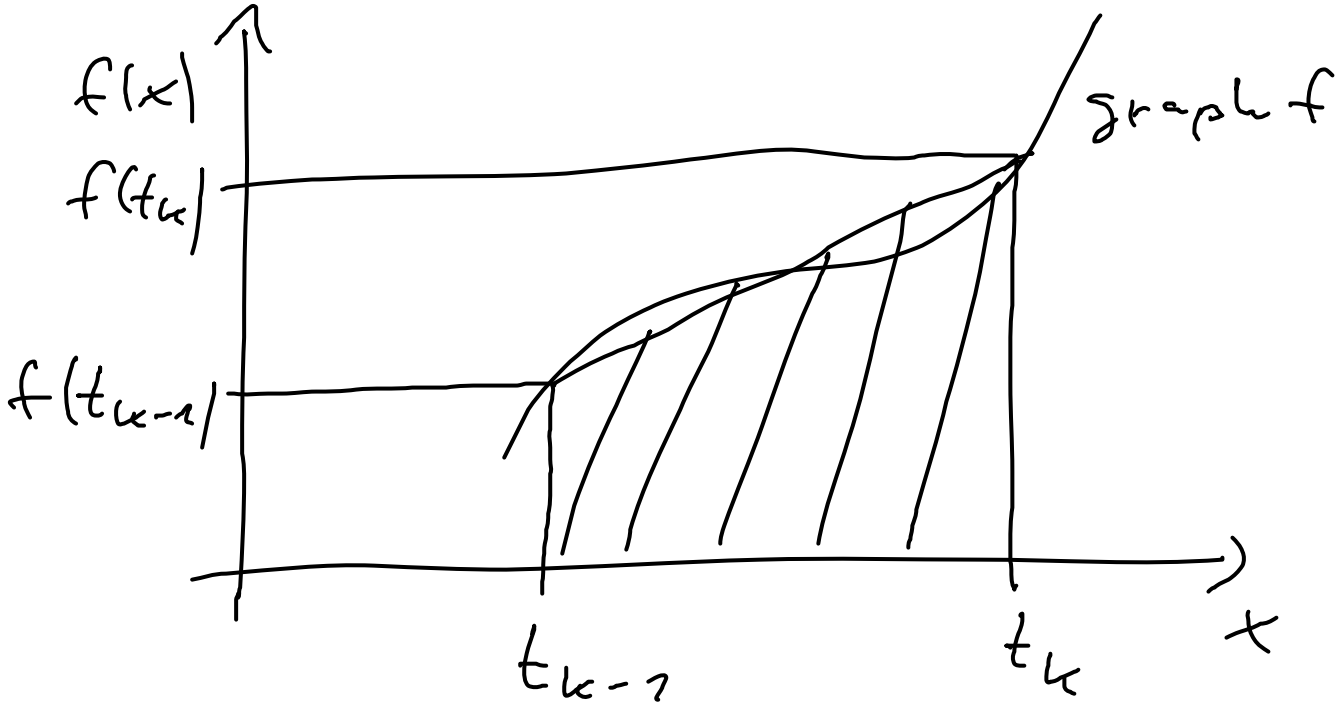


Abb. 8.3

(8.1) liefert mit (8.2) durch Summation

$$\int_a^b f(x)\,dx \approx T(h) := h \cdot \left[\frac{f(a)}{2} + f(t_1) + \ldots + f(t_{n-1}) + \frac{f(b)}{2} \right] \quad (8.4)$$

(summierte Trapezregel)

Lemma 8.5 (Fehler d. Trapezregel)

Sei $f \in C^2([t_{k-1}, t_k])$, also 2-mal stetig differenzierbar auf $[t_{k-1}, t_k]$.

Dann gilt:

$$\left| T(h) - \int_a^b f(x)\,dx \right| \leq \frac{h^3}{12} \cdot n \cdot \max_{x \in [a,b]} |f''(x)|$$

mit $nh = b - a$:

$$\left| T(h) - \int_a^b f(x)\,dx \right| \leq \frac{h^2}{12} (b-a) \cdot \max_{x \in [a,b]} |f''(x)| \quad (8.6)$$

Bemerkung 8.7

Wg. (8.6) konvergiert $T(h)$ für $h \to 0$ quadratisch gegen $\int_a^b f(x)\,dx$.

Fehler ist abhängig von der Krümmung von f, d.h. Trapezregel ist (trivialerweise) exakt für lineare Funktionen.

§ 8.1 Newton–Coates Formeln

Seien $x_i \in [c,d]$, $i = 0, \dots, m$, $x_i \neq x_j$ für $i \neq j$ gegeben und $P(f|x_0, \dots, x_m)$ das Interpolationspolynom (s Kap. 7).

Approximiere

$$\int_c^d f(x)\,dx \approx I_m(f) := \int_c^d P(f|x_0,\dots,x_m)(x)\,dx \qquad (8.8)$$

Bemerkung 8.9

$m=1$, $x_0=c$, $x_1=d$ in (8.8) liefert die Trapezregel.

Satz 8.10

Für jedes Polynom $Q \in \Pi_m$ gilt

$$I_m(Q) = \int_c^d Q(x)\,dx, \quad \text{d.h.}$$

die „Quadraturformel" (8.8) ist „exakt vom Grade m".

Beweis: über Eindeutigkeit des Interpolationspolynoms.

Aus der Darstellung des Lagrange-Interpolationspolynoms (im Satz 7.1) folgt ...