

Einführung in Computational Engineering

Grundlagen der Modellierung und Simulation



TECHNISCHE
UNIVERSITÄT
DARMSTADT

8. Vorlesung: Zeitkontinuierliche Modellierung und Simulation

2. Dezember 2013

Prof. Dr. Jan Peters

MOODLE
TEST IN
ZWEI
FOLIEN




produziert vom



- Bitte Kommentare bei Moodlefragen nach Vorlesen vermeiden! SCHWEIGEN!
OK!
- Bitte „Matrix“ nicht „Matrize“ sagen. *Von mir aus...*
- Keine Pause da „mein Bus um 16:05 geht und ich lieber früher aufhören möchte“ ... Umfrage: <https://moodle.informatik.tu-darmstadt.de/mod/choice/view.php?id=19383>
- Früheres Ende für Wiederholungen nutzen! Umfrage: <https://moodle.informatik.tu-darmstadt.de/mod/choice/view.php?id=19382>
- Downloadversion der Aufzeichnung in höherer Auflösung? *Geht technisch leider nicht ...*
- Überblicksfolie vor jeder Vorlesung. *Ja, mache ich in Robot Learning auch, werde ich ab nächster Vorlesung auch hier einführen.*

- Umfrage-Ergebnisse (Meise ergab 422 : 1 : 2)

Stimmabgaben

Abstimmoptionen	Teilnehmerzahl	Prozent der Teilnehmer/innen	Grafische Darst
Noch nicht abgestimmt	343	80,7%	
Ja, bitte lies die Fragen nicht mehr vor.	23	5,4%	
Nee, is' ok. Weitervorlesen!	59	13,9%	

- Neue Umfrage zu Übung und Vorlesung: <https://moodle.informatik.tu-darmstadt.de/mod/choice/view.php?id=19381>

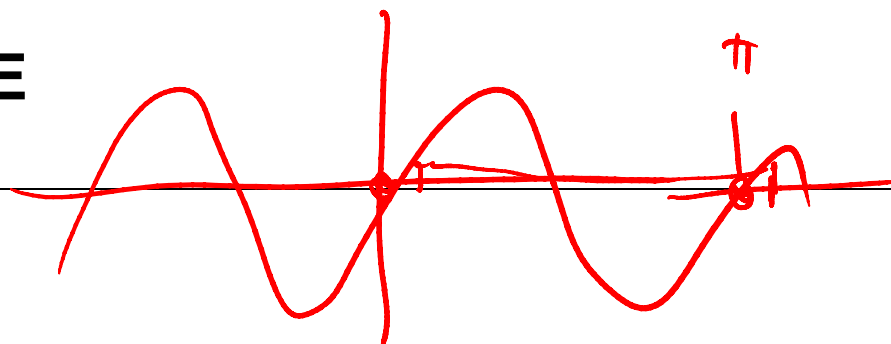
Überblick der Vorlesungsinhalte

1. Einführung
2. Diskrete Modellierung und Simulation
3. Zeitkontinuierliche Modellierung und Simulation
4. Teilschritte einer Simulationsstudie
5. Interpretation und Validierung
6. Modulare und objektorientierte Modellierung und Simulation
7. Parameteridentifikation von Modellen

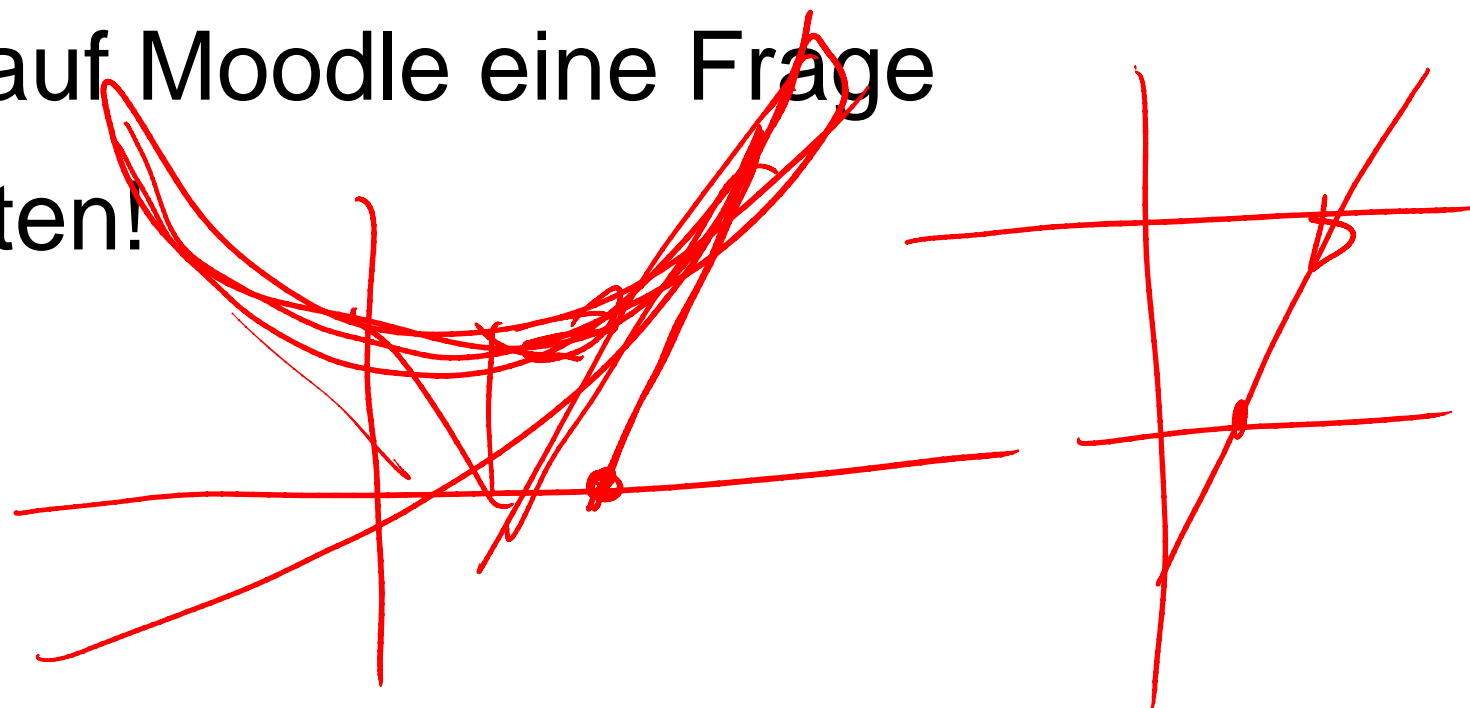
MOODLE FRAGE



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Bitte jetzt auf Moodle eine Frage
beantworten!





Grundlagen der Modellierung und Simulation

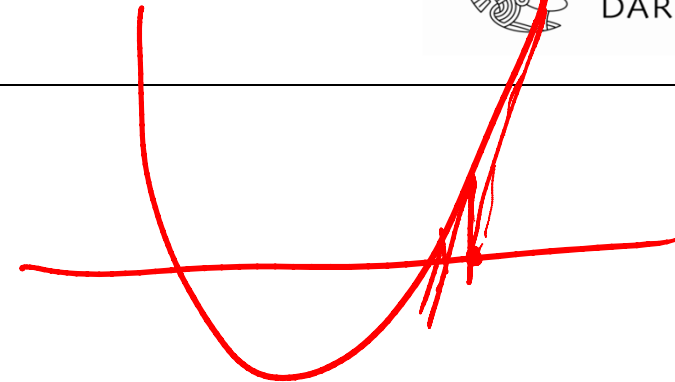
3. ZEITKONTINUIERLICHE MODELLIERUNG UND SIMULATION





Wiederholung: Newton-Verfahren

- Gesucht: x_s mit $f(x_s) = 0$ $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$
- Beginn mit einem Startvektor x_0



- Iterationsschritt ($i \rightarrow i + 1$)

- Berechnung von Funktion $f(x_i)$ und Jacobi-Matrix $\frac{\partial f}{\partial x}(x_i)$
- Berechnung der Korrektur Δx_i durch Lösung des linearen Gleichungssystems (LGS):

$$\frac{\partial f}{\partial x}(x_i) \cdot \Delta x_i = -f(x_i)$$

Handwritten red notes: $A \Delta x = b \Rightarrow x = A^{-1}b$ with arrows pointing from the matrix and vector in the equation above to the corresponding terms in the handwritten formula.

- Berechnung der neuen Näherung (evtl. mit Schrittweitensteuerung α_i)

$$x_{i+1} = x_i + \alpha_i \Delta x_i$$

Wiederholung: Newton-Verfahren



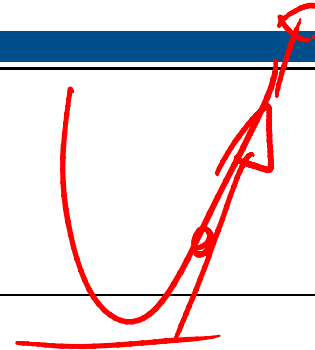
TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Terminierung, falls „nahe genug“ an der Lösung also „kein Fortschritt“ mehr gemacht wird
- Sonst wird der nächste Iterationsschritt ausgeführt



3.5.11 Jacobi-Matrix

$$J = \frac{\partial f}{\partial x}$$



- In der Praxis ist es häufig schwierig, die Jacobi-Matrix mit vertretbarem Aufwand explizit (d.h. in Formeln) zu ermitteln

→ Alternative
nötig?

- Sobald ein Element der Jacobi-Matrix falsch ist, nur noch
(max.) lineare statt quadratische Konvergenz

BESTE FALL

$$\frac{\partial f_i}{\partial x_j} = \text{falsches Vorzeichen}$$

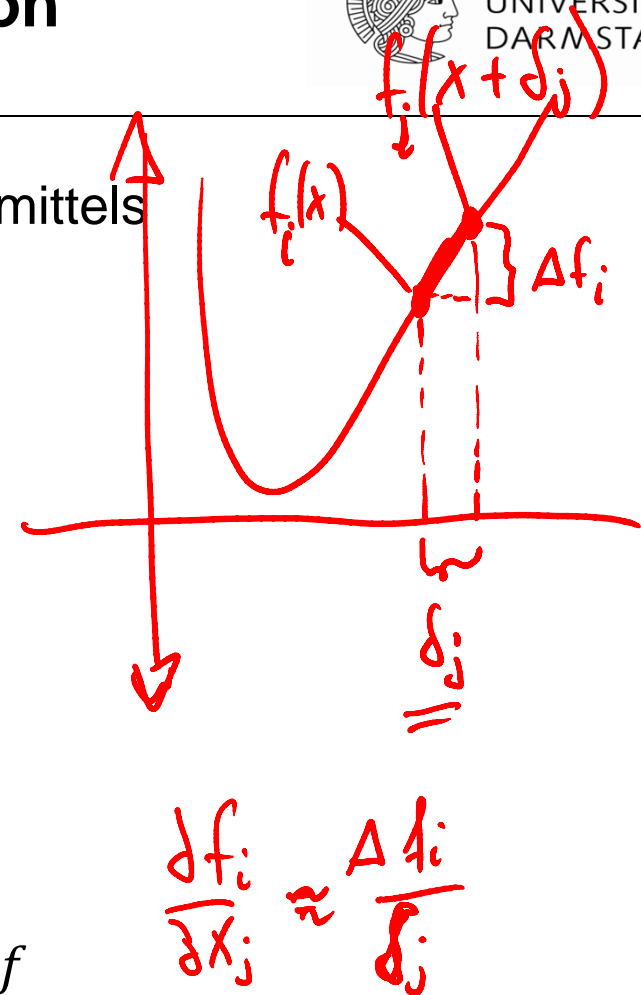
- Darum in der Praxis häufig Vorwärtsdifferenzen-Approximation
der Jacobi-Matrix

3.5.11 Vorwärtsdifferenzen-Approximation

- Approximation der j -ten Spalte der Jacobi-Matrix mittels Vorwärtsdifferenzenquotient

$$\frac{\partial f}{\partial x_j} \approx \frac{1}{\delta_j} \left(f(x + e_j \delta_j) - f(x) \right)$$

- j -ter Einheitsvektor e_j
- Schrittweite δ_j , z.B. $\delta_j = \varepsilon \cdot (1 + |x_j|)$
- Toleranz ε
- Approximation erfordert $n + 1$ Auswertungen von f



3.5.11 Vorwärtsdifferenzen-Approximation

- Man betrachte |exakter Wert – Approximation|

$$\left| \frac{\partial f_i(x)}{\partial x_j} - \frac{1}{\delta_j} (f_i(x + e_j \delta_j) - f_i(x)) \right| \leq |\delta_j| \cdot \left| \frac{\partial^2 f_i(\hat{x})}{\partial x_j^2} \right|$$

= Fehler

- Vorwärtsdifferenzenapproximation liefert in der Regel maximal die Hälfte der gültigen Dezimalstellen von f

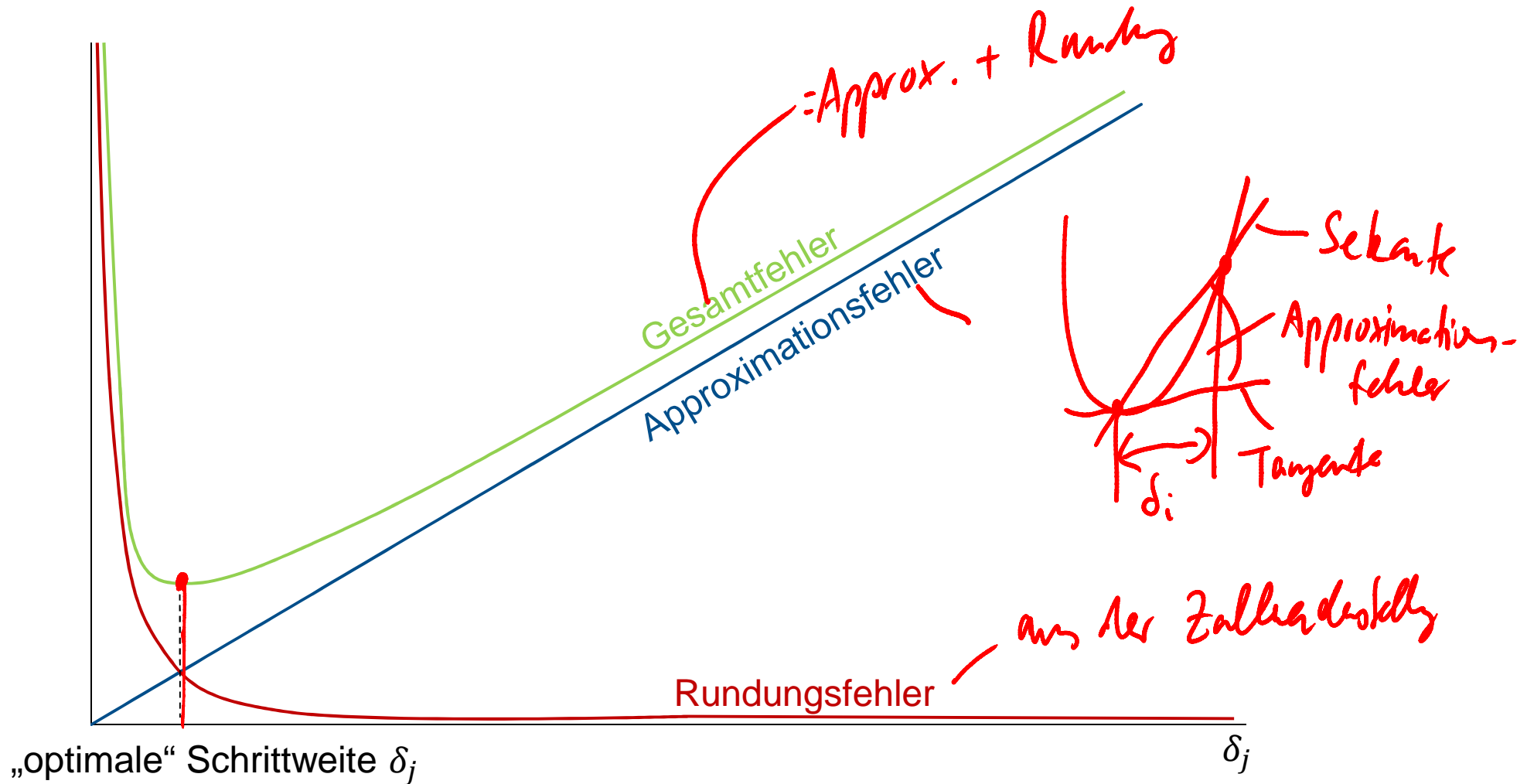
- Wird δ immer kleiner, nimmt der Einfluss von Rundungsfehlern zu

f = auf 10 Stellen
 $\Rightarrow \frac{\partial f}{\partial x}$ maximal auf 5 Stellen!

3.5.11 Jacobi-Matrix: Numerische Genauigkeit



TECHNISCHE
UNIVERSITÄT
DARMSTADT



3.5.11 Jacobi-Matrix: Besetztheitsstruktur

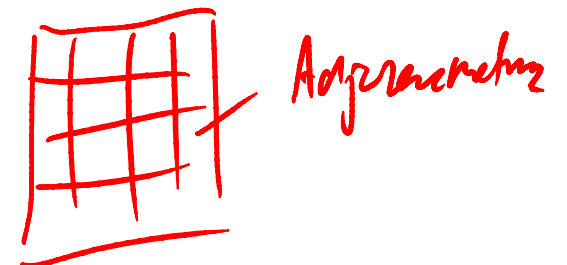


TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Besetztheitsstruktur gibt die Struktur der Kopplung in den Differentialgleichungen wieder
- Sehr wichtig für die Simulation größerer Systeme!
 - Ausnutzung von dünner Besetztheitsstruktur zur *optimalen* effizienten numerischen Simulation

$n = 10\,000$ 4B
 $f(x)$, x als 10000 floats

$$\frac{\partial f_i(x)}{\partial x_j} = \frac{100\,000\,000}{400\text{ MB}}$$



3.5.11 Jacobi-Matrix: Besetztheitsstruktur Beispiel

- Beispiel für $J_f \in \mathbb{R}^{n \times n}$ mit $n = 3$:

$$\frac{\partial f}{\partial x} = \begin{matrix} & \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \\ \begin{matrix} \times & 0 & \times \\ 0 & \times & 0 \\ 0 & \times & \times \end{matrix} & \begin{matrix} f_1 \\ f_2 \\ f_3 \end{matrix} \end{matrix}$$

Handwritten red annotations: a horizontal line under the first column of the matrix, circles around the \times elements in the first and third columns, and a vertical line under the second column. To the right of the matrix are handwritten red symbols resembling a stylized '5' and a '9'.

- Hier hängt z.B. $f_1(x)$ nur von x_1 und x_3 ab



3.5.11 Jacobi-Matrix: Optimierung

- Untersuchung der Jacobi-Matrix auf Dünnbesetztheit
 - Effizientere Speicherung der Jacobi-Matrix mithilfe von Sparse Matrizen
 - Approximation der von Null verschiedenen Einträge
- Berechnung einer konstante Jacobi-Matrix (z.B. $J_f(x_0)$)
 - Verfahren ist nicht mehr quadratisch, sondern höchstens linear konvergent!
 - Falls konvergent, dann dennoch als „normales“ Newton-Verfahren

TH2

Hier kann man noch ein echt tolles Beispiel für Sparse Matrizen mit reinnehmen und evtl. auch einfach mal die Matlab Befehle dafür vorstellen!

Beispiel: <http://www.andreas-schreiber.net/diplomarbeit/node13.html>

Matlab: <http://www.mathworks.de/de/help/matlab/ref/sparse.html>

Und hier ist noch ein echt nices paper zu dem Thema!

http://www.hpl.hp.com/personal/Robert_Schreiber/papers/Sparse%20Matrices%20in%20Matlab/simax.pdf

Thomas Hesse; 30.11.2013

3.5.11 Jacobi-Matrix: Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Approximation durch schrittweise Addierung mittels Update
 - Approximatives Verfahren, wird auch Quasi-Newton Verfahren genannt
 - Im Allgemeinen ein Rang-1 Verfahren (z.B. nach Broyden)
 - Bei symmetrischer Jacobi-Matrix ein Rang-2 Verfahren





3.5.12 Jacobi-Matrix Approximation

- Allgemeines Rang-1, bzw. Rang-2 Vorgehen für $J_{f,i} \approx \frac{\partial f}{\partial x}(x_i)$ bei
gegebener Funktion $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$
- Schrittweise Addierung mittels Update $J_{f,i} = J_{f,(i-1)} + \underline{U_i}$ mit einer
Matrix U_i vom Rang 1 oder 2
 - Sekanten-Bedingung (Quasi-Newton-Bedingung) muss erfüllt sein

$$\underline{J_{f,i}} \cdot \underline{(x_i - x_{(i-1)})} = \underline{f(x_i) - f(x_{(i-1)})}$$

3.5.12 Jacobi-Matrix: (Quasi-)Newton Verfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Gegeben ist eine Funktion $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ und gesucht ist ein x_s womit dann gilt $f(x_s) = 0$

- Das Verfahren beginnt bei einem x_i (wobei $i = 0$), also x_0

- Iterationsschritt von i nach $i + 1$ ist wie folgt:

- Berechnung von $J_{f,i} \cdot \Delta x_i = -f(x_i)$ und auflösen nach Δx_i

mit $J_{f,i} \approx \frac{\partial f}{\partial x}(x_i)$ und $x_{i+1} = \Delta x_i + x_i$



3.6 Systemtypen von Differentialgleichungen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

$$\begin{array}{l} \dot{\underline{x}} = \underline{f}(\underline{x}, \underline{u}, \underline{t}) \\ \underline{y} = \underline{g}(\underline{x}, \underline{u}, \underline{t}) \end{array}$$

System im Allgemeinen
nichtlinear und
zeitvariant (und explizit)

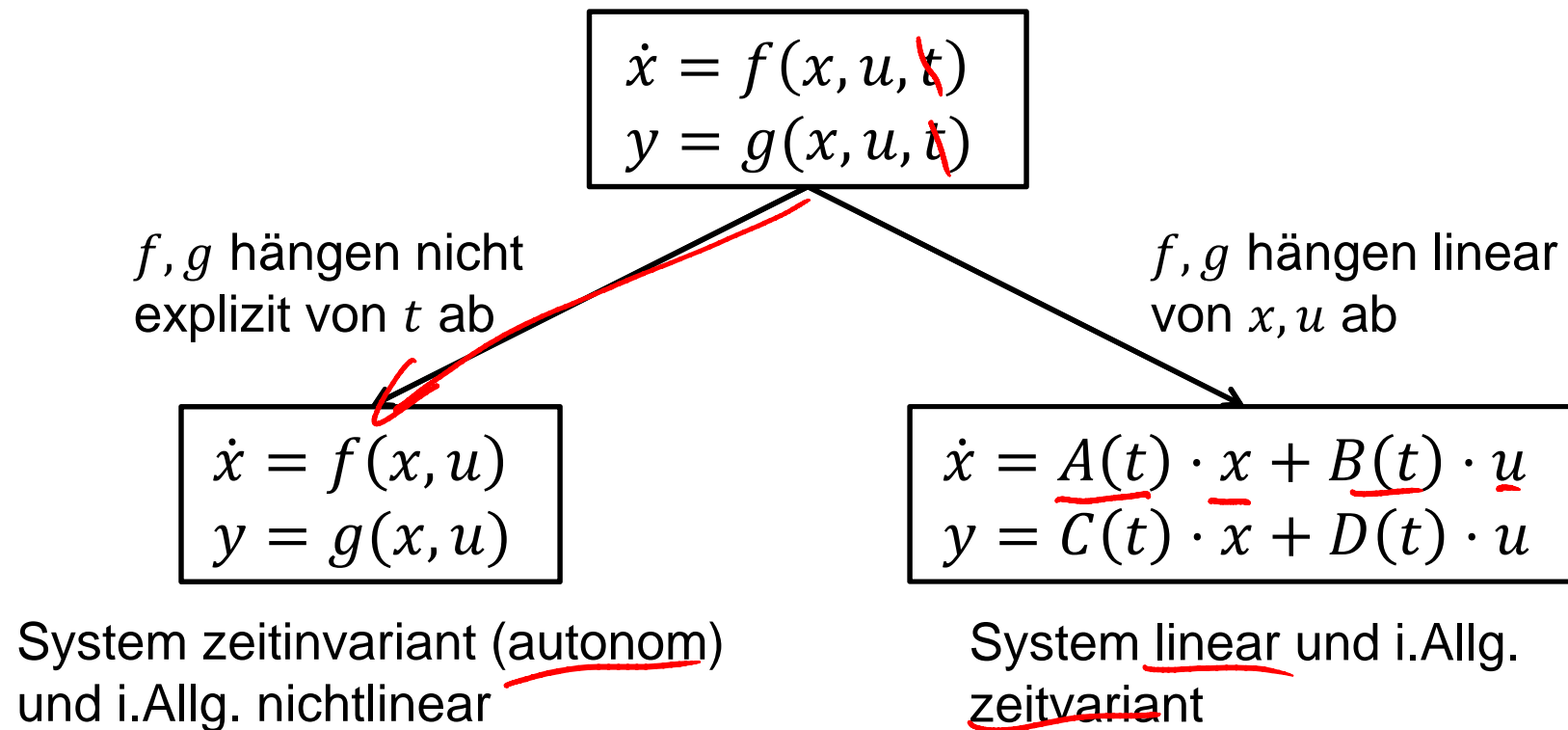
GLEICH
KOMMT
EIN
MOODLE
TEST



3.6 Systemtypen von Differentialgleichungen



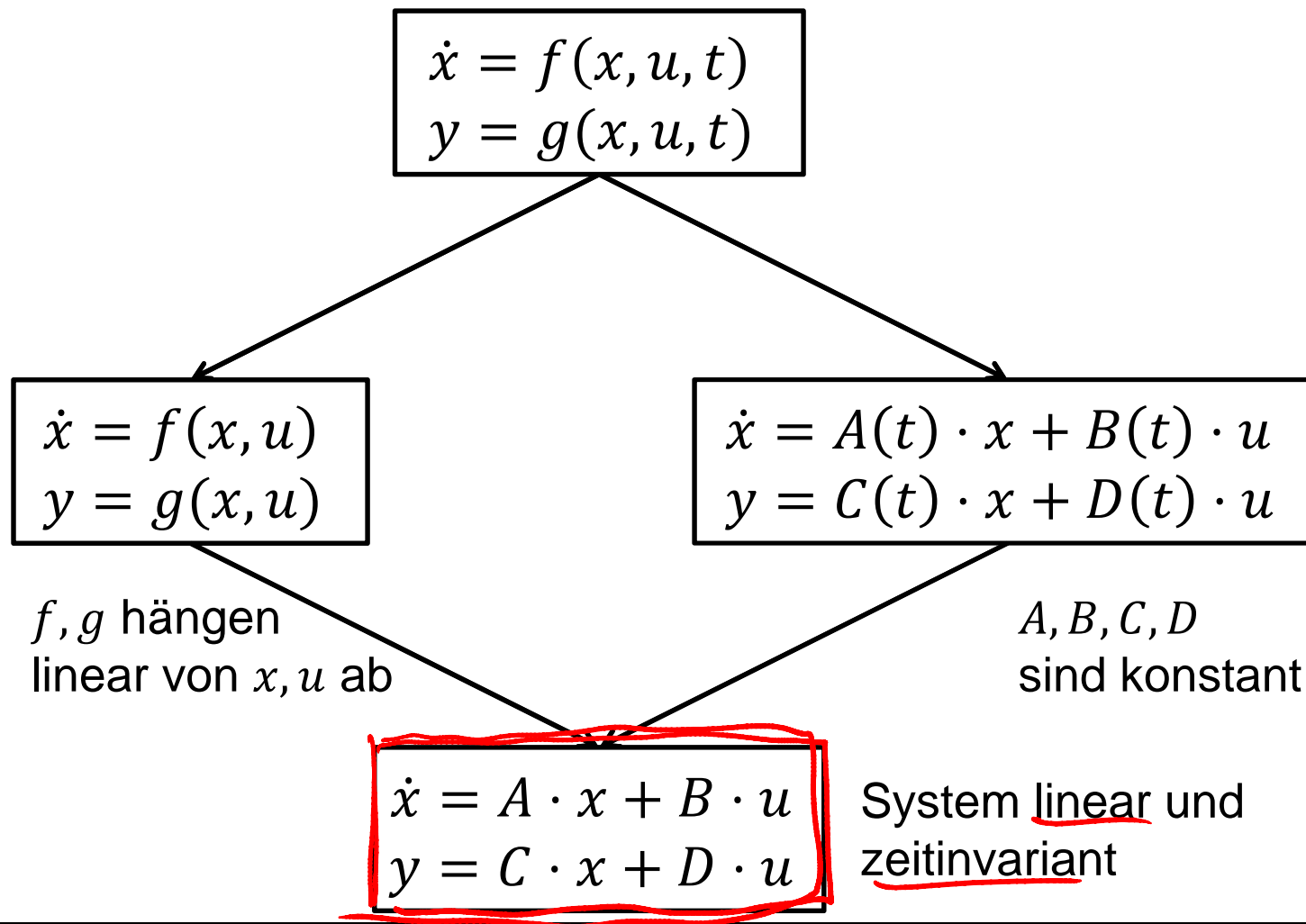
TECHNISCHE
UNIVERSITÄT
DARMSTADT



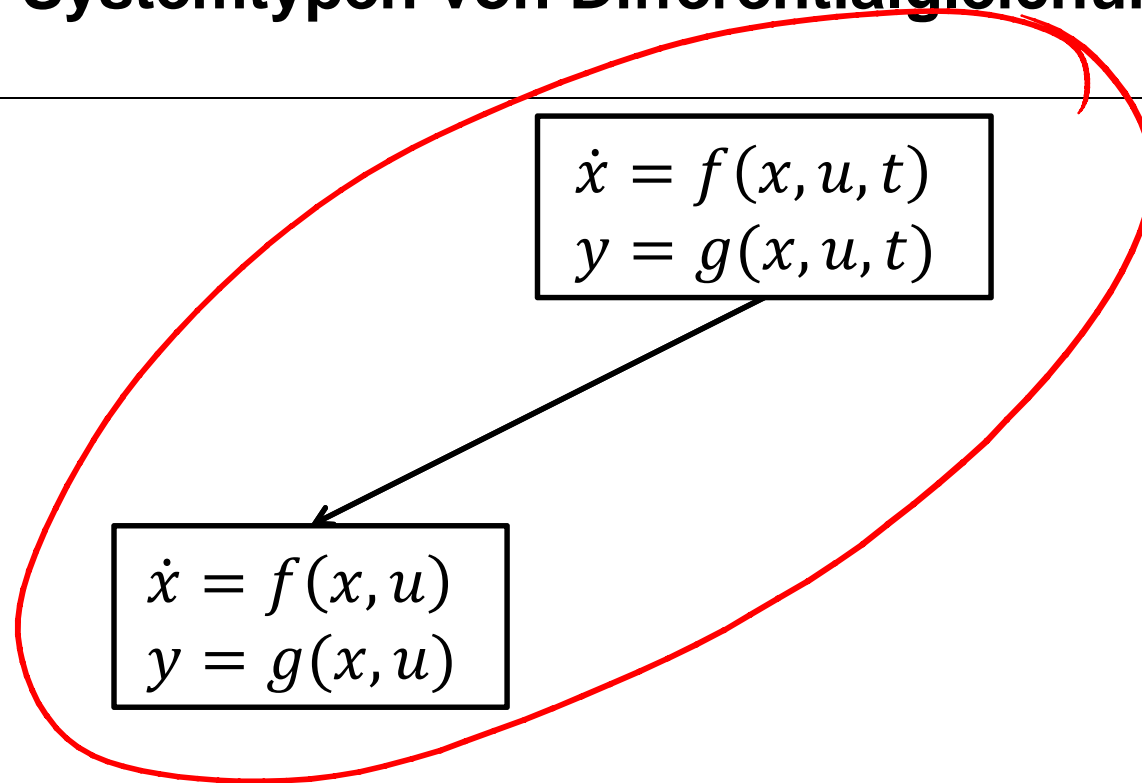
3.6 Systemtypen von Differentialgleichungen



TECHNISCHE
UNIVERSITÄT
DARMSTADT



3.6 Systemtypen von Differentialgleichungen



- Für diese Fälle ist im Allg. keine explizite, formelmäßige, nur numerische Lösung möglich!

MOODLE FRAGE

$$\left. \begin{array}{l} \dot{x} = A(x, t) \\ t = 1 \end{array} \right\} \begin{array}{l} n+1 \text{ Ordng} \\ \text{antworten} \end{array}$$



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bitte jetzt auf Moodle eine Frage
beantworten!

n-ter Ordnung
nicht antworten

x(t)



$$\dot{x} = A(x)$$

A(x)

$$\begin{array}{l} \dot{x} = A(x) \\ x(t) = \int_0^t A(x) dt \end{array}$$

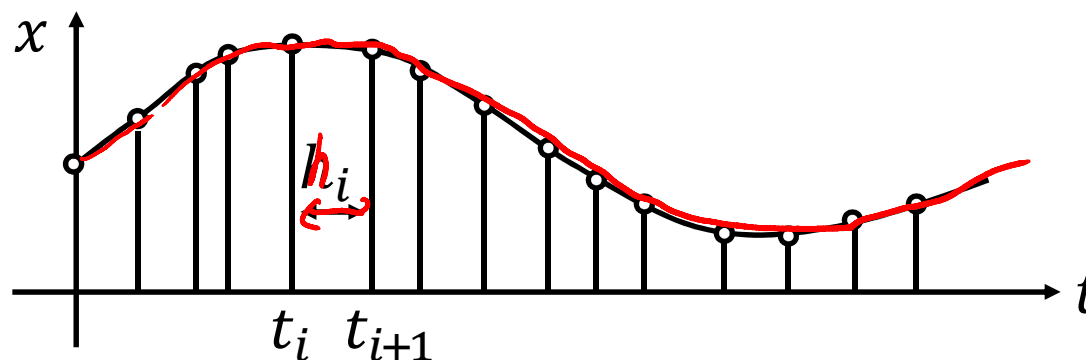
3.6.1 Numerische Integration: Grundidee

- Man betrachte die skalare (Zustands-)DGL:

$$\dot{x} = f(x), \quad x(0) = 0, \quad 0 \leq t \leq t_f$$

mit “final time” t_f

- Aufstellen der Lösung zu diskreten Zeitpunkten $t_{i+1} = t_i + h_i$





3.6.1 Numerische Integration: Grundidee

- Integration der (Zustands-)DGL liefert:

$$\int_0^t \dot{x}(\tau) d\tau = x(t) - x(0) = \int_0^t f(x(\tau)) d\tau$$

- Und daraus folgt:

$$x(t_{i+1}) = x(0) + \int_0^{t_i} f(x(\tau)) d\tau + \int_{t_i}^{t_{i+1}} f(x(\tau)) d\tau = x(t_i) + \int_{t_i}^{t_{i+1}} f(x(\tau)) d\tau$$

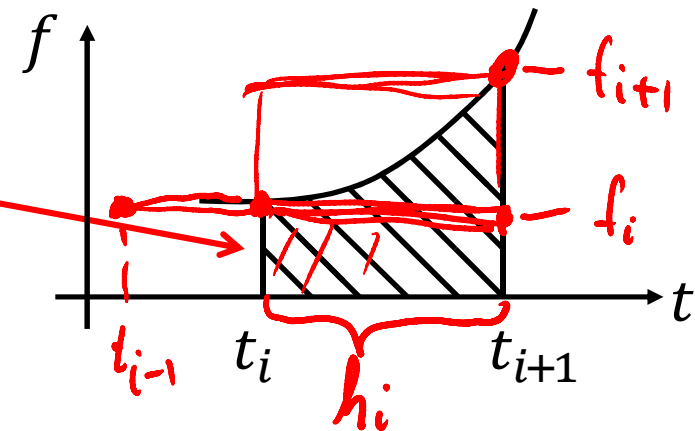
$x(t_i)$

3.6.1 Numerische Integration: Grundidee

- Ausgangsgleichung:

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(x(\tau)) d\tau$$

Approximieren



- Ansatz für numerische Integrationsverfahren
 - Durch diskretisieren und summieren von Teillösungen ergibt sich die Ausgangsgleichung:

$$x(t_{i+1}) = x(t_i) + \underline{h_i} \cdot \underline{f_i}, \quad \underline{f_i} := \underline{f(x(t_i))}$$

MOODLE
TEST IN
DREI
FOLIEN

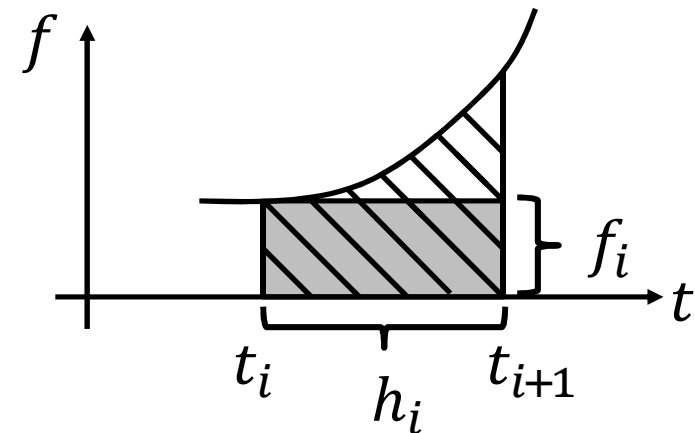
3.6.1 Numerische Integration: Grundidee

- Ausgangsgleichung:

$$x(t_{i+1}) = x(t_i) + h_i \cdot f_i, \quad f_i := f(x(t_i))$$

"Untersumme"

- Rechteck Approximation als Ansatz
die Fläche unter der Funktion zu bestimmen



- Lässt sich auch mit dem Vorwärtsdifferenzenquotient formulieren als:

$$\frac{(x(t_{i+1}) - x(t_i))}{h_i} \approx \dot{x}(t_i) = f(x(t_i))$$



3.6.1 Numerische Integration: Übersicht

- Unterscheidung in der Art der Approximation der Fläche unter der Funktion f und/oder des Gradienten dieser
 - Einschrittverfahren (ESV), z.B. $\int_{t_i}^{t_{i+1}} f(x(\tau)) d\tau \approx hf(x(t_i))$
 - Mehrschrittverfahren (MSV), z.B. $\int_{t_i}^{t_{i+1}} f(x(\tau)) d\tau \approx \frac{h}{2} (f(x(t_i)) + f(x(t_{i+1})))$
 - Extrapolationsverfahren
- Diese Verfahrensklassen können weiter unterteilt werden:
 - Explizites Verfahren $\int_{t_i}^{t_{i+1}} f(x(\tau)) d\tau \approx hf(x(t_i))$ *4 untere!*
 - Implizites Verfahren $\int_{t_i}^{t_{i+1}} f(x(\tau)) d\tau \approx hf(x(t_{i+1}))$ *2 obere!**Symplektisch*



Bitte jetzt auf Moodle eine Frage
beantworten!



3.6.2 Einschrittverfahren: Allgemein

- Gegeben sei ein $x_i \approx x(t_i)$, oft mit $i = 0$

- Gesucht ist $x_{i+1} \approx x(t_{i+1})$

- Allgemeiner Ansatz gegeben durch:

$$x_{i+1} = x_i + h_i \cdot \Phi(t_i, x_i, x_{i+1}, h; f)$$

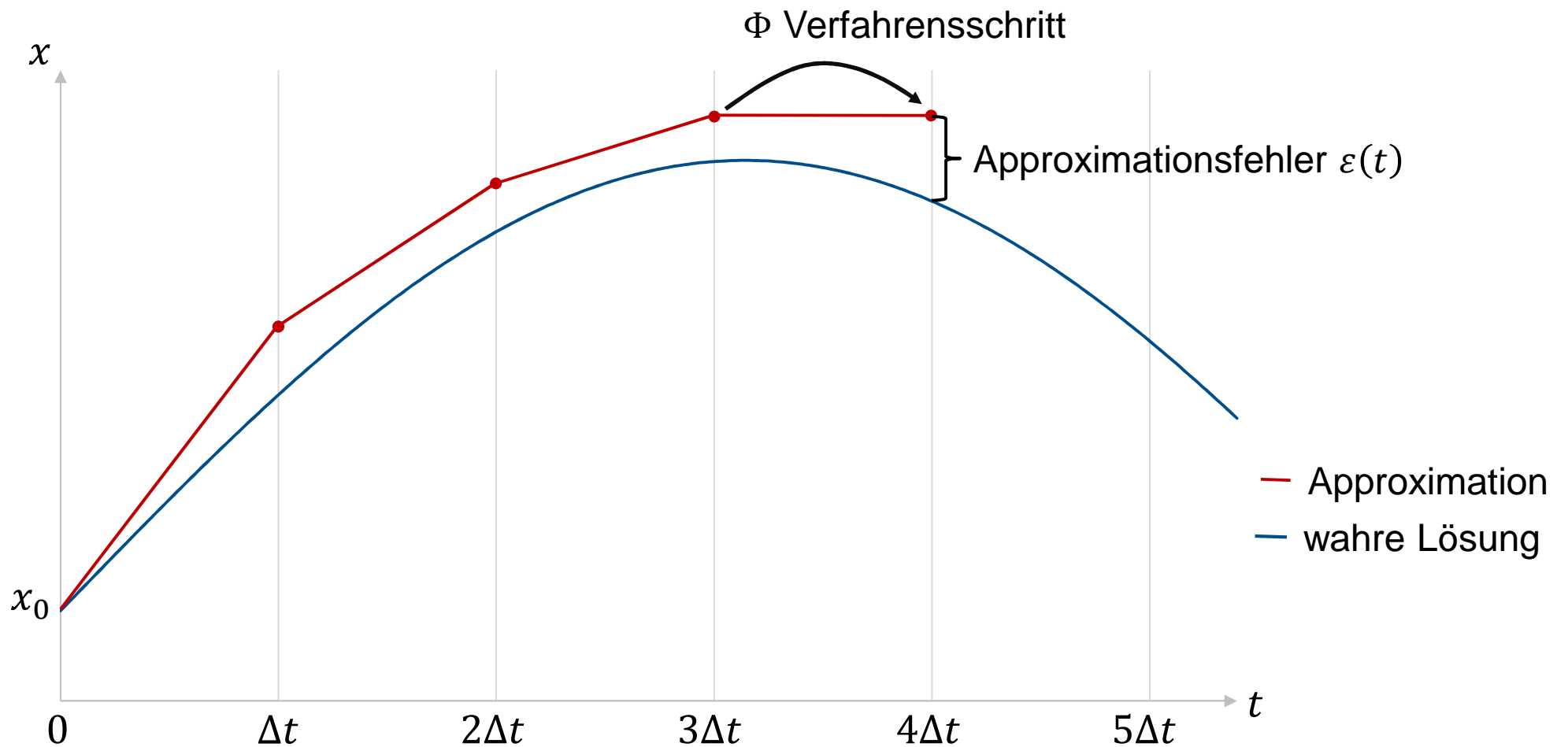
$$= x_i + \underbrace{h_i}_{\Delta t} \cdot f(x_i)$$

$$= \int_{t_i}^{t_{i+1}} f(x(t)) dt$$

- Konsistenzbedingung (muss immer gelten):

$$\lim_{h \rightarrow 0} \Phi(t_i, x_i, x_{i+1}, h; f) = f(x_i)$$

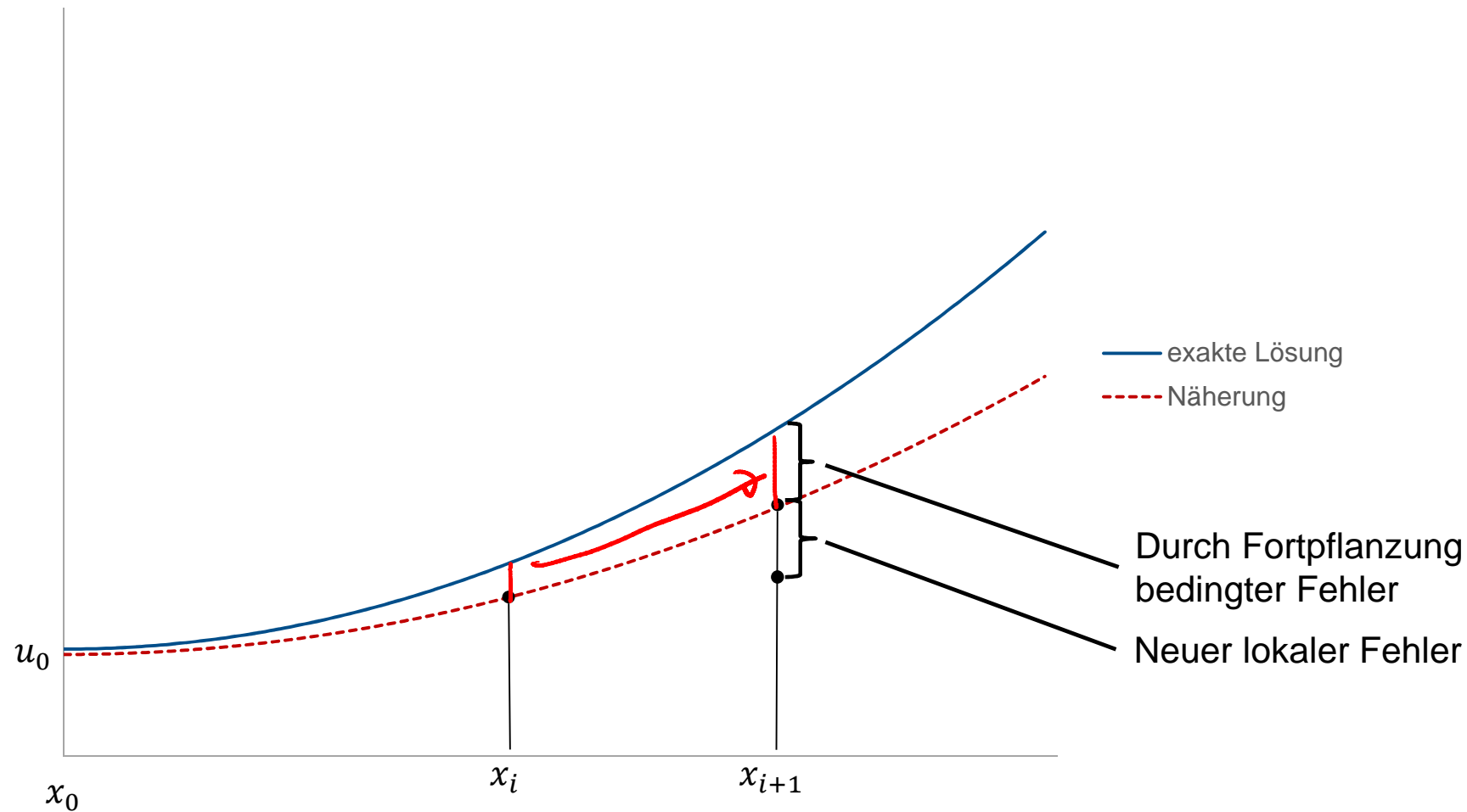
3.6.2 Einschrittverfahren: Allgemein



3.6.3 Einzelschritt- und Fortpflanzungsfehler

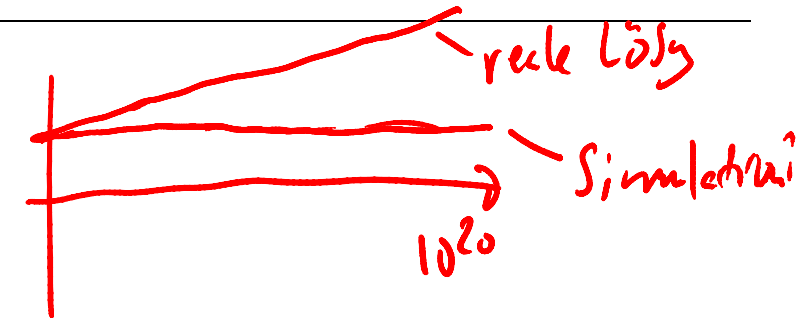


TECHNISCHE
UNIVERSITÄT
DARMSTADT



3.6.3 Einzelschritt- und Fortpflanzungsfehler: Rundungsfehler

- Beispiel: $\dot{x} = 10^{-5} \cdot x$, $x_0 = 1$
- Schrittweite: $\Delta t = 10^{-5}$
- Rechengenauigkeit 10 Dezimalstellen



- Euler-Schritt

$$x_1 = x_0 + \Delta t \cdot 10^{-5} \cdot x_0 = x_0 + 10^{-10} x_0$$

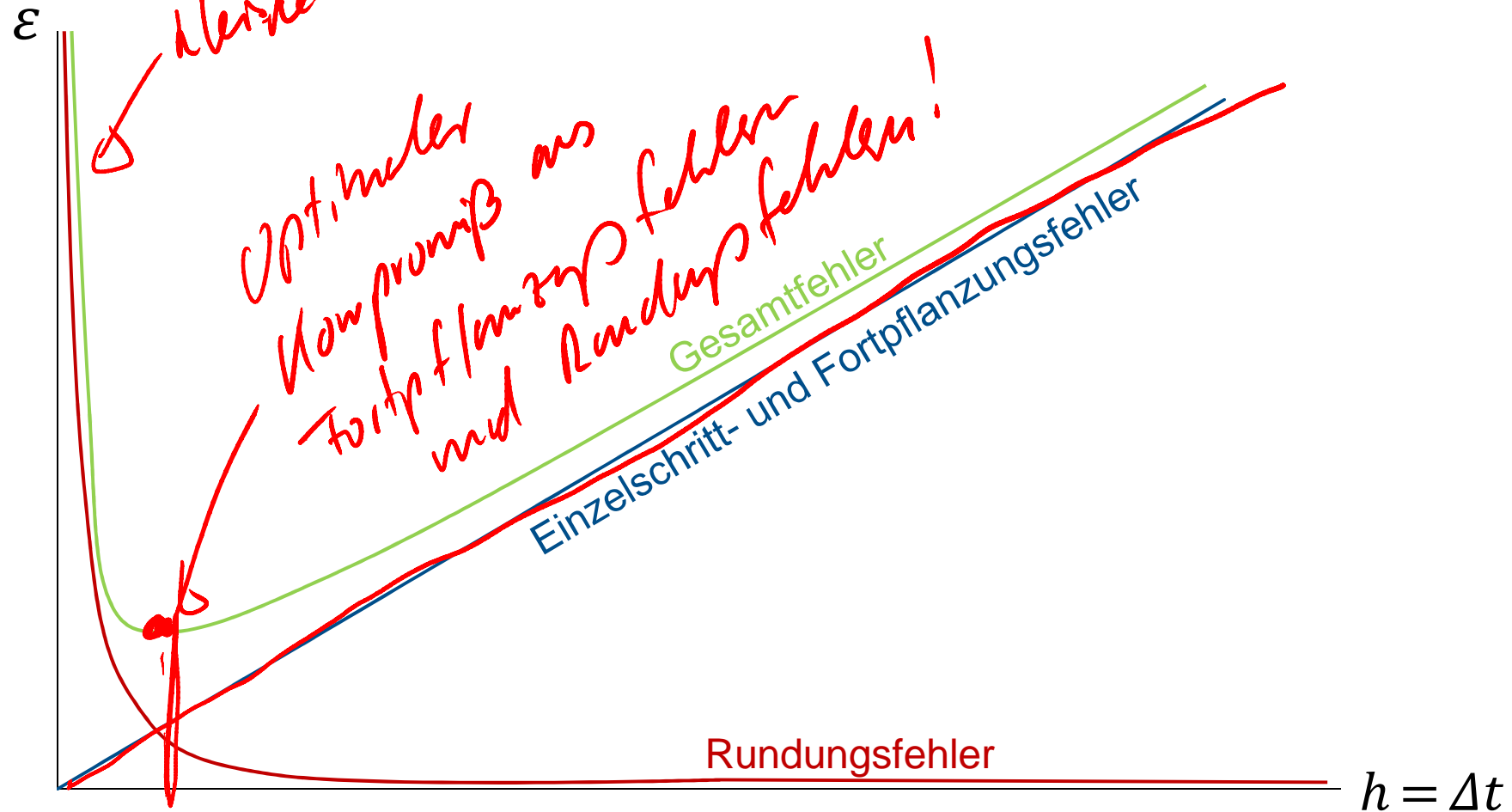
	1.0000000000	0
+	0.0000000000	1
<hr/>		
	1.0000000000	

Rundung
weggerundet!

3.6.3 Verlauf des Gesamtfehlers



TECHNISCHE
UNIVERSITÄT
DARMSTADT





3.6.3 Verlauf des Gesamtfehlers

- **Fazit:** Notwendigkeit für adaptive Schrittweitensteuerung gegeben
- Vermeidung von unnötig kleinen Zeitschritten zur Steigerung der Recheneffizienz
- Vermeidung von großem Gesamtfehler durch angepasste und hinreichend kleine Zeitschritte
 - Gesamtfehler liegt dabei i.d.R. unter selbstgewählter Toleranzgrenze, welche Genauigkeitsgrad der Approximation angibt

$$t_f = N \Delta t$$
$$O(N) = O\left(\frac{t_f}{\Delta t}\right)$$

↑
kleiner Δt
 \Rightarrow mehr Schritte

3.6.4 Einschrittverfahren: Euler-Verfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Explizites Euler-Verfahren für skalar $x(t_i)$

- Verfahrensvorschrift gegeben durch Δt $\dot{x}(t_i)$
 $x(t_{i+1}) = x(t_i) + h_i \cdot f(x(t_i))$

mit Verfahrensfunktion $\Phi := f$

- Formulierung der Verfahrensvorschrift aus dem
Vorwärtsdifferenzenquotienten





3.6.4 Explizites Euler-Verfahren: Beispiel

- Beispiel-DGL: $\dot{x} = -x(t)$, mit $x(0) = 1$
- Gesucht: $x(t)$, mit $0 \leq t \leq 5$
- Anwendung des Euler-Verfahrens ($x(t_i) = x_i$)

$$\underline{x(t_{i+1}) = x(t_i) + h_i \cdot f(x(t_i))}, \quad \text{mit } h_i = h = 0.5$$

$$\underline{x_0 = x(0) = 1}$$

$$\underline{x_1 = x(0.5) = x_0 + 0.5 \cdot (-1) = 0.5}$$

$$\underline{x_2 = x(1) = x_1 + 0.5 \cdot (-0.5) = 0.25}$$

$$\underline{x_3 = x(1.5) = x_2 + 0.5 \cdot (-0.25) = 0.125}$$

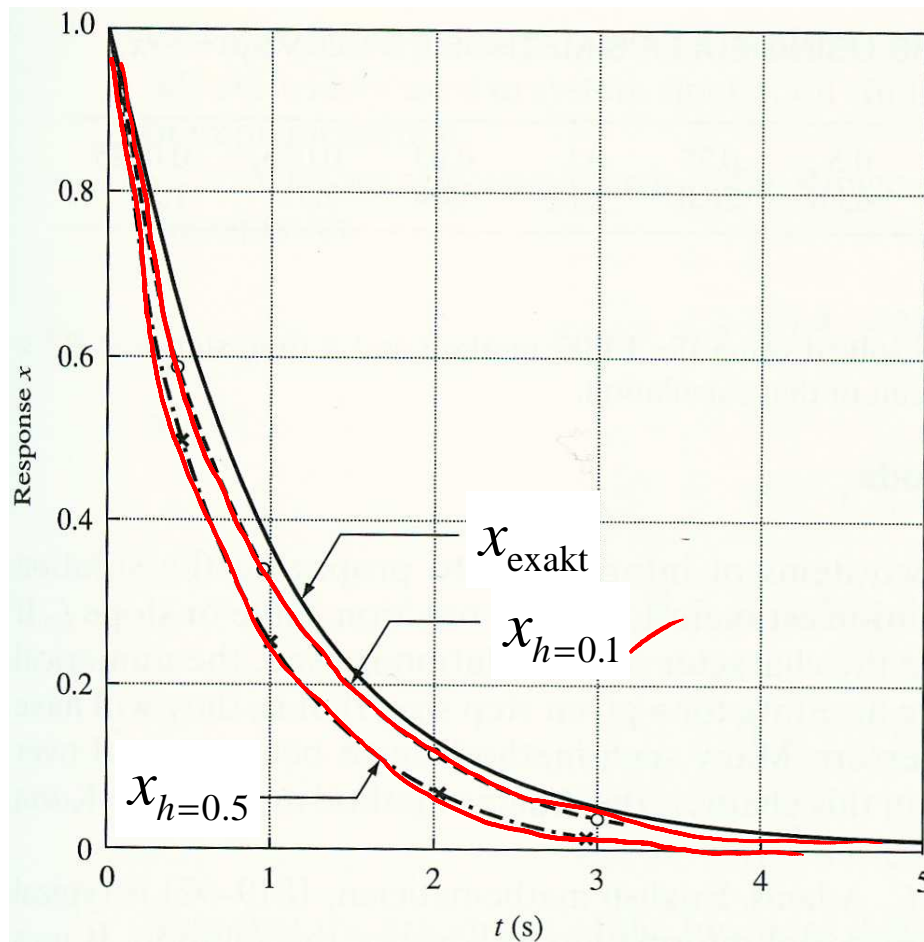
⋮

$\left(\frac{1}{2}\right)^0$
 $\left(\frac{1}{2}\right)^1$
 $\left(\frac{1}{2}\right)^2$
 $\left(\frac{1}{2}\right)^3$
⋮

3.6.4 Explizites Euler-Verfahren: Beispiel



TECHNISCHE
UNIVERSITÄT
DARMSTADT



$h = 0.5$			
t	x (Euler)	x (exact)	error (%)
0.5	0.500000	0.606531	17.56
1.0	0.250000	0.367879	32.04
1.5	0.125000	0.223130	43.98
2.0	0.062500	0.135335	53.82
2.5	0.031250	0.082085	61.93
3.0	0.015625	0.049787	68.62
3.5	0.007813	0.030197	74.13
4.0	0.003906	0.018316	78.67
4.5	0.001953	0.011109	82.42
5.0	0.000977	0.006738	85.51

$h = 0.1$			
t	x (Euler)	x (exact)	error (%)
0.5	0.590490	0.606531	2.64
1.0	0.348678	0.367879	5.22
1.5	0.205891	0.223130	7.73
2.0	0.121577	0.135335	10.17
2.5	0.071790	0.082085	12.54
3.0	0.042391	0.049787	14.86
3.5	0.025032	0.030197	17.11
4.0	0.014781	0.018316	19.30
4.5	0.008728	0.011109	21.43
5.0	0.005154	0.006738	23.51



3.6.4 Einschrittverfahren: Euler-Verfahren

- Implizites Euler-Verfahren für skalar $x(t_i)$

- Verfahrensvorschrift gegeben durch

$$x(t_{i+1}) = x(t_i) + h_i \cdot \underline{f(x(t_{i+1}))}$$

mit Verfahrensfunktion $\Phi := f$

- Auflösen der rechten Seite, anstatt der linken Seite



MOODLE
TEST IN
ZWEI
FOLIEN

3.6.4 Einschrittverfahren: Euler-Verfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Implizites Euler-Verfahren für skalar $x(t_i)$
- (Näherungsweise) Lösen einer nicht-lineare Gleichung (z.B. mit Newton-Verfahren) pro Iterationsschritt:

$$0 = x(t_{i+1}) - x(t_i) - h_i \cdot \underline{f(x(t_{i+1}))}$$

- Verknüpft mit erhöhtem Rechenaufwand relativ zum expliziten Euler-Verfahren

Expliziter Euler
 $x(0) = x_0$
for (t = ...) {
 $x_{i+1} = x_i + h_i f(x_i)$
}

Impliziter Euler
 $x(0) = x_0$
for (...) {
 NEWTON
 $x_{i+1} = \dots$
}

Genauigkeit!
hoher
Preis



3.6.4 Allgemeine Formulierung: Euler-Verfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Gegeben sei die Näherungslösung $x(t_i)$ zum Zeitpunkt t_i
- Gesucht ist die Näherungslösung $x(t_{i+1})$ zum Zeitpunkt t_{i+1}
- Explizites Euler-Verfahren für Vektor $x(t_i)$
 - Verfahrensvorschrift: $x(t_{i+1}) = x(t_i) + h_i \cdot f(x(t_i))$
- Implizites Euler-Verfahren für Vektor $x(t_i)$
 - Verfahrensvorschrift: $x(t_{i+1}) = x(t_i) + h_i \cdot f(x(t_{i+1}))$





Bitte jetzt auf Moodle eine Frage
beantworten!