4. Kontinuierliche und hybride Simulation

Bisher wurden ereignisdiskrete Systeme analysiert
(Ausnahme kurzer „Ausflug in MAO“)

Also Systeme, bei denen Zustandsänderungen nur zu Ereigniszeitpunkten auftreten

Solche Systeme sind insbesondere im technischen Bereich zu finden, wenn einzelne Individuen (Kunden, Teile,, ...) betrachtet werden

Andere Systeme sind gekennzeichnet durch kontinuierliche Änderungen der Zustandsvariablen oder durch die aggregierte Betrachtung diskreter Vorgänge (Übergänge sind fließend)

$\Rightarrow$ **kontinuierliche Simulation**

Basismodell Differentialgleichungssysteme

In manchen Bereichen wird der Begriff Simulation (fälschlicherweise) mit der numerischen Lösung von Differentialgleichungen gleichgesetzt

Neben der etablierten kontinuierlichen Simulation gibt es verstärkt Anwendungen, bei denen kontinuierliche und diskrete Anteile eine Rolle spielen, z.B.

- Diskontinuitäten in kontinuierlichen Modelle
- Steuerung kontinuierlicher Prozesse durch diskrete Steuerung
- ➢ Hybride Modelle mit kontinuierlichen und diskreten Anteilen entstehen

Damit ergeben sich Fragen nach der

- Beschreibung solcher Modelle
- Simulation der Modelle
- Auswertung der Resultate

Ziele:

- Basismodelle der kontinuierlichen Simulation kennen lernen
- Erkennen, wann kontinuierliche, wann diskrete und wann hybride Simulation eingesetzt werden soll/kann
- Überblick über typische Anwendungsgebiete der kontinuierlichen und hybriden Simulation bekommen
- Beschreibungstechniken und –sprachen für kontinuierliche und hybride Systeme kennen lernen
- Grenzen der kontinuierlichen und hybriden Simulation erkennen
- Basismethoden zur numerischen Analyse kontinuierlicher Simulationsmodelle kennen lernen
- Simulationsalgorithmen für hybride Systeme kennen lernen

Im Gegensatz zur diskreten Simulation gibt es für die kontinuierliche Simulation kaum Lehrbücher, die die Methodik allgemein erläutern

Am ehesten geeignet sind:

- H. Bossel. Simulation dynamischer Systeme, Vieweg 1982.
- F. E. Cellier. Continuous System Modeling, Springer 1991.
- F. E. Cellier, E. Kofman. Continuous System Simulation, Springer 2006.

aus denen die Ergebnisse zur kontinuierlichen Simulation entnommen wurden.

Für die hybride Simulation wurden im Wesentlichen Originalartikel verwendet.

Kapitelgliederung:

4.1 Beispiele kontinuierlicher Simulationsmodelle

4.2 Beschreibungsansätze für kontinuierliche Modelle

4.3 Analyse kontinuierlicher Modelle

4.4 Beispiele hybrider Simulationsmodelle

4.5 Hybride Simulationsansätze

4.6 Beschreibungsansätze für hybride Modelle

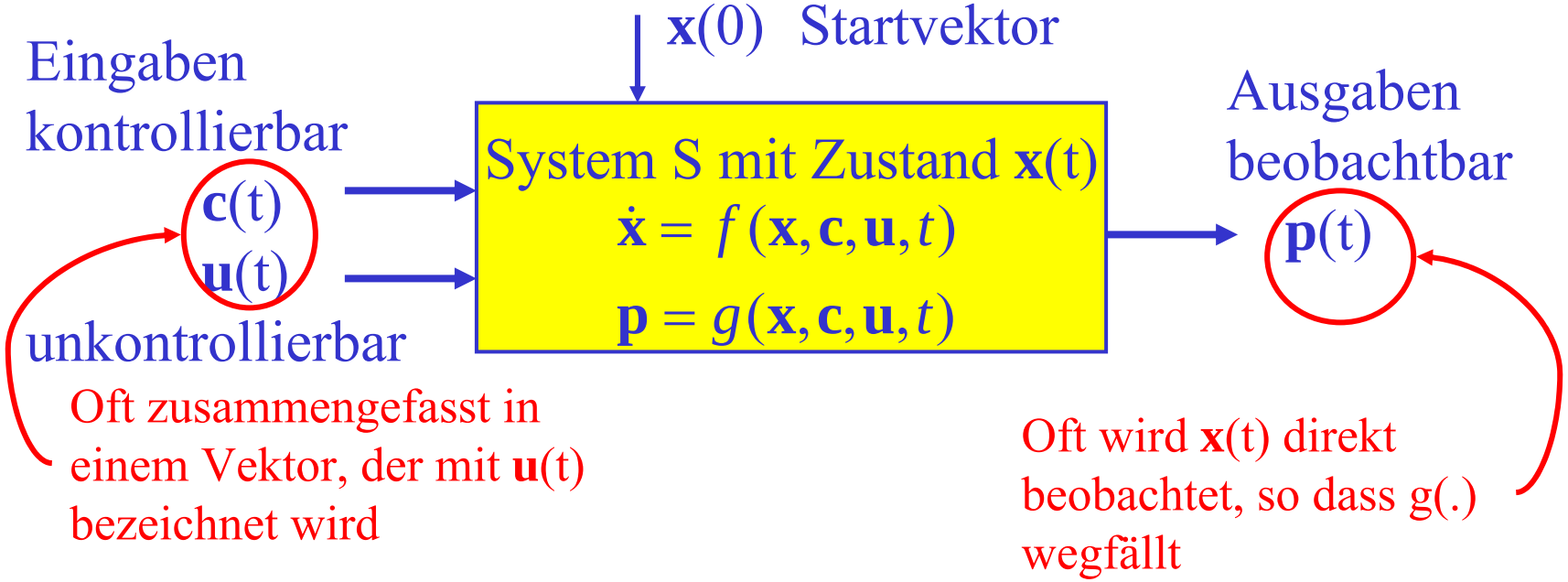
4.1 Beispiele kontinuierlicher Simulationsmodelle

Beschreibung kontinuierlicher Vorgänge in der Natur und Technik

Ebenen der Modellierung

- Deduktive Modellierung und white-box-Modelle
  - Elektrotechnik, Mechanik, ...
    Abläufe sind gut verstanden und können durch Differentialgleichungen exakt beschrieben werden
- Mischung aus induktiver und deduktiver Modellierung und grey-box-Modelle
  - Biologie, Chemie, ...
    Abläufe sind nur zum Teil detailliert erklärbar/herleitbar, gewisse Einflüsse müssen durch die Kalibrierung der Modelle induktiv ergänzt werden
- Rein induktive Modellierung und black-box-Modelle
  - Sozialwissenschaften, Weltmodelle, ...
    Beobachtung der Realität liefert Zusammenhang zwischen Eingaben C, U und Ausgaben P, Zusammenhang wird durch mathematische Funktion dargestellt

Etwas detailliertere Darstellung eines (kontinuierlichen) Modells



Bemerkungen

- In manchen Darstellungen wird ein Parametervektor von f (.) und g(.) mit einbezogen, wir gehen davon aus, dass die Parameter mit dem Modell festliegen
- Die Schreibweise $\dot{\mathbf{x}}$ statt$\mathbf{x}'$ wird verwendet, wenn nach der Zeit t abgeleitet wird
- Wir betrachten im Folgenden die einfache Variante der Darstellung mit einem Eingabevektor **u**(t), Ausgaben **x**(t) und Systemfunktion f(**x**,**u**,t) oder teilweise auch nur f(**x**,t)

Beispiel aus dem Bereich Populationsdynamik: Räuber-Beute System

Klasse von Modellen, die den Zusammenhang zwischen unterschiedlichen Populationen in Ökosystemen ausdrücken

Unterschiedliche Arten der Dynamik können ausgedrückt werden

- Ressourcen, die einem „natürlichen“ Wachstumsprozess unterworfen sind, z.B.
  - Exponentielles Wachstum (falls BR > DR)
    BR Geburtsrate, DR Todesrate, P Population:
    $\dot{P} = BR - PR$
    realistischer ist eine von der Population abhängige Rate
    $BR = k_{BR} \cdot P$ und $DR = k_{DR} \cdot P \Rightarrow \dot{P} = (k_{BR} - k_{DR}) \cdot P$
  - Logistisches Wachstum (falls BR > DR)
    Endliche Ressourcen beschränken das Wachstum
    $\dot{P} = (k_{BR} - k_{DR}) \cdot P \cdot (1.0 - P / P_{max})$

Bei unterschiedlichen Spezies interagieren diese, z.B. durch

- Räuber – Beute Beziehung
  - Räuber frisst die Beute, dadurch nimmt die Population zu
    ohne Beute stirbt der Räuber aus
  - Beute wird gefressen, dadurch nimmt die Population ab,
    ohne Räuber wächst die Beute exponentiell/logistisch
  - In mehrstufigen Systeme können Spezies Räuber und Beute sein
- Konkurrenz um Ressourcen
  - Spezies stehen nicht direkt in Konkurrenz (durch Räuber-Beute Beziehung), sondern benötigen die selben Spezies als Ressourcen

Lotka-Volterra Modell

$$\dot{x}_{pred} = -a \cdot x_{pred} + k \cdot b \cdot x_{pred} \cdot x_{prey}$$
$$\dot{x}_{prey} = c \cdot x_{prey} - b \cdot x_{pred} \cdot x_{prey}$$

$a, b, c > 0$ und $1 \geq k > 0$

predator = Räuber, prey = Beute

Einfachste Form der Interaktion
Annahmen:

- Anzahl de Begegnungen zwischen Räuber und Beute hängt von den jeweiligen Populationen ab
- Pro Begegnung verliert die Beutepopulation und gewinnt die Räuberpopulation
- Der Gewinn der Räuberpopulation ist höchstens so groß, wie der Verlust der Beutepopulation

Weitere Interaktionsmechanismen

Konkurrenz:

$\dot{x}_1 = a \cdot x_1 - b \cdot x_1 \cdot x_2$

$\dot{x}_2 = c \cdot x_2 - d \cdot x_1 \cdot x_2$

Kooperation:

$\dot{x}_1 = -a \cdot x_1 + b \cdot x_1 \cdot x_2$

$\dot{x}_2 = -c \cdot x_2 + d \cdot x_1 \cdot x_2$

Allgemeine Form der Interaktion zwischen n Spezies:

$$\dot{x}_i = \left(a_i + \sum_{j=1}^{n} b_j \cdot x_j \right) \cdot x_i$$

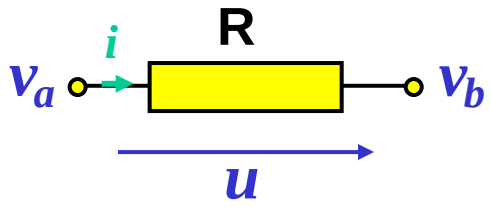
In Vektor-Matrix-Schreibweise:

$$\dot{\mathbf{x}} = (\mathrm{diag}(\mathbf{a}) + \mathrm{diag}(\mathbf{x}) \cdot \mathbf{B}) \cdot \mathbf{x}$$

Beispiel aus dem Bereich Elektrotechnik: Einfache Schaltungen

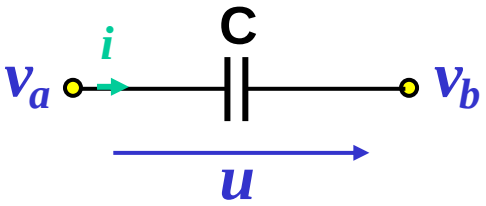
Basiskomponenten und zugehörige Gleichungen (nach Cellier):

Widerstand



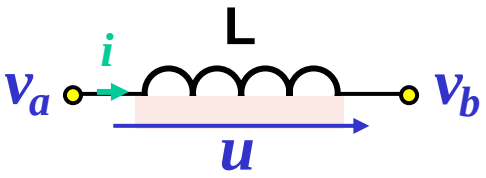
$$u = v_a - v_b$$
$$u = R \cdot i$$

Kondensator



$$u = v_a - v_b$$
$$i = C \cdot \frac{du}{dt}$$

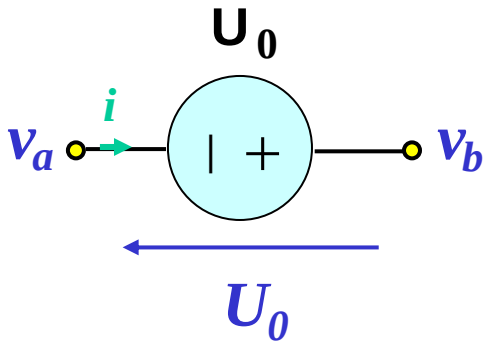
Spule



$$u = v_a - v_b$$
$$u = L \cdot \frac{di}{dt}$$

Weitere Komponenten:

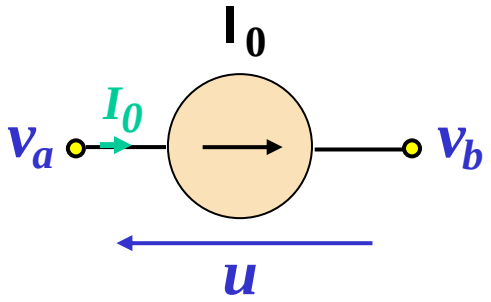
Spannungsquelle



$U_0 = v_b - v_a$

$U_0 = f(t)$

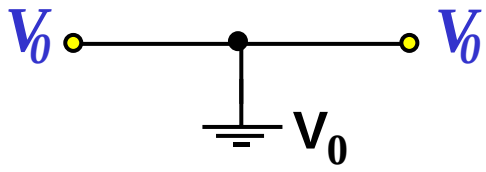
Stromquelle



$u = v_b - v_a$

$I_0 = f(t)$

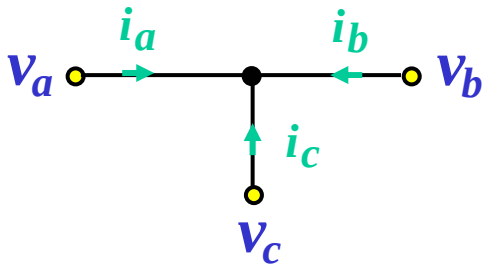
Erdung



$V_0 = 0$

Kirchhoffsche Regeln:

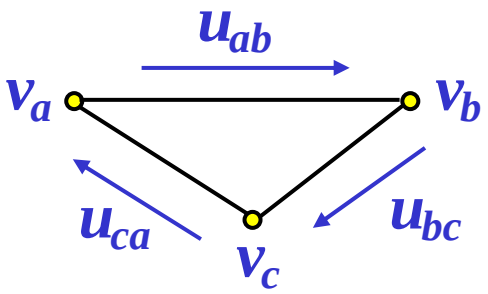
Knotenregel: Die Summe aller Ströme in einem Knotenpunkt ist Null.



$$v_a = v_b = v_c$$

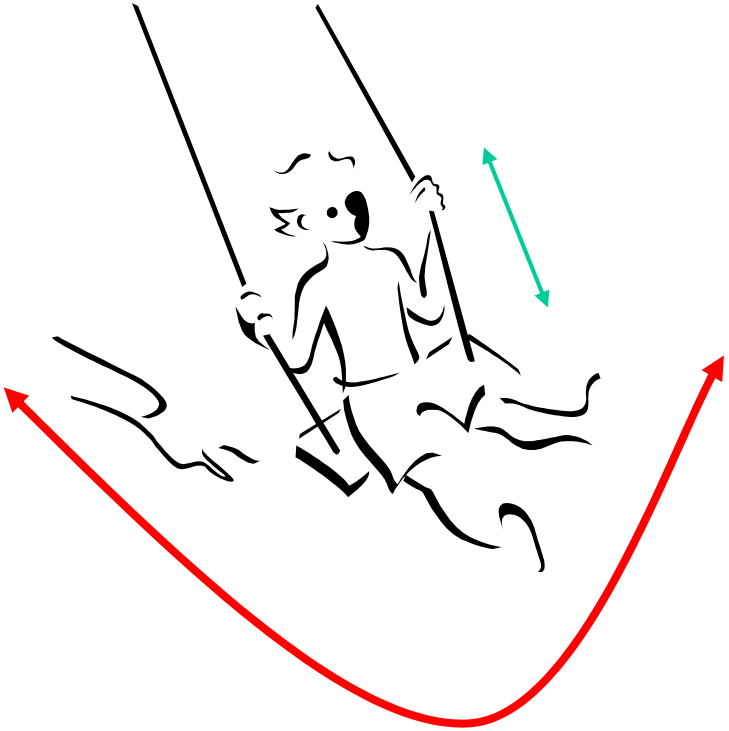
$$i_a + i_b + i_c = 0$$

Maschenenregel: Alle Teilspannungen eines Umlaufs bzw. einer Masche in einem elektrischen Netzwerk addieren sich zu Null .



$$u_{ab} + u_{bc} + u_{ca} = 0$$

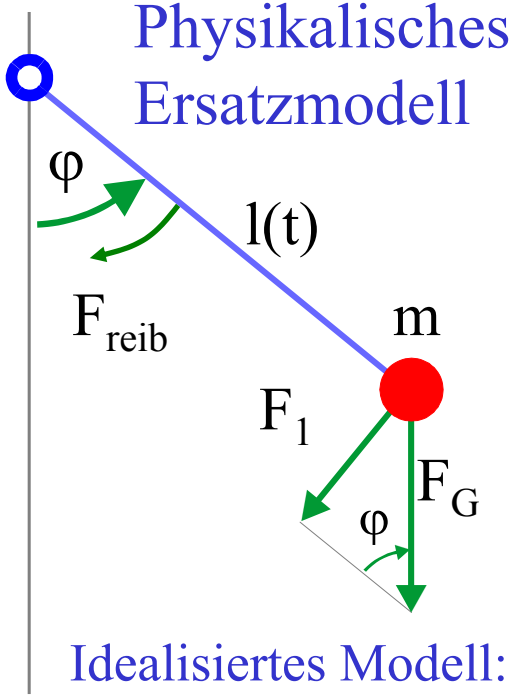
Beispiel: Schaukel (aus einem Skript von Prof. Wiechert, Uni Siegen)



Physikalischer Vorgang, in den zahlreiche Parameter einfließen, z.B.

- Gewicht und Form des/der Schaukelnden
- Materialeigenschaften der Schaukel
- Aufhängung der Schaukel

(Vereinfachtes) Physikalisches Ersatzmodell



Idealisiertes Modell:

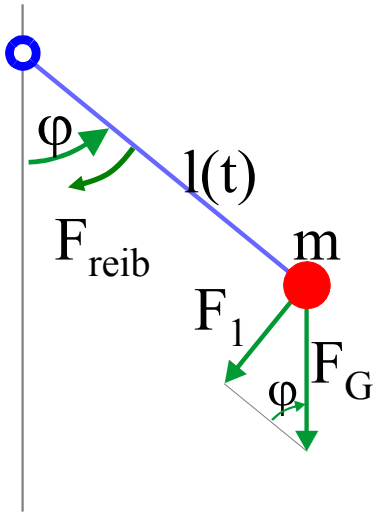
- so einfach wie möglich
- so realitätsnah wie nötig

immer unter Berücksichtigung der Zielstellung (hier Nachbildung der Position der Schaukel)

Vereinfachende Annahmen:

- Masse m ist punktförmig am Ende der Schaukel (dadurch entsteht eine gewöhnliche Diff.Gl.)
- Massepunkt ändert sich mit der Bewegung des Schaukelnden (dargestellt durch Länge l(t))
- Reibung hängt linear von der Geschwindigkeit ab (eigentlich komplexer nichtlinearer Vorgang)

Annahmen müssten eigentlich validiert werden!



Mathematisches Modell:

Gewichtskraft: $F_G = m \cdot g$ (g ist die Erdbeschleunigung)

Nach dem Kräfteparallelogramm: $F_1(t) = \sin(\varphi) \cdot F_G$

Winkelgeschwindigkeit: $\omega = \frac{d\varphi}{dt}$

Reibungskraft: $F_{reib} = -d \cdot \omega$

Trägheitsmoment: $J = m \cdot l^2(t)$

Drehmoment: $\tau = - l(t) \cdot F_1$

Gesetz von Newton: $$\frac{d(J\omega)}{dt} = \tau + F_{reib}$$

Weiterhin gilt: $$\frac{d(J\omega)}{dt} = \dot{J} \cdot \omega + J \cdot \dot{\omega} \; und \; \dot{J} = 2 \cdot m \cdot l \cdot \dot{l}$$

Eingesetzt ergibt sich:

$$\frac{d(J\omega)}{dt} = 2 \cdot m \cdot l \cdot \dot{l} \cdot \omega + m \cdot l^2 \cdot \dot{\omega} = m \cdot l \cdot \left(2 \cdot \dot{l} \cdot \omega + l \cdot \dot{\omega}\right)$$

Herleitung einer Differentialgleichung in $\dot{\omega}$:

$$\begin{aligned} m \cdot l \cdot \left(2 \cdot \dot{l} \cdot \omega + l \cdot \dot{\omega}\right) &= \tau + F_{reib} & \Leftrightarrow \\ 2 \cdot \dot{l} \cdot \omega + l \cdot \dot{\omega} &= \frac{F_{reib}}{m \cdot l} - \frac{F_1}{m} & \Leftrightarrow \\ \dot{\omega} &= \frac{1}{m \cdot l} \cdot \left(\frac{F_{reib}}{l} - F_1\right) - 2 \cdot \omega \cdot \frac{\dot{l}}{l} \end{aligned}$$

Differentialgleichungssystem zur Beschreibung des Modells

$$\dot{\omega} = \frac{1}{m \cdot l} \cdot \left(\frac{F_{reib}}{l} - F_1 \right) - 2 \cdot \omega \cdot \frac{ld}{l} \qquad \dot{\varphi} = \omega$$

$$F_1 = \sin(\varphi) \cdot m \cdot g \qquad F_{reib} = -d \cdot \omega$$

$$\dot{l} = ld$$

Differentialgleichung mit den Variablen ω und φ und den Hilfsvariablen

Hilfsvariablen

Steuerung von Außen durch Vergrößerung/Verkleinerung der Länge

Anfangszustand definieren durch

- Festlegung des Winkels φ, der Winkelgeschwindigkeit ω und der Länge l (jeweils zum Startzeitpunkt)
- Festlegung der Strategie zur Veränderung von ld üblicherweise ld Funktion von φ und l (Rückkopplung)

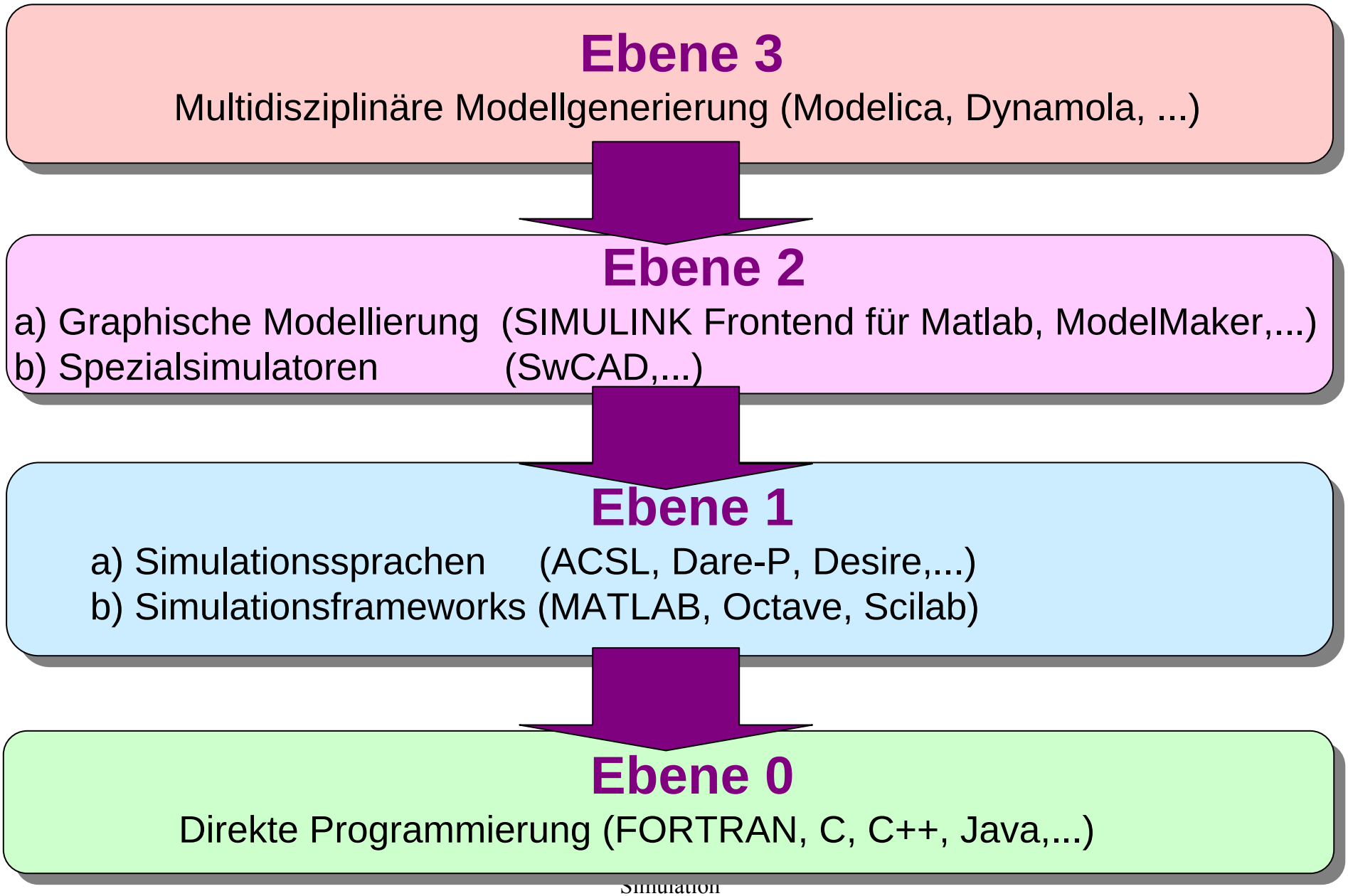
4.2 Beschreibungsansätze für kontinuierliche Probleme

- Vielfalt eher noch größer als bei diskreten Systemen
- Spannbreite von der Bibliothek numerischer Verfahren über Computeralgebrasysteme bis hin zu domänenspezifischen Simulationsumgebungen
- Matlab und zugehörige Bibliotheken hat sich an vielen Stellen als Standard etabliert (hat aber auch gravierende Nachteile)

Mindestanforderungen an Software zur kontinuierlichen Modellierung-Simulation
(über die Eigenschaften moderner Programmiersprachen hinausgehend):

- Darstellung/Beschreibung von Differentialgleichungen
- Numerische Lösung von Differentialgleichungen
- Visualisierung der Resultate

Klassifikation kontinuierlicher Simulationswerkzeuge



Beispiel für eine FORTRAN-basierte Sprache ACSL: (für ld=0)

```
PROGRAM Schaukel
  INITIAL
    constant ...
     m = 70.0, g = 9.81, d = 0.5, l = 2.0, ...
    tend = 100, omega0 = 0, phi0 = 90
   cinterval cint = 0.05
  END $ „of  INITIAL“
  DYNAMIC
    DERIVATIVE
     phid = omega ;
     omegad = (d*ω/(m*l^2) – g*sin(φ)/l)
     phi = integ(phid, phi0) ;
     omega = integ(omegad, omega0) ;
    END $ „of DERIVATIVE“
    termt(t.ge.tend)
  END $ „of DYNAMIC“
END $ „of PROGRAM“
```

Deklaration der Konstanten

Definition der initialen Variablenbelegung

Definition der Simulationsdauer

Ausgabeintervall

Werte der Ableitungen

Integration

Vorgehen und Verwendung von ACSL:

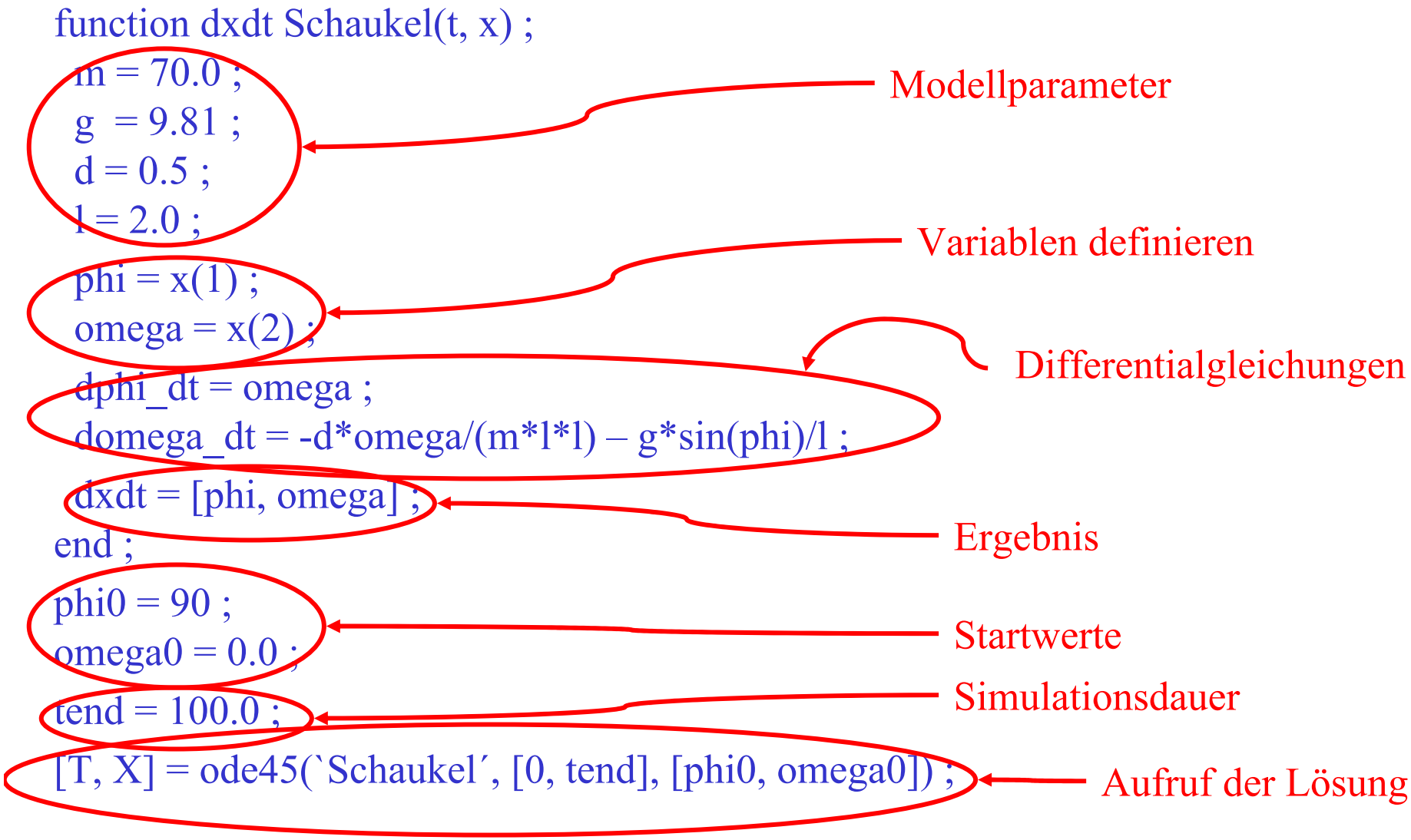
- kommerzielle Programmiersprache, die in unterschiedlichen Programmierumgebungen verfügbar ist, dadurch heute
  - oft mit integrierter graphischer Ausgabeschnittstelle
  - teilweise mit graphischer Eingabeschnittstelle
- wird trotz des Alters und der FORTRAN-Struktur noch heute benutzt
- enthält sehr effiziente Lösungsverfahren
- wird i.d.R. nach FORTRAN übersetzt und mit einer Bibliothek gelinkt

Weitere ähnliche Simulationssprachen existieren z.B. Dare-P oder Desire

Gemeinsames Prinzip:

- Programmiersprachliche Beschreibung der Gleichungen
- Simulationsunterstützung durch integrierte Funktionen

Programmierung in MATLAB
durch Beschreibung der Modellgleichungen (hier für ld=0)



MATLAB erlaubt

- Modellbeschreibung sehr nahe beim mathematischen Modell durch direkte Eingabe der Gleichungen
- Nutzung einer Vielzahl vordefinierter Datenstrukturen
- Nutzung mathematischer Operationen auf komplexen Daten (z.B. Matrixmultiplikation, Matrixinvertierung etc.)
- Nutzung unterschiedlicher Lösung unterschiedlicher Lösungsalgorithmen
- graphische Ausgabe

Durch weite Verbreitung existieren eine Vielzahl von Erweiterungen mit Basismodellen aus unterschiedlichen Anwendungsgebieten

Nachteile von MATLAB:

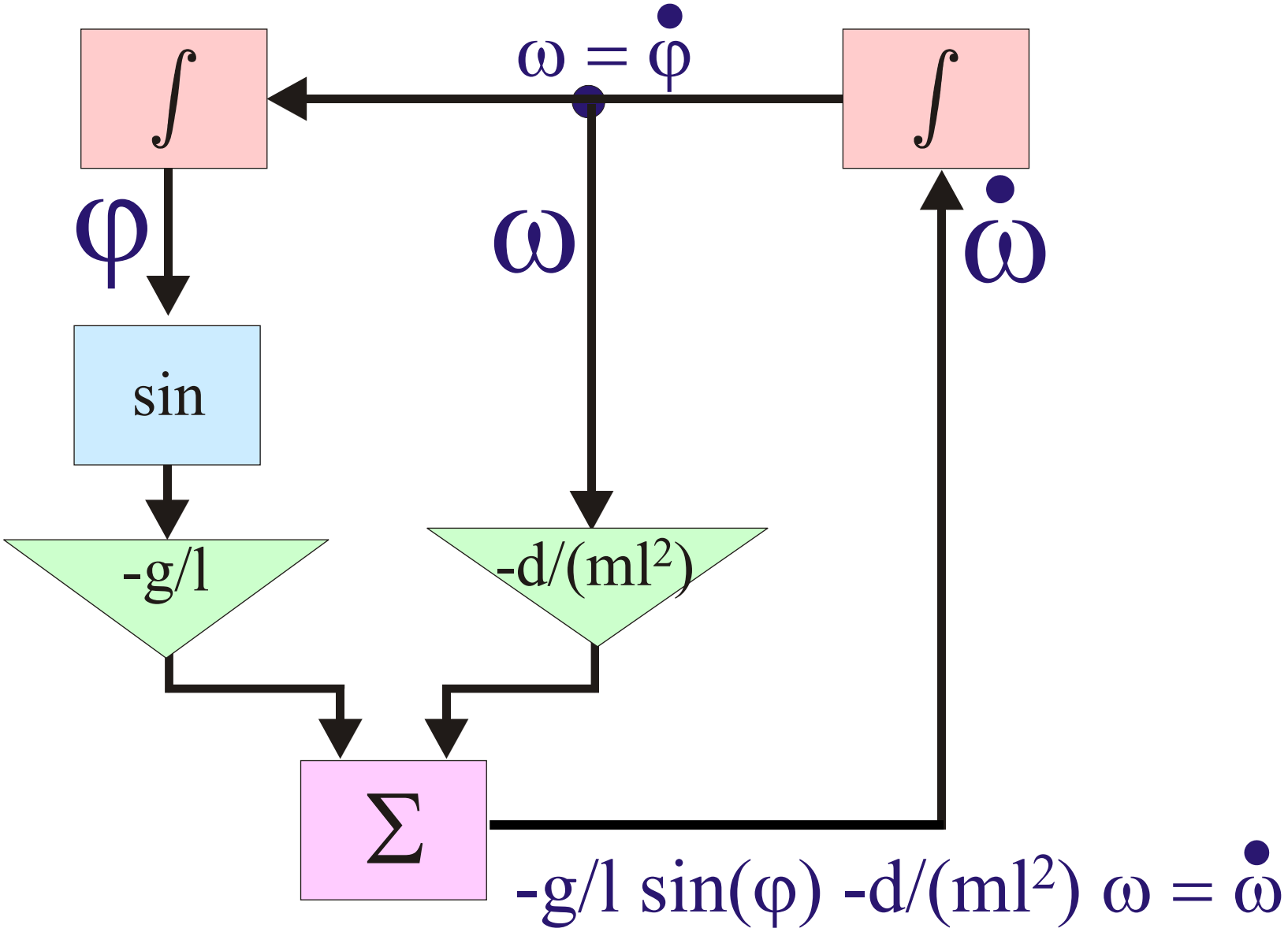
- Kommerziell und damit nicht frei nutzbar (freie Alternativen: Octave oder Scilab (weniger mächtig aber oft ausreichend))
- mangelnde Effizienz

Werkzeuge mit graphischer Eingabeschnittstelle:

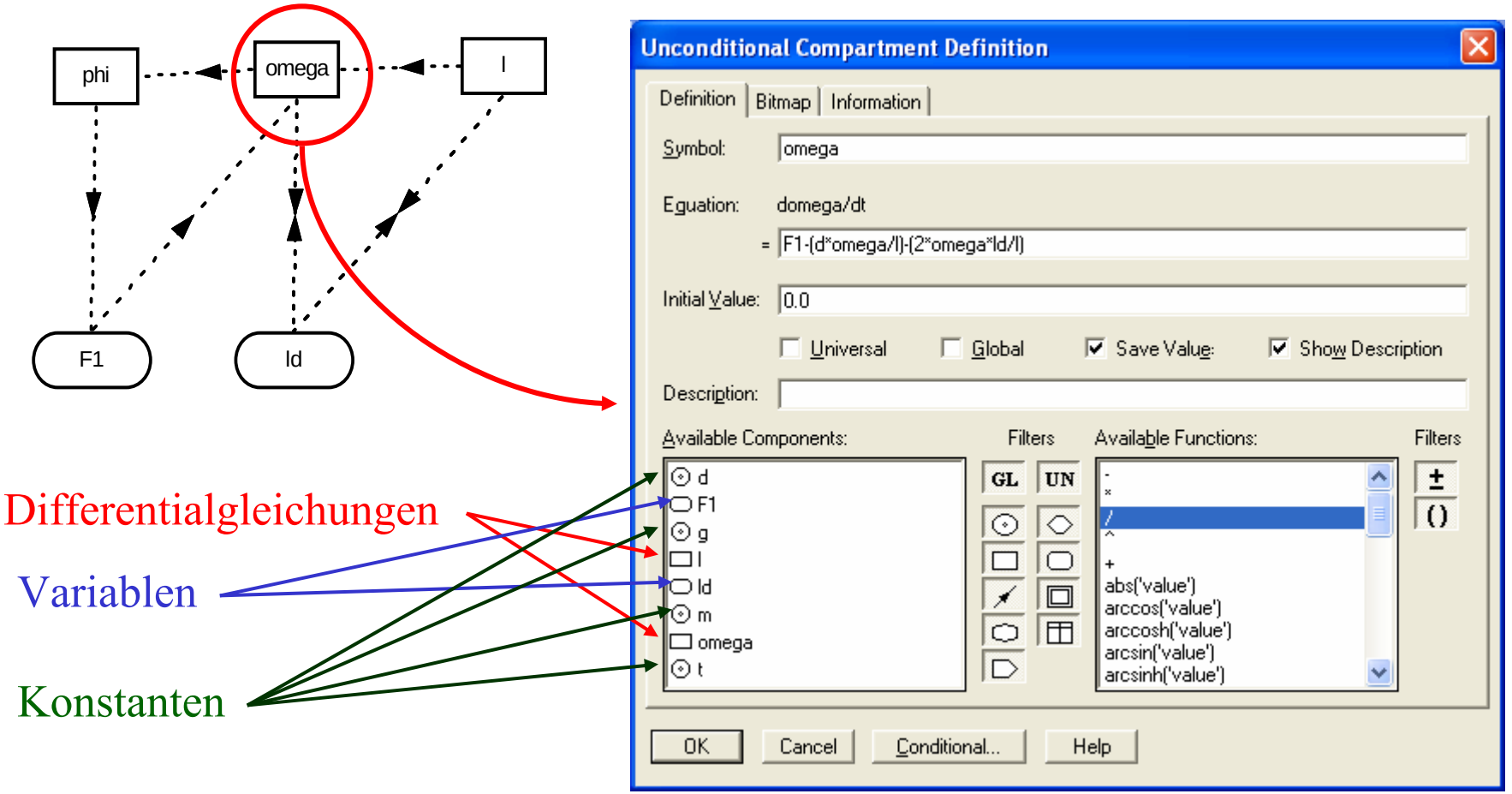
- Blockorientierte Beschreibung der Abhängigkeiten zwischen Variablen und Funktionen
- Je nach Werkzeug Darstellung
  - der Variablen als Blöcke und textuelle Beschreibung der Funktionen
  - der Variablen und Funktionen als Blöcke
- Man kann unterscheidet
  - anwendungsspezifische Werkzeuge mit Symbolen der Anwendung (z.B. elek. Schaltkreise mit Symbolen für Drähte, Wiederstände, Kondensatoren etc)
  - allgemeine Darstellungen mit neutralen Symbolen

  Methodik zur Beschreibung existiert nur in einzelnen Anwendungsgebieten, keine allgemein akzeptierte Modellwelt
- Werkzeuge z.T. als Frontends für textuelle Werkzeuge (z.B. Simulink für MATLAB, Scicos für Scilab)

Funktionsorientierte Beschreibung in Simulink



Variablenorientierte Beschreibung in ModelMaker

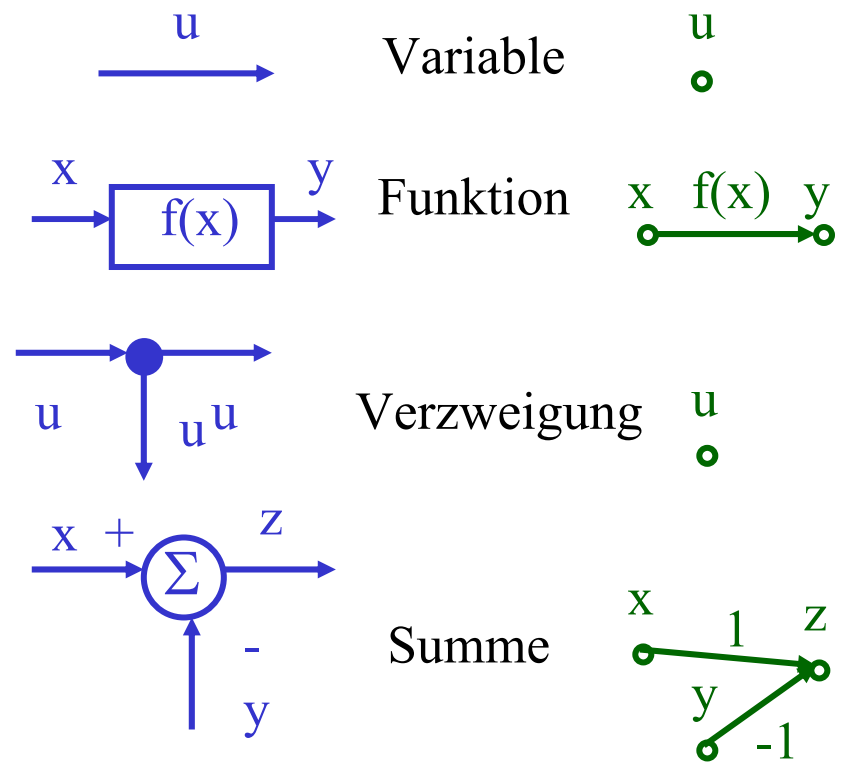


Sichtbarkeit auf Eingabegrößen beschränkt!

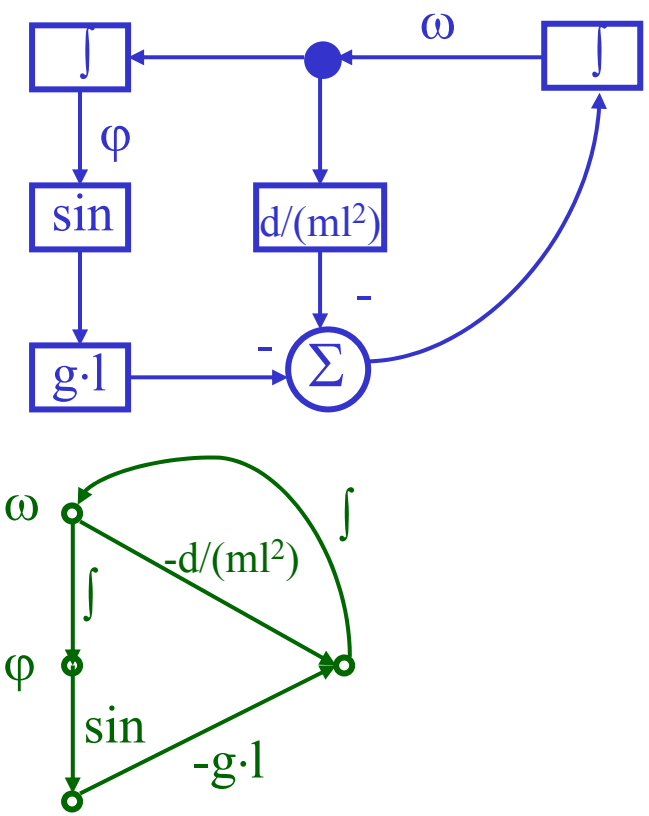
Basis der graphischen Beschreibungen

Blockdiagramme

Flussgraphen



Schaukel-Beispiel



- Basiselemente reichen zur Beschreibung einfacher Systeme aus
- Erweiterung der graphischen Modellierung durch Bond-Graphen (hier nicht behandelt)

Objektorientierte Beschreibung in Modelica

Modelica ist eine objektorientierte Sprache auf Ebene 3

- Relativ neue Sprache Version 1.0 1997, aktuell Version 2.2.1 (April 2006)
- basiert auf der Arbeit einer internationalen Gruppe
  - aus Universitäten, Forschungseinrichtungen (z.B. Fraunhofer, Linköping University, DLR, ...)
  - und Industrie (z.B. Dynasim, Ford, ABB, ...)

Entwurf einer Sprache

- mit objektorientierter Struktur (Templates, Mehrfachvererbung)
- zur Beschreibung physikalischer Systeme aus unterschiedlichen Anwendungsgebieten
- zur nicht-kausalen Modellierung über Differentialgleichungen oder algebraischen Gleichungen
  - nicht kausale Modellierung basiert auf Gleichungen und nicht auf Zuweisungen (!)
  - Kausalität tritt erst bei der Lösung auf

Ziel: Schaffung eines quasi Standards für die Beschreibung kontinuierlicher Systeme

Objektorientierte mathematische Modellierung basiert auf

Objekten: Sammlung von Variablen, Gleichungen, Funktionen, die zu einer Sicht (Abstraktion) gehören

Klassen: Templates aus denen Objekte generiert werden können

Vererbung: Subklassen erben Gleichungen, Funktionen und Variablen von den Eltern

Objektorientierung bietet die Möglichkeit Klassenbibliotheken für unterschiedliche Anwendungsgebiete zu entwickeln durch Beschreibung von Komponenten als Erweiterung von Basiskomponenten $\Rightarrow$ Bibliothek generischer Basiskomponenten

Modelica Programm ist eine deklarative Beschreibung des mathematischen Modells, welches

- analysiert werden kann und
- auf Basis von Gleichheiten Schlüsse erlaubt

Ein kleines Beispiel aus der Elektrotechnik

Viele elektrische Elemente haben zwei Anschlüsse, deshalb definieren wir zuerst einen Anschluss

```
connector Pin
  Voltage v ;
  flow Current i ;
end Pin ;
```

} Typen Voltage und Current sind zu definieren z.B. als real

connector-Klassen können mittels der connect-Anweisung verbunden werden

```
connect (Pin1, Pin2)
```

definiert implizit die Gleichungen

Pin1.v = Pin2.v und Pin1.i + Pin2.i = 0

Spannungen an beiden verbundenen Teilen gleich, Ströme addieren sich zu 0 (Gesetz von Kirchhoff)

Ähnliche Zusammenhänge existieren in anderen Anwendungsgebieten (z.B. Kräfte in der Mechanik)

Definition einer Schnittstellenklasse mit zwei Anschlüssen:

Graphische Darstellung

Klasse TwoPin ist partiell, da sie durch zusätzliche Gleichungen für bestimmte Komponenten ergänzt werden muss!

Modelica Definition:

```
partial class TwoPin
  Pin p, n ;
  Voltage v ;
  Current i ;
equation
  v = p.v – n.v ;
  0 = p.i + n.i ;
  i  = p.i ;
end TwoPin ;
```

Definition eines Widerstandes:

```
class Resistor
  extends TwoPin ;
  parameter Real R(unit = „Ohm“) ;
equation
  R · i = v ;
end Resistor ;
```

Weitere Elemente

Wechselspannungsquelle:

```
class VsourceAC
   extends TwoPin ;
   parameter Voltage VA = 220 ;
   parameter Real f (unit = „Hz“) = 50 ;
  constant Real PI = 3.14159 ;
 equation
   v = VA · sin(2 · PI · f · time) ;
end VsourceVA ;
```

time ist globale Systemvariable, welche die Modellzeit angibt

Kondensator:

```
class Capacitor
   extends TwoPin ;
   parameter Real C (unit = „F“) = 50 ;
 equation
   C · der(v) = i ;
end Capacitor ;
```

der(v) liefert die Ableitung, so dass $C \cdot \dot{v}(t) = i(t)$ für alle t

Spule:

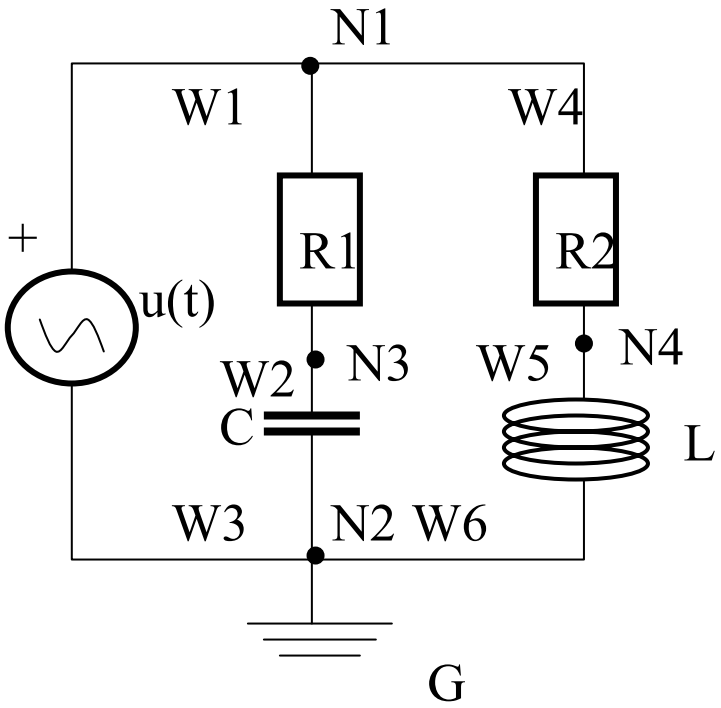
```
class Inductor
   extends TwoPin ;
   parameter Real L (unit = „H“) = 50 ;
 equation
   L · der(i) = v ;
end Inductor ;
```

Erdung:

```
class ground
   Pin p ;
 equation
   p.v = 0 ;
end Ground ;
```

- Durch Erweiterung der Klassen lassen sich beliebige komplexe Strukturen inkrementell definieren
  (auch in anderen Anwendungsgebieten)
- Alle Abhängigkeiten sind Gleichungen und keine Zuweisungen
  Also R · i = v kann als i := v / R oder v := R · i interpretiert werden

Beispielschaltung




```
class circuit                 equation
  Restistor R1(R=10) ;          connect(AC.p, R1.p) ;
  Capacitor C(C=0.01) ;         connect(R1.n, C.p) ;
  Resistor R2(R=100) ;          connect(C.n, AC.n) ;
  Inductor L(L=0.1) ;           connect(R1.p, R2.p);
 VsourceAC AC ;                 connect(R2.n, L.p) ;
  Ground G ;                    connect(L.n, C.n) ;
                                connect(AC.n, G.p) ;
                              end circuit ;
```

Resultierende Gleichungen

AC	0 = Ac.p.i + AC.n.i	L	0 = L.p.i + L.n.i
	AC.v = AC.p.v - AC.n.v		L.v = L.p.v – L.n.v
	AC.i = AC.p.i		L.i = L.p.i
	AC.v = AC.V.A ·		L.v = L.l · der(L.i)
	sin(2 · PI · AC.f · time)	G	G.p.v = 0
R1	0 = R1.p.i + R1.n.i	W1	R1.p.v = AC.p.v
	R1.v = R1.p.v – R1.n.v	W2	C.p.v = R1.n.v
	R1.i = R1.p.i	W3	AC.n.v = C.n.v
	R1.v = R1.R · R1.i	W4	R2.p.v = R1.p.v
R2	0 = R2.p.i + R2.n.i	W5	L.p.v = R2.n.v
	R2.v = R2.p.v – R2.n.v	W6	L.n.v = C.n.v
	R2.i = R2.p.i	W7	G.p.v = AC.n.v
	R2.v = R2.R · R2.i		
C	0 = C.p.i + C.n.i	N1	0 = AC.p.i + R1.p.i + R2.p.i
	C.v = C.p.v – C.n.v	N2	0 = C.n.i + G.i + AC.n.i + L.n.i
	C.i = C.p.i	N3	0 = R1.n.i + C.p.i
	C.i = C.C · der(C.v)	N4	0 = R2.n.i + L.p.i

Vielzahl von Gleichungen (z.T. redundant)

Schritte zur Lösung/Analyse

- Sortierung der Gleichungen
- Elimination überflüssiger Gleichungen
- Falls nötig symbolische Gleichungslösung
- Lösung des resultierenden Differentialgleichungssystems
  - u.U. symbolische Lösung
  - i.d.R. Regel numerische Simulation

Philosophie von Modelica:

- Sprache zur Beschreibung komplexer Modelle
- Programmiersysteme zur Erstellung von Modellen und zur Syntaxüberpüfung (teilweise graphische Eingabe)
- Bibliotheken oft mit graphischer Schnittstelle für Anwendungsszenarien (Elektronische Schaltungen, Mechanik, Hydraulik, ...)
- Übersetzung von Modelica Programmen in Modelle anderer Simulatoren (z.B. Mathematica, Dynasim)
- Open Source Modelica Projekt existiert, bisher aber nur wenige Programme verfügbar (insbesondere keine vollständige Modelica-Umgebung)

4.3 Analysetechniken für kontinuierliche Systeme

Allgemeine Form eines kontinuierlichen Systems: $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t)$

Wir betrachten den einfachen Sonderfall: $\dot{\mathbf{x}} = f(\mathbf{x})$

- Vektor **u** kann durch Erweiterung während der Lösung berücksichtigt werden
- Oft gilt für den Lösungsvektor **p** = g(**x**), so dass Beobachtung von **x** ausreicht

Beschreibung des Modells kann (und wird oft) allgemeiner sein, da

1. höhere Ableitungen vorkommen
2. die Zeit t explizit vorkommt
3. nur Beziehungen zwischen Variablen beschrieben werden

In diesen Fällen muss zuerst die Modellbeschreibung (möglichst automatisch) so transformiert werden, dass die obige Darstellung entsteht

Ersetzung höherer Ableitungen durch neue Variablen

Differentialgleichung der Ordnung n wird durch n Differentialgleichungen erster Ordnung ersetzt

Beispiel: $x^{(n)} = f(x, \dot{x}, \ddot{x}, \ldots, x^{(n-1)})$ mit $x^{(i)}(0) = C_i$

wird ersetzt durch

$$\left.\begin{aligned} \dot{x}_0 &= x_1 \\ \dot{x}_1 &= x_2 \\ &\vdots \\ \dot{x}_{n-2} &= x_{n-1} \\ \dot{x}_{n-1} &= f(x_0, \ldots, x_{n-2}) \end{aligned}\right\}$$

System erster Ordnung mit initialen Werten $x_i = C_i$

Darstellung der Zeit durch Einführung einer expliziten Variablen

$\dot{\mathbf{x}} = f(\mathbf{x}, t)$ mit n-1 Variablen $\Rightarrow$

Einführung von x_n mit $\dot{x}_n = \frac{dx_n}{dt} = 1$ und $x_n(0) = 0$

Modellbeschreibung (wie z.B. auf Folie 34/35) liefert eine Menge von Beziehungen zwischen den Variablen

- Beziehungen sind als Gleichungen (und nicht als Zuweisungen) zu verstehen
- In der Regel sind deutlich mehr Gleichungen als Variablen vorhanden

$\Rightarrow$ Gleichungen sind so zu transformieren, dass Differentialgleichungssystem in der gewünschten Form entsteht

Transformationsschritte (erst horizontale Sortierung):

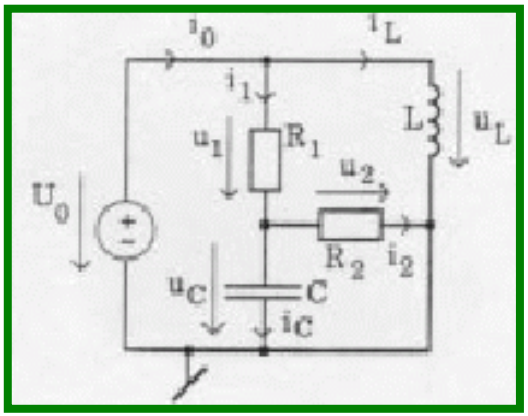
1. Zustandsvariablen (die in Differentialgleichungen auftauchen oder als Funktionen definiert sind) können als bekannt vorausgesetzt werden (ergeben sich später aus den Differentialgleichungen)
2. Variablen, die in Gleichungen vorkommen, die neben der Variable nur eine Zustandsvariable enthält werden aus dieser Gleichung berechnet
3. Variablen, die in nur einer Gleichung vorkommen, müssen aus dieser Gleichung berechnet werden

Schritte 1.-3. werden rekursiv angewendet

dann vertikale Sortierung

4. Gleichungen so sortieren, dass Variablen Werte zugewiesen werden, bevor sie benutzt werden
5. Falls noch nicht alle Werte bekannt $\Rightarrow$ symbolische Gleichungslösung

Bespiel (aus Cellier 1991):



5 Elemente in der Schaltung jeweils
Berechnung von Strom und Spannung ⇒
10 Werte sind zu berechnen ⇒
10 Gleichungen werden benötigt

Vollständiger Satz von Gleichungen (redundante Beziehungen wurden bereits entfernt)

$U_0 = f(t)$	$i_0 = i_1 + i_L$
$u_1 = R_1 \cdot i_1$	$i_1 = i_2 + i_C$
$u_2 = R_2 \cdot i_2$	$U_0 = u_1 + u_C$
$i_C = C \cdot du_C/dt$	$u_C = u_2$
$u_L = L \cdot di_L/dt$	$u_L = u_1 + u_2$

○ bekannt nach Regel 1.

○ berechnen nach Regel 2.

○ berechnen nach Regel 3.

◌ bekannt nach einmaliger Anwendung der Regel

Resultierende Gleichungen nach mehrfacher Anwendung der Regeln und Umformung der Gleichungen, so dass Zuweisungen definiert werden

$U_0 = f(t)$	$i_0 = i_1 + i_L$
$i_1 = u_1 / R_1$	$i_C = i_1 - i_2$
$i_2 = u_2 / R_2$	$u_1 = U_0 - u_C$
$du_C/dt = i_C / C$	$u_2 = u_C$
$di_L/dt = u_L / L$	$u_L = u_1 + u_2$

vertikale Sortierung

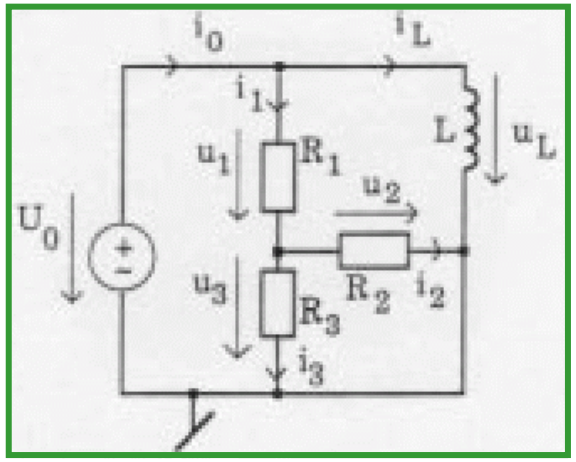
$U_0 = f(t)$	$i_2 = u_2 / R_2$
$u_1 = U_0 - u_C$	$i_C = i_1 - i_2$
$i_1 = u_1 / R_1$	$u_L = u_1 + u_2$
$i_0 = i_1 + i_L$	$du_C/dt = i_C / C$
$u_2 = u_C$	$di_L/dt = u_L / L$

Resultierendes Differentialgleichungssystem mit 2 Variablen (i_L und u_C), die restlichen Variablenwerte ergeben sich aus den Konstanten.

$$
\begin{aligned}
du_C/dt &= i_C / C \\
&= (i_1 - i_2) / C \\
&= i_1 / C - i_2 / C \\
&= u_1 / (R_1 \cdot C) - u_2 / (R_2 \cdot C) \\
&= (U_0 - u_C) / (R_1 \cdot C) - u_C / (R_2 \cdot C)
\end{aligned}
$$

$$
\begin{aligned}
di_L/dt &= u_L / L \\
&= (u_1 + u_2) / L \\
&= u_1 / L + u_2 / L \\
&= (U_0 - u_C) / L + u_C / L \\
&= U_0 / L
\end{aligned}
$$

Ein weiteres Bespiel (aus Cellier 1991):



5 Elemente in der Schaltung jeweils
Berechnung von Strom und Spannung ⇒
10 Werte sind zu berechnen ⇒
10 Gleichungen werden benötigt

Vollständiger Satz von Gleichungen (redundante Beziehungen wurden bereits entfernt)

$U_0 = f(t)$	$i_0 = i_1 + i_L$
$u_1 = R_1 \cdot i_1$	$i_1 = i_2 + i_3$
$u_2 = R_2 \cdot i_2$	$U_0 = u_1 + u_3$
$u_3 = R_3 \cdot i_3$	$u_3 = u_2$
$u_L = L \cdot di_L/dt$	$u_L = u_1 + u_2$

○ Bekannte Variablen nach rekursiver Anwendung der Regeln

○ zu berechnen nach Regel 3.

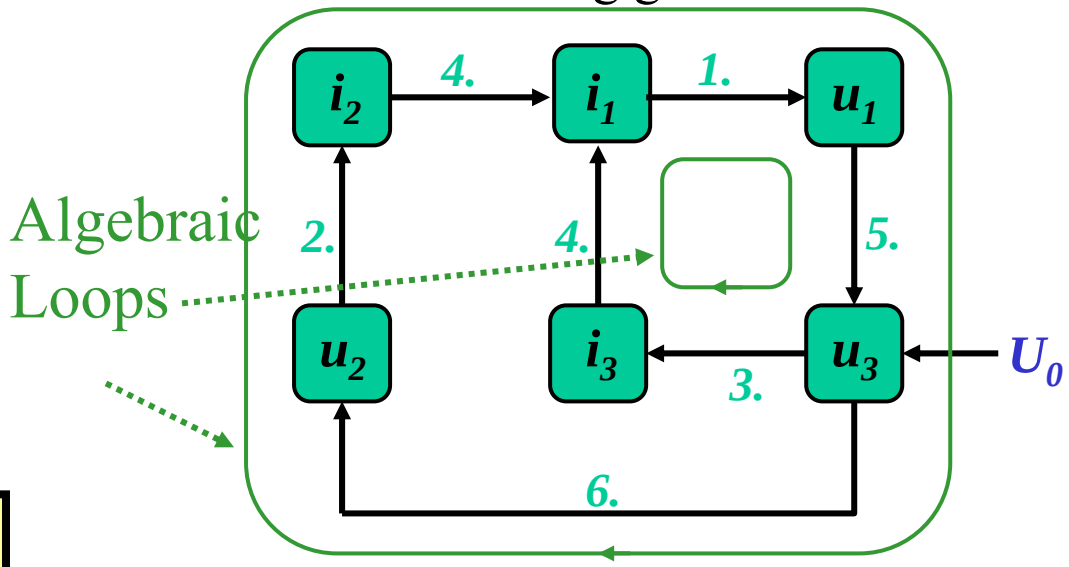
- alle Gleichungen enthalten mindestens zwei unbekannte Variablen und
- jede unbekannte Variable kommt in mindestens zwei Gleichungen vor

„*algebraic loop*“

Verbleibende Gleichungen

1. $u_1 = R_1 \cdot i_1$	4. $i_1 = i_2 + i_3$
2. $u_2 = R_2 \cdot i_2$	5. $U_0 = u_1 + u_3$
3. $u_3 = R_3 \cdot i_3$	6. $u_3 = u_2$

Struktur der Abhängigkeit



Variablen freistellen (jede Variable einmal auf linker Seite)

1. $u_1 = i_1 / R_1$	4. $i_1 = i_2 + i_3$
2. $i_2 = u_2 / R_2$	5. $u_3 = u_1 + U_0$
3. $i_3 = u_3 / R_3$	6. $u_2 = u_3$

Zyklen können nicht durch Umordnen der Variablen gebrochen werden!

Auflösung der Schleifen durch symbolische Gleichungslösung

Hier konkret:

- Auswahl einer Variablen, die auf beiden Schleifen vorkommet (z.B. i_1)
- Annahme, dass diese Variable bekannt sei
- Auflösung des restlichen Gleichungssystems
- Berechnung der ausgewählten Variablen aus der Gleichungslösung

Lösung des Gleichungssystems bei bekanntem i_1:

$u_1 = R_1 \cdot i_1$	$u_2 = u_3$
$u_3 = U_0 - u_1$	$i_2 = u_2 / R_2$
$i_3 = u_3 / R_3$	

Berechnung von i_1:

$$
\begin{aligned}
i_1 &= i_2 + i_3 \\
&= u_2 / R_2 + u_3 / R_3 \\
&= u_3 / R_2 + u_3 / R_3 \\
&= ((R_2 + R_3) / (R_2 \cdot R_3)) \cdot u_3 \\
&= ((R_2 + R_3) / (R_2 \cdot R_3)) \cdot (U_0 - u_1) \\
&= ((R_2 + R_3) / (R_2 \cdot R_3)) \cdot (U_0 - R_1 \cdot i_1)
\end{aligned}
$$

$$
i_1 = \frac{R_2 + R_3}{R_1 R_2 + R_1 R_3 + R_2 R_3} \cdot U_0
$$

Vorgehen im allgemeinen Fall :

- Algorithmen (z.B. von Tarjan) existieren,
  - um Schleifen zu identifizieren und
  - Variablen entsprechend umzuordnen
- Lösung der resultierenden Gleichungssysteme
  - für lineare Gleichungssysteme mit wenigen Variablen symbolische Lösung
  - für lineare Gleichungssysteme mit vielen Variablen erfolgt eine numerische Lösung (mittels Gauß-Elimination oder iterativen Verfahren)
  - für nichtlineare Systeme numerische Lösung oft mittels Newton-Iteration

Etwas mehr am Ende des Abschnitts!

Kontinuierliche Simulation zur Berechnung von $\mathbf{x}(t)$
auf Basis der Information $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t)$ und $\mathbf{x}(0) = \mathbf{x}_0$

Einfachster Lösungsansatz Euler-Verfahren,
d.h. Diskretisierung mit fester Schrittweite Δt:
Sei $\mathbf{x}_k = \mathbf{x}(k \cdot \Delta t)$ und $\mathbf{u}_k = \mathbf{u}(k \cdot \Delta t)$ sowie $t = K \cdot \Delta t$
(damit ist $\mathbf{x}_K$ zu berechnen)
Die Approximation

$$\int_{k \cdot \Delta t}^{(k+1) \cdot \Delta t} f(\mathbf{x}(t), \mathbf{u}(t), t) dt \approx \Delta t \cdot f(\mathbf{x}_k, \mathbf{u}_k, k \cdot \Delta t)$$

liefert

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \cdot f(\mathbf{x}_k, \mathbf{u}_k, k \cdot \Delta t) \qquad \text{(Linearisierung der Funktion)}$$

Ergebnis kann einfach in K Schritten berechnet werden
Fehler durch die Linearisierung der Funktion

Freiheitsgrad: Wahl von Δt

Euler-Verfahren: Einfach aber zu ungenau!

Euler Verfahren ist eine sehr simple Methode der numerischen Simulation ⇒
viele Ansätze der Verbesserung existieren

Beispiele:

1. Berücksichtigung bereits vorher berechneter Werte
   - Einschritt (nur aktueller Wert)
   - Mehrschritt (auch weiter zurückliegende Werte)
2. Einbeziehung zukünftiger Werte
   - Explizite Verfahren (keine Einbeziehung)
   - Implizite Verfahren (Einbeziehung)
3. Konvergenzordnung
4. Schrittweitensteuerung
   - feste Schrittweite
   - dynamische Schrittweite
5. Extrapolationsansätze
6. Ordnungssteuerung
   - feste Ordnung
   - dynamische Ordnung

Euler Verfahren: explizites Einschritt-Verfahren 1. Ordnung mit fester Schrittweite

Kompromiss zwischen Aufwand und Genauigkeit in Abhängigkeit der zu analysierenden Funktion

Kurzer Einschub: Taylor-Reihe einer Funktion

f(x) sei n+1 mal differenzierbar in $[x_0-\Delta, x_0+\Delta]$ ($\Delta>0$)

Ziel: Approximation von $f(x+\Delta)$ unter Ausnutzung von $f(x)$, $d/dx\ f(x)$, $d^2/dx^2\ f(x)$,...

Taylor-Reihe $$f(x)=\sum_{i=0}^{n}\frac{f^{(i)}(x_0)}{i!}(x-x_0)^i+R_n(x)$$

Restglied $$R_n(x)=\frac{f^{(n+1)}(x_0+\eta(x-x_0))}{(n+1)!}(x-x_0)^{n+1} \text{ mit } \eta\in(0,1)$$

Interpretation:

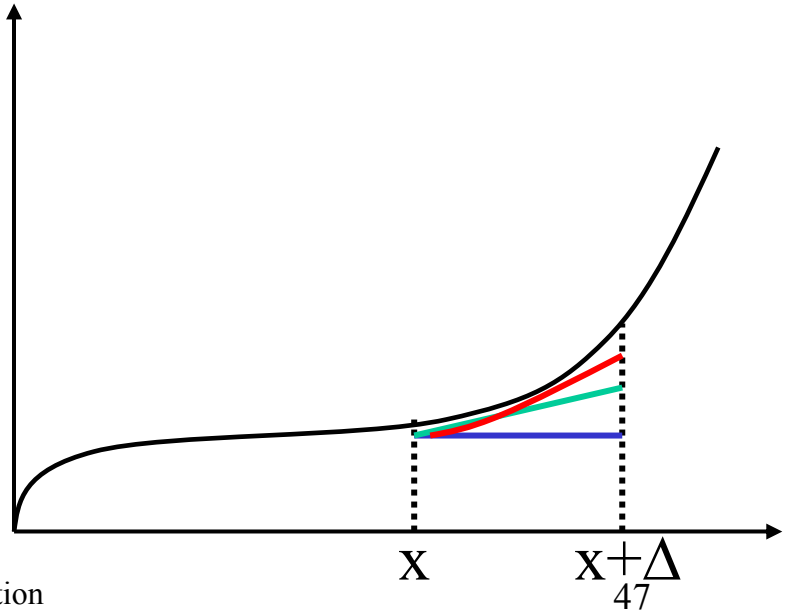
0. Ordnung $f(x+\Delta)\approx f(x)$

1. Ordnung

$$f(x+\Delta)\approx f(x)+\Delta\cdot f^{(1)}(x)$$

2. Ordnung

$$f(x+\Delta)\approx f(x)+\Delta\cdot f^{(1)}(x)+0.5\Delta\cdot f^{(2)}(x)$$



Beispiele mit 1. Variable

$$f(x) = \sin(x)$$

$$f^{(1)}(x) = \cos(x)$$

$$f^{(2)}(x) = -\sin(x)$$

....

$$f(x_0 + \Delta) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} \Delta^n$$
$$= \sin(x_0) + \Delta \cdot \cos(x_0) - \frac{\Delta^2}{2}\sin(x_0) - \frac{\Delta^3}{6}\cos(x_0) + ..$$

Konvergenz für alle x_0 und Δ
(global konvergent, analytisch)

$$f(x) = 1/x$$

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

$$f(x_0 + \Delta) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} \Delta^n$$
$$= \sum_{n=0}^{\infty} (-1)^n \frac{\Delta^n}{(x_0)^{n+1}}$$
$$= \frac{1}{x_0} \sum_{n=0}^{\infty} \left(\frac{-\Delta}{x_0}\right)^n$$
$$= \frac{1}{x_0} \cdot \frac{1}{1 + \frac{\Delta}{x_0}}$$

Falls $\Delta < x_0$

Konzept auf mehrere Funktionen und Variablen übertragbar!

Verbesserung der Verfahrensordnung

Wir betrachten zur Vereinfachung den Fall $\dot{\mathbf{x}} = f(\mathbf{x})$

Entwicklung der Taylorreihe im Punkt t:

$$T_n^k(\Delta t) = f(\mathbf{x}_k) + \Delta t \cdot f'(\mathbf{x}_k) + \frac{\Delta t^2}{2} \cdot f''(\mathbf{x}_k) + \ldots + \frac{\Delta t^n}{n!} \cdot f^{(n)}(\mathbf{x}_k) + O(\Delta t^{n+1})$$

und es gilt für viele Funktionen $\lim_{n\to\infty} \mathbf{x}_k + T_n^k(\Delta t) = \mathbf{x}_{k+1}$

falls Δt genügend klein

Konvergenz hängt von Δt und der Funktion ab

Beispiele siehe vorherige Folie

Pragmatische Abschätzung des Fehlers ε_D:

Einzelschrittfehler $\varepsilon_{ED} = \left|\mathbf{x}_{k+1} - \mathbf{x}_k - T_n^k(\Delta t)\right| \in O(\Delta t^{n+1})$

für genügend kleine Δt

Falls $\varepsilon_{ED} \in O(\Delta t^{n+1})$ dann geht man davon aus, dass $\varepsilon_D \in O(\Delta t^n)$

Verfahren höherer Ordnung für lineare Systeme:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} \text{ mit } \mathbf{x}(0) = \mathbf{x_0}$$

$$\ddot{\mathbf{x}} = \frac{d}{dt}(\mathbf{A} \cdot \mathbf{x}) = \mathbf{A} \cdot \dot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x}$$

$$\dddot{\mathbf{x}} = \frac{d}{dt}(\mathbf{A}^2 \cdot \mathbf{x}) = \mathbf{A}^2 \cdot \dot{\mathbf{x}} = \mathbf{A}^3 \cdot \mathbf{x}$$

...

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \cdot \mathbf{A} \cdot \mathbf{x}_k + \frac{\Delta t^2}{2} \cdot \mathbf{A}^2 \cdot \mathbf{x}_k + \ldots + \frac{\Delta t^n}{n!} \cdot \mathbf{A}^n \cdot \mathbf{x}_k + O(\Delta t^{n+1})$$

Die exakte Lösung lautet $\mathbf{x}(T) = \mathbf{x}(0) \cdot e^{\Delta t \cdot \mathbf{A}}$

Berechnungsaufwand $\Delta t/T$ Schritte mit jeweils n Vektor-Matrix-Multiplikationen

Auswirkungen der Fehlerordnung:

- Soll bei einem Verfahren der Ordnung 1 die Genauigkeit um eine Dezimalstelle verbessert werden, muss die Schrittzahl verzehnfacht werden
- Bei einem Verfahren der Ordnung 2 muss die Schrittzahl nur verdreifacht werden (da $3 \approx 10^{1/2}$)
- Gebräuchliche Runge-Kutta Verfahren der Ordnung 4 erzielt bei Schrittweitenhalbierung eine $2^4 = 16$-fache Genauigkeit

Vorsicht: Verfahrensordnung berücksichtigt keine konstanten Faktoren

Wie im Fall diskreter Systeme, ist die stationäre Lösung oft von Interesse.
Es gilt im stationären Fall $\dot{\mathbf{x}} = 0 \Rightarrow \mathbf{A} \cdot \mathbf{x} = \mathbf{0}$
also Lösung eines linearen Gleichungssystems!

Verfahren höherer Ordnung für allgemeine Systeme:

$$\dot{\mathbf{x}} = f(\mathbf{x}) \text{ mit } \mathbf{x}(0) = \mathbf{x_0}$$

Zur Bestimmung der zweiten Ableitung wird die verallgemeinerte Kettenregel benötigt:

Sei f: $\partial^n \rightarrow \partial^m$ total differenzierbar und
$g_i: \partial_+ \rightarrow$ (i=1,..n) im Punkt t_0 differenzierbar, dann gilt

$$\left.\frac{df(g_1(t),\ldots,g_n(t))}{dt}\right|_{t=t_0} = \sum_{i=1}^{n} \frac{\partial f(x_1^0,\ldots,x_n^0)}{\partial x_i} g_i'(t_0) \text{ mit } x_i^0 = g_i(t_0)$$

Angewendet auf die obige Darstellung liefert dies

$$\ddot{\mathbf{x}} = \frac{d}{dt}\left[f(\mathbf{x})\right] = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) \cdot \dot{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) \cdot f(\mathbf{x})$$

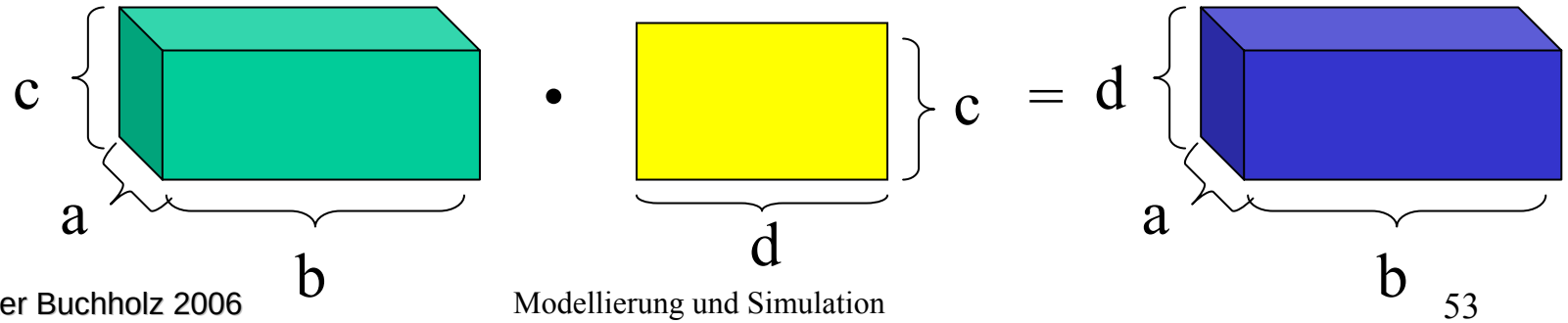
Struktur der Ableitungen:

$$f(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{pmatrix} \qquad \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

Dritte Ableitung liefert ein dreidimensionale Matrix mit $m \cdot n^2$ Elementen
Beispiel 5 Funktionen mit jeweils 5 Variablen $\Rightarrow$
4. Ableitung mit 625 Elementen

Multiplikation höherer Ableitungen kann über mehrdimensionale Matrixmultiplikation als Erweiterung der bekannten Matrixmultiplikation erfolgen

Skizze



Weitere Ableitungen:

$$\ddot{\mathbf{x}} = \frac{d}{dt}\left[\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot f(\mathbf{x})\right] = \frac{\partial^2 f}{\partial \mathbf{x}^2}(\mathbf{x})\cdot f(\mathbf{x})\cdot f(\mathbf{x}) + \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot f(\mathbf{x})$$

$$\begin{aligned}\mathbf{x}^{(4)} = \quad & \frac{\partial^3 f}{\partial \mathbf{x}^3}(\mathbf{x})\cdot f(\mathbf{x})\cdot f(\mathbf{x})\cdot f(\mathbf{x}) + \frac{\partial^2 f}{\partial \mathbf{x}^2}(\mathbf{x})\cdot\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot f(\mathbf{x})\cdot f(\mathbf{x}) + \\ & \frac{\partial^2 f}{\partial \mathbf{x}^2}(\mathbf{x})\cdot f(\mathbf{x})\cdot\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot f(\mathbf{x}) + \frac{\partial^2 f}{\partial \mathbf{x}^2}(\mathbf{x})\cdot f(\mathbf{x})\cdot\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot f(\mathbf{x}) + \\ & \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot\frac{\partial^2 f}{\partial \mathbf{x}^2}(\mathbf{x})\cdot f(\mathbf{x})\cdot f(\mathbf{x}) + \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\cdot f(\mathbf{x})\end{aligned}$$

$\mathbf{x}^{(5)}$ enthält schon 24 Terme!

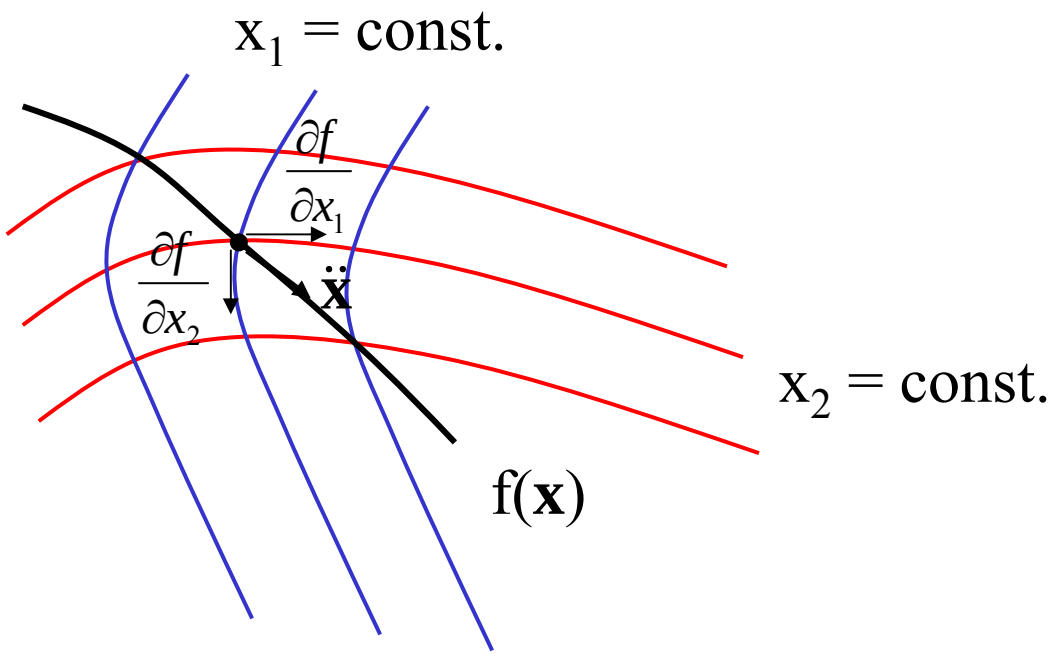
Probleme:

- Notwendige Anzahl von Werten steigt schnell an
- Höhere Ableitungen sind nicht bekannt

$\Rightarrow$ Höhere Ableitungen numerisch berechnen

$\Rightarrow$ Verwendung der Richtungsableitung statt der partiellen Ableitung

Richtungsableitung statt partiellen Ableitungen am Bsp. $\ddot{\mathbf{x}}$



Allgemeine Darstellung:

$$\ddot{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) \cdot f(\mathbf{x}) = \frac{\partial}{\partial \tau} f(\mathbf{x} + \tau \cdot f(\mathbf{x}))\Big|_{\tau=0}$$

Annäherung durch Differenzenquotienten

$$\frac{\partial}{\partial \tau} f(\mathbf{x} + \tau \cdot f(\mathbf{x}))\Big|_{\tau=0} \approx \frac{f(\mathbf{x} + \tau \cdot f(\mathbf{x})) - f(\mathbf{x})}{\tau} \left(+ O(\tau)\right)$$

Numerische Approximation der Ableitung

Partielle Ableitungen:
Auswertung von $f(\mathbf{x}+\Delta t \cdot \mathbf{e}_i)$ für $i = 1, ..., n$ wobei $\mathbf{e}_i$ der i-te Einheitsvektor ist
(n Funktionsauswertungen!)

Richtungsableitung:
Auswertung von $f(\mathbf{x}+\Delta t \cdot f(\mathbf{x}))$ als Richtungsableitung in Richtung $f(\mathbf{x})$
(1 Funktionsauswertung)

Auf Basis der vorherigen Beobachtungen können Verfahren höherer Ordnung konstruiert werden ⇒ Runge-Kutta-Verfahren

Runge-Kutta-Verfahren 1. Ordnung (entspricht dem Euler-Verfahren):

$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t)) + O(\Delta t^2)$ (keine numerische Differentation notwendig)

Runge-Kutta-Verfahren 2. Ordnung (Euler-Cauchy-Verfahren):

Es muss auch $\ddot{\mathbf{x}}(t)$ numerisch approximiert werden, dazu sind mindestens zwei Funktionsauswertungen notwendig!

$$\begin{aligned}
\mathbf{x}(t+\Delta t) &= \mathbf{x}(t) + \Delta t \cdot \dot{\mathbf{x}}(t) + \frac{\Delta t^2}{2} \cdot \ddot{\mathbf{x}}(t) + O(\Delta t^3) \\
&= \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t)) + \frac{\Delta t^2}{2} \cdot \left(\frac{f(\mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t))) - f(\mathbf{x}(t))}{\Delta t} + O(\Delta t) \right) + O(\Delta t^3) \\
&= \mathbf{x}(t) + \frac{\Delta t}{2} \cdot \left(f(\mathbf{x}(t)) + f(\mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t))) \right) + O(\Delta t^3)
\end{aligned}$$

Anders aufgeschrieben:

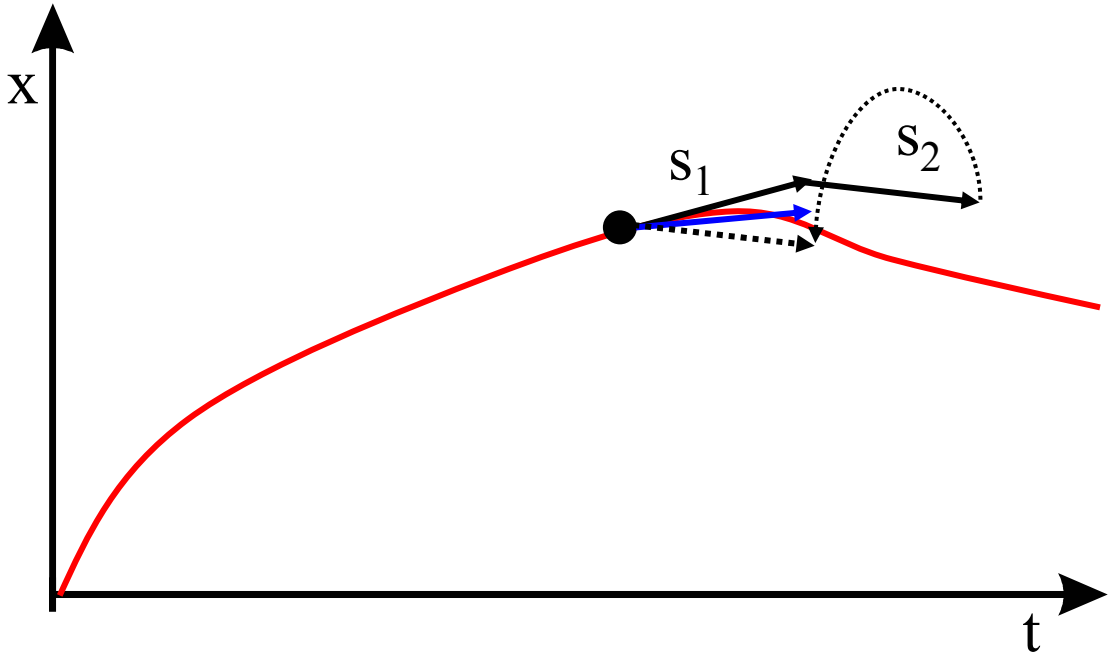
$$\mathbf{x}(t+\Delta t) \approx \mathbf{x}(t) + 0.5 \cdot \mathbf{s}_1 + 0.5 \cdot \mathbf{s}_2$$

mit

$\mathbf{s}_1 = \Delta t \cdot f(\mathbf{x}(t))$ ← Euler-Schritt

$\mathbf{s}_2 = \Delta t \cdot f(\mathbf{x}(t) + s_1)$ ← Richtungsvektor am Ende des Euler-Schritts

Graphische Darstellung (im eindimensionalen Fall):



Es gibt weitere Möglichkeiten Verfahren der Ordnung 2 zu definieren, z.B.

$$\mathbf{x}(t+\Delta t) \approx \mathbf{x}(t) + \frac{1}{4}\cdot\mathbf{s}_1 + \frac{3}{4}\cdot\mathbf{s}_2$$

mit

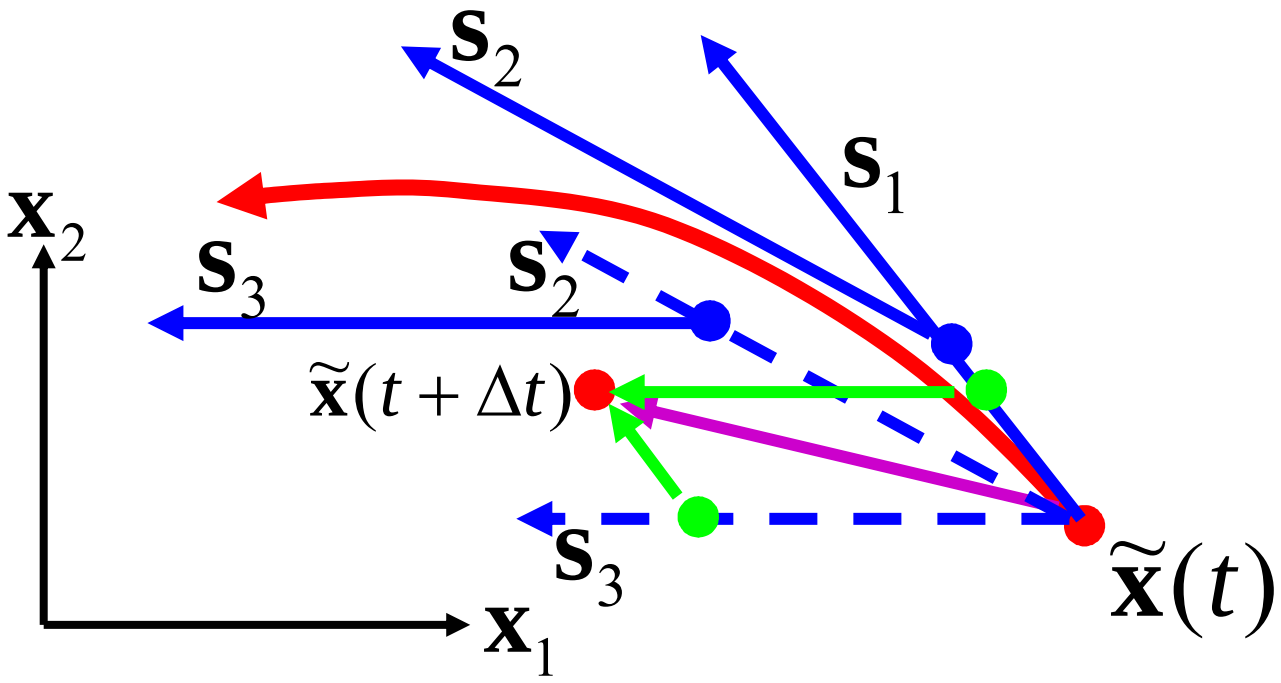
$$\mathbf{s}_1 = \Delta t \cdot f(\mathbf{x}(t))$$

$$\mathbf{s}_2 = \Delta t \cdot f\left(\mathbf{x}(t) + \frac{2}{3}\cdot\mathbf{s}_1\right)$$

Definition von Verfahren höherer Ordnung:
Finden einer Darstellung der obigen Form, so dass

- die Terme der Taylorreihe bis zu einer vorgegebenen Ordnung (Ableitung) berechnet werden und
- möglichst wenige Funktionsauswertungen benötigt werden

Graphische Darstellung der Runge-Kutta-Verfahren der Ordnung 3 (Heun)
(nach dem Skript von W. Wiechert U. Siegen)



$$\mathbf{x}(t+\Delta t) \approx \mathbf{x}(t) + \frac{1}{4}\cdot\mathbf{s}_1 + \frac{3}{4}\cdot\mathbf{s}_3$$

mit

$$\mathbf{s}_1 = \Delta t \cdot f(\mathbf{x}(t))$$

$$\mathbf{s}_2 = \Delta t \cdot f\left(\mathbf{x}(t) + \frac{1}{3}\cdot\mathbf{s}_1\right)$$

$$\mathbf{s}_3 = \Delta t \cdot f\left(\mathbf{x}(t) + \frac{2}{3}\cdot\mathbf{s}_2\right)$$

3 Funktionsauswertungen und Berücksichtigung der ersten 3 Ableitungen
Verfahrensfehler $O(\Delta t^3)$

Allgemeines Schema für k-stufige Runge-Kutta-Verfahren

$$
\begin{aligned}
\mathbf{s}_1 &= \Delta t \cdot f(\mathbf{x}(t)) \\
\mathbf{s}_2 &= \Delta t \cdot f(\mathbf{x}(t) + \alpha_{21} \cdot \mathbf{s}_1) \\
\mathbf{s}_3 &= \Delta t \cdot f(\mathbf{x}(t) + \alpha_{31} \cdot \mathbf{s}_1 + \alpha_{32} \cdot \mathbf{s}_2) \\
&\vdots \quad \vdots \\
\mathbf{s}_k &= \Delta t \cdot f(\mathbf{x}(t) + \alpha_{k1} \cdot \mathbf{s}_1 + \alpha_{k2} \cdot \mathbf{s}_2 + \ldots + \alpha_{kk-1} \cdot \mathbf{s}_{k-1})
\end{aligned}
$$

Vektorberechnung

$$
\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \beta_1 \cdot \mathbf{s}_1 + \beta_2 \cdot \mathbf{s}_2 + \ldots + \beta_k \cdot \mathbf{s}_k
$$

Berechnungsschritt

Allgemeine Darstellung des k-stufigen Runge-Kutta-Verfahrens:

$$
\begin{array}{ccccc}
0 & & & & \\
\alpha_{21} & 0 & & & \\
\alpha_{31} & \alpha_{32} & 0 & & \\
\vdots & \vdots & \ddots & \ddots & \\
\alpha_{k1} & \alpha_{k2} & \cdots & \alpha_{kk-1} & 0 \\
\hline
\beta_1 & \beta_2 & \cdots & \beta_{k-1} & \beta_k
\end{array}
$$

Kompakte Darstellung der Berechnungsvorschrift (untere Dreiecksmatrix + Vektor)

Zur Bestimmung der Koeffizienten

Zur Erinnerung: Lösung durch Approximation der Taylor-Reihe am Punkt t

⇒ Koeffizienten der Taylorreihe bis zu einer vorgegebenen Potenz müssen durch entsprechende Wahl der α_{ij} und β_i wiedergegeben werden

Beispiele:

n=1

Taylorreihe:

$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t)) + O(\Delta t^2)$

RK-Verfahren:

$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \beta_1 \cdot \Delta t \cdot f(\mathbf{x}(t))$

$\Rightarrow \beta_1 = 1$ muss gelten

(eindeutiges Verfahren)

n=2

Taylorreihe:

$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t)) + \Delta t^2 / 2 \cdot \dot{f}(\mathbf{x}(t)) + O(\Delta t^3)$

RK-Verfahren:

$\mathbf{x}(t+\Delta t) =$

$\mathbf{x}(t) + \beta_1 \cdot \Delta t \cdot f(\mathbf{x}(t)) + \beta_2 \cdot \Delta t \cdot f(\mathbf{x}(t) + \alpha_{21} \cdot \Delta t \cdot f(\mathbf{x}(t))$

$\Rightarrow \beta_1 + \beta_2 = 1$ und $\beta_2 \cdot \alpha_{21} = 0.5$ muss gelten

(kein eindeutiges Verfahren)

Mögliche Realisierungen

0	
1	0
1/2	1/2

0	
2/3	0
1/4	3/4

n=3

Taylorreihe:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t)) + \Delta t^2/2 \cdot \dot{f}(\mathbf{x}(t)) + \Delta t^3/6 \cdot \ddot{f}(\mathbf{x}(t)) + O(\Delta t^4)$$

RK-Verfahren:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \beta_1 \cdot \Delta t \cdot f(\mathbf{x}(t) + \beta_2 \cdot \Delta t \cdot f(\mathbf{x}(t) + \alpha_{21} \cdot \Delta t \cdot f(\mathbf{x}(t))) + \beta_3 \cdot \Delta t \cdot f(\mathbf{x}(t) + \alpha_{31} \cdot \Delta t \cdot f(\mathbf{x}(t)) + \alpha_{32} \cdot \Delta t \cdot f(\mathbf{x}(t) + \alpha_{21} \cdot \Delta t \cdot f(\mathbf{x}(t))))$$

$\beta_1 + \beta_2 + \beta_3 = 1$ und $\beta_2 \cdot \alpha_{21} + \beta_3 \cdot \alpha_{31} + \beta_3 \cdot \alpha_{32} = 1/2$ und $\beta_3 \cdot \alpha_{21} \cdot \alpha_{32} = 1/6$ muss gelten (Lösung eines nichtlinearen Gleichungssystems)

Mögliche Realisierungen

Heun

0			
1/3	0		
0	2/3	0	
1/4	0	3/4	

Kutta

0			
1/2	0		
−1	2	0	
1/6	4/6	1/6	

Einige Bemerkungen

- für Verfahrensordnung > 1 gibt es mehrere Alternativen
  es ist i.d.R. nicht klar, welche Alternative die besten Resultate liefert
- es gilt Verfahrensordnung ≤ Stufenzahl
  falls Verfahrensordnung > 4 gilt sogar Verfahrensordnung < Stufenzahl
- überlicherweise verwendete Verfahrensordnung 4 (Runge-Kutta Verfahren)
- bei höherer Ordnung (≥ 4) nur noch geringe Effekte durch Erhöhung der Ordnung
- Berechnung der Koeffizienten höherer Ordnung ist komplex
  (Lösung eines nichtlinearen Gleichungssystems)
- bisher höchste erreichte Ordnung 10 mit 13 Stufen (praktisch nicht relevant)
- Lösungsverfahren vom Runge-Kutta-Typ sind weit verbreitet, es gibt aber noch viele andere Ansätze (die an dieser Stelle nicht im Detail vorgestellt werden)

Vorgehen bei der Analyse kontinuierlicher Systeme
(wie bisher beschrieben):

(Approximative) Berechnung der Funktionswerte zu äquidistanten Zeitpunkten mit Abstand Δt

- kleinere Werte von Δt führen zu genaueren Resultaten
  (zumindest so lange bis Rundungsfehler auftreten)
- kleinere Werte von Δt erhöhen den Aufwand
- tatsächlicher Fehler der Berechnung hängt von der Struktur des zu analysierenden Systems ab
  (Δt muss entsprechend der Zeitkonstanten des Systems gewählt werden)

Auftretende Probleme:

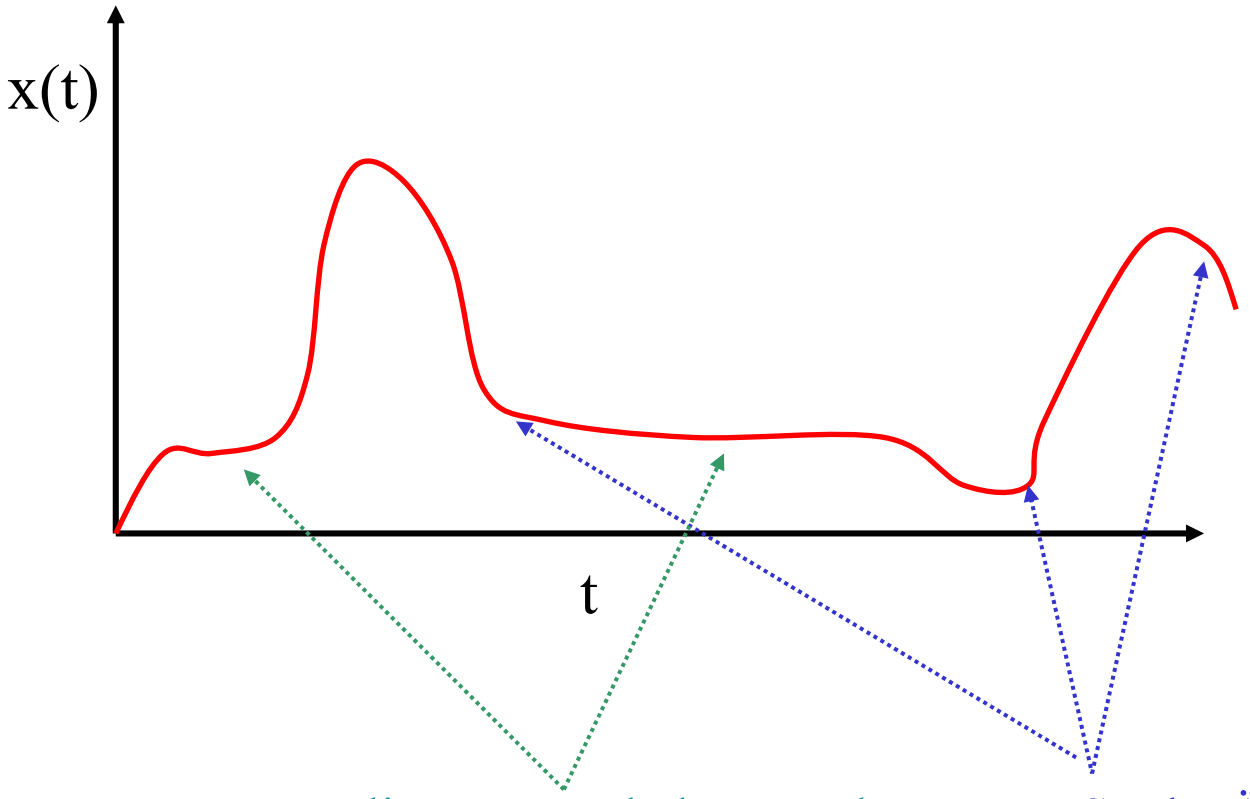
- Zeitkonstanten eines Systems sind nur schwer ablesbar und variieren in Abhängigkeit vom Zustand

$\Rightarrow$ die Festlegung einer adäquaten Schrittweite ist schwieirg

$\Rightarrow$ die adäquate Schrittweite kann sich während der Simulation ändern

$\Rightarrow$ naheliegender Ansatz:
Adaptive Schrittweitensteuerung während der Analyse
(heute in fast allen Werkzeugen zur kontinuierlichen Simulation verwendet)

Darstellung des Problems am einfachen Beispiel



Falls Δt zu

- groß gewählt wird, wird die Berechnung zu ungenau
- klein gewählt wird, wird der Berechnungs-aufwand zu hoch

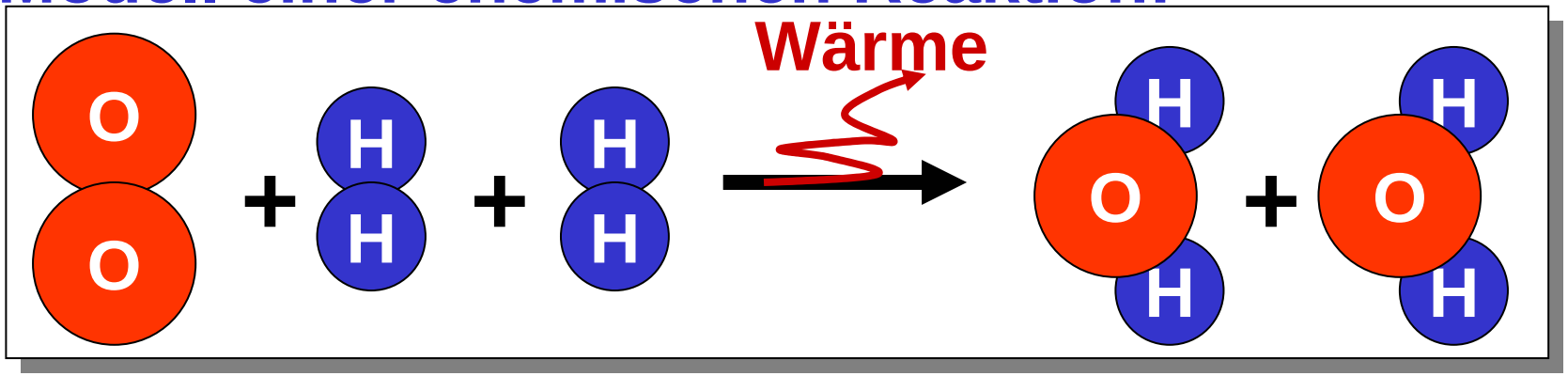
➢ es existiert kein festes Δt, welches genügend genaue Resultate mit vertretbarem Aufwand liefert

Fast lineares Verhalten, auch bei relativ großem Δt genau genug approximierbar

Starke Änderung der Ableitung, nur bei kleinem Δt genau genug approximierbar

Ein konkretes Beispiel (nach dem Skript von Prof. Wiechert)

Modell einer chemischen Reaktion:



Zustandsvariablen:

- H(t) Wasserstoffmenge zum Zeitpunkt t in Molar ($H(0)=H_0$)
- O(t) Sauerstoffmenge zum Zeitpunkt t in Molar ($O(0)=O_0=0.5H_0$)
- T(t) Temperatur zum Zeitpunkt t in Kelvin ($H(0)=H_0$)

Ziel: Simulation der Reaktion bis zum Zeitpunkt τ

τ ist so groß zu wählen, dass (fast) nur noch Wasser existiert

Wie ist Δt zu wählen?

Modellgleichungen zur Beschreibung der Reaktion

$$\dot{H} = -A \cdot e^{-\frac{E}{R \cdot T}} \cdot H \cdot O$$
$$\dot{O} = \frac{1}{2} \cdot \dot{H}$$
$$\dot{T} = -C \cdot \dot{H}$$

(T Temperatur)

Mit den Konstanten

- E Aktivierungsenergie
- R Gaskonstante
- C Wärmekapazität multipliziert mit der freiwerdenden Energie
- A Arrhenius Konstante

Umstellen der zweiten und dritten Gleichung liefert

$$O - O_0 = \frac{1}{2} \cdot (H - H_0) \Rightarrow O = \frac{1}{2} \cdot H \quad (\text{wg. } O_0 = \frac{1}{2} \cdot H_0)$$
$$T - T_0 = -C \cdot (H - H_0) \Rightarrow T = T_0 - C \cdot (H - H_0)$$

Einsetzen in die erste Gleichung führt zu einer Reaktionsgleichung:

$$\dot{H} = \frac{1}{2} \cdot A \cdot e^{-\frac{E}{R \cdot (T_0 + C \cdot (H - H_0))}} \cdot H^2$$

Für kurze Zeiträume verhält sich das System linear:

$$\dot{H} = \alpha \cdot H$$

mit $\alpha(H) = -\frac{1}{2} \cdot A \cdot e^{-\frac{E}{R \cdot (T_0 + C \cdot (H - H_0))}} \cdot H$

Damit gilt $H(t + \Delta t) \approx H(t) \cdot e^{-\alpha(H(t)) \cdot t}$

(Exponentialfunktion mit zeitanhängiger Rate!)

Verhalten von α(H(t)):

$$\alpha(H_0) = -\frac{1}{2} \cdot A \cdot e^{-\frac{E}{R \cdot T_0}} \cdot H_0 \qquad \alpha_{\max} = -\frac{1}{2} \cdot A \cdot H_0 \qquad \lim_{t \to \infty} \alpha(H) = 0$$

Oft gilt $\alpha_{max}/\alpha(H_0) \approx 10^6$ ⇒ Integrator mit fester Schrittweite müsste $\Delta t \approx 10^{-6}$ wählen

⇒ sehr ineffiziente Berechnung

Zur sinnvollen Berechnung sollte Δt so klein sein, dass

$|\varepsilon_{ED}(\Delta t)| \approx C \cdot \Delta t^{n+1}$ für ein Verfahren der Ordnung n

Bei bekanntem C und vorgegebener Fehlertoleranz ε_{Tol} gilt:

$$
\begin{aligned}
& |\varepsilon_{ED}(\Delta t)| && \leq && \varepsilon_{Tol} \\
\Leftrightarrow \quad & C \cdot \Delta t^{n+1} && \leq && \varepsilon_{Tol} \\
\Leftrightarrow \quad & \Delta t && \leq && \left(\frac{\varepsilon_{Tol}}{C} \right)^{1/(n+1)}
\end{aligned}
$$

$\varepsilon_{ED}(Dt)$ und ε_{Tol} können als absolute oder relative Fehler berechnet werden:
Praktisch verwendete Definition des relativen Fehlers
($\mathbf{x}_e$ exakt, $\mathbf{x}_a$ approximativ)

$$\varepsilon_{rel} = \frac{\|\mathbf{x}_e - \mathbf{x}_a\|}{\max(\|\mathbf{x}_e\|, \|\mathbf{x}_a\|, \delta)}$$

Da C nicht bekannt ist, muss der Wert geschätzt werden
Eine Schätzung kann nur auf Basis numerisch berechneter Resultate erfolgen $\Rightarrow$

- Berechne Resultate auf unterschiedliche Weise
- Benutze Differenz der Ergebnisse zur Schätzung von C

Methode der Schrittweitenhalbierung

In jedem Schritt werden 3 Werte berechnet:

- Wert zum Zeitpunkt t + Δt: $\mathbf{x}_{\Delta t}(t+\Delta t)$
- Wert zum Zeitpunkt t + 0.5·Δt: $\mathbf{x}_{0.5,\Delta t}(t+0.5\cdot\Delta t)$ und davon ausgehend Wert zum Zeitpunkt t + Δt: $\mathbf{x}_{0.5,\Delta t}(t+\Delta t)$

Approximation des auftretenden Fehlers:

$$\begin{aligned}\left|\varepsilon_{ED}(\Delta t)\right| &\approx \left\|\mathbf{x}_{\Delta t}(t+\Delta t)-\mathbf{x}_{0.5,\Delta t}(t+\Delta t)\right\| \\ &\approx C\cdot\Delta t^{n+1}\end{aligned}$$

Womit folgende Abschätzung gilt: $C \approx \dfrac{\left\|\mathbf{x}_{\Delta t}(t+\Delta t)-\mathbf{x}_{0.5,\Delta t}(t+\Delta t)\right\|}{\Delta t^{n+1}}$

Anschließend wird Δt verändert, um die vorgegeben Fehlerschranke zu erreichen: $\Delta t_{opt} = \alpha \cdot \sqrt[n+1]{\varepsilon_{Tol}/C}$ mit $0<\alpha<1$

Vierfacher Rechenaufwand pro Schritt !!
(3 Schritte zur Bestimmung von Δt_{opt} + 1 Verfahrensschritt)

Methode der eingebetteten Verfahren

Die Lösung $\mathbf{x}(t+\Delta t)$ wird, ausgehend von $\mathbf{x}(t)$, mit zwei unterschiedlichen Verfahren berechnet, die Lösungsvektoren $\mathbf{x}_1(t+\Delta t)$ und $\mathbf{x}_2(t+\Delta t)$ liefern (z.B. RK 2/3 oder RK 4/5)

Anschließend schätzt man $C \approx \dfrac{\left\|\mathbf{x}_1(t+\Delta t) - \mathbf{x}_2(t+\Delta t)\right\|}{\Delta t^{n+1}}$

(wobei n die kleinere der beiden Fehlerordnungen ist)

Δt_{opt} wird wie auf der vorherigen Folie beschrieben berechnet

Aufwand pro Schritt: 3 Berechnungsschritte
(2 zur Bestimmung von Δt_{opt} und 1 Verfahrensschritt)

Aufwand lässt sich auf 2 Berechnungsschritte reduzieren, falls Runge-Kutta Verfahren so konstruiert werden, dass Zwischenergebnisse wiederverwendbar sind (eingebettete Verfahren).

Weitere Bemerkungen:

- C wird nur geschätzt und die Schätzung kann falsch sein (deshalb eher etwas kleinere Werte für α verwenden)
- Für die meisten Modelle wird der Aufwand zusätzlicher Berechnungen durch die dynamische Anpassung der Schrittweite mehr als ausgeglichen
- Eine a priori Schätzung von Δt ist nicht notwendig
- Falls das aktuell verwendet Δt kleiner als das ermittelte Δt_{opt} ist, so kann auf den abschließenden Verfahrensschritt verzichtet werden und mit dem „genaueren“ der ermittelten Vektoren $x(t+\Delta t)$ fortgefahren werden
- Da durch eine Verkleinerung der Schrittweite die Zahl der Schritte steigt, steigt auch der Gesamtfehler bei konstantem Einzelschrittfehler
  Berücksichtigung des Gesamtfehlers liefert:
  Schrittzahl $n = \tau/\Delta t$ und $\Delta t_{opt} = \sqrt[n]{\varepsilon_{DTol} \cdot \tau / C}$
  (ε_{DTol} Gesamtfehlerschranke)

Differential-algebraische Gleichungssysteme (DAEs)

Allgemeine Form: $\dot{x} = f(x, y)$ *und* $0 = g(x, y)$

für den Anfangszustand gilt $0 = g(x_0, y_0)$

x differential Variablen
y algebraische Var.

Simultane Lösung eines (nicht notwendigerweise linearen) Gleichungssystems und eines Differentialgleichungssystems

Lineare DAEs: $\dot{x} = A \cdot x + B \cdot y + v$ *und* $0 = D \cdot x + E \cdot y + w$

Annahme: E sei quadratische reguläre Matrix
(kann oft durch Streichung von redundanten Gleichungen erreicht werden)

Damit gilt $y = -E^{-1}(Dx + w)$ und

$$\begin{aligned}\dot{x} &= Ax - E^{-1}(Dx + w) + v \\ &= (A - E^{-1}D)x + (v - E^{-1}w)\end{aligned}$$

Kann mit einem „normalen“ Verfahren für DGLs analysiert werden!

Allgemeine DAEs:

Simultane Lösung:

Verwendung eines impliziten Verfahrens zur DGL-Lösung

z.B. implizites Euler Verfahren: $x(t+\Delta t) = x(t) + \Delta t \cdot f(x(t+\Delta t))$

$$\begin{aligned} \frac{1}{\Delta t}\big(x(t+\Delta t) - x(t)\big) &= f(x(t+\Delta t), y(t+\Delta t)) \\ 0 &= g(x(t+\Delta t), y(t+\Delta t)) \end{aligned}$$

Umwandlung in ein Fixpunktproblem und Lösung mittels Newton-Verf.

$$\begin{aligned} x(t+\Delta t) &= \Delta t \cdot f(x(t+\Delta t), y(t+\Delta t)) + x(t) \\ y(t+\Delta t) &= y(t+\Delta t) - g(x(t+\Delta t), y(t+\Delta t)) \end{aligned}$$

Newton-Verfahren:

$$x^{(k+1)} = x^{(k)} - \Delta x^{(k)}$$
$$\Delta x^{(k)} = \left(H\left(x^{(k)}\right)\right)^{-1} \cdot f\left(x^{(k)}\right)$$
$$H(x) = \frac{df}{dx}$$

Falls Newton nicht konvergiert $\Rightarrow \Delta t$ verkleinern

<u>Kombination ODE-Lösung und Gleichungslöser:</u>

1. Starte mit $x^{(0)}(t+\Delta t)$ und $y^{(0)}(t+\Delta t)$, setze k=1
2. Berechne $x^{(k)}(t+\Delta t)$ mit ODE Löser
3. Berechne $y^{(k)}(t+\Delta t)$, so dass $g(x^{(k)}(t+\Delta t), x^{(k)}(t+\Delta t))=0$ mittels Newton-Verfahren
   - falls keine Konvergenz verkleinere Δt und fahre bei 2 fort
4. Falls $x^{(k)}(t+\Delta t) \approx x^{(k-1)}(t+\Delta t)$ und $y^{(k)}(t+\Delta t) \approx y^{(k-1)}(t+\Delta t)$ stop, sonst k=k+1 und fahre bei 2. fort

Stationäre Analyse

Ziel: Lösung $f(\mathbf{x}^*) = 0$ für gegebenes $\mathbf{x}(0)$

Häufige Annahme: System schwingt schnell den stationären Zustand ein

Lösungsansätze (nicht jeder Ansatz funktioniert für jede Funktion!):

1. Stationärer Punkt im DGL: $\lim_{t\to\infty}\mathbf{x}(t) = \mathbf{x}^*$
   Berechne $\mathbf{x}(t)$ für großes t mit DGL-Löser
2. Umwandlung in eine Fixpunktgleichung
   $0 = f(\mathbf{x}) \Leftrightarrow \mathbf{x} = f(\mathbf{x}) + \mathbf{x}$ $(:= g(\mathbf{x}))$
   Lösung durch Fixpunktiteration $\mathbf{x} = g(\mathbf{x})$
   (Konvergenz falls $dg/d\mathbf{x}$ $(\mathbf{x}^*)$ im Einheitskreis)
3. Taylorreihenentwicklung
   $f(\mathbf{x}+\Delta\mathbf{x}) \approx f(\mathbf{x}) + df/d\mathbf{x}(\mathbf{x}) \cdot \Delta\mathbf{x}$ (Approximation 1. Ordnung)
   Näherung $0 = f(\mathbf{x}) + df/d\mathbf{x}(\mathbf{x}) \cdot \Delta\mathbf{x} \Rightarrow -\Delta\mathbf{x} = \left(df/d\mathbf{x}(\mathbf{x})\right)^{-1} \cdot f(\mathbf{x})$
   $\Rightarrow$ neue Näherung $\mathbf{x} := \mathbf{x} + \Delta\mathbf{x}$ (Newton-Verfahren)
4. Mehrdimensionale Optimierung
   $f(\mathbf{x}) = 0 \Leftrightarrow \| f(\mathbf{x}) \| = 0$
   Minimierungsproblem: $f(\mathbf{x}^*) = 0 \Leftrightarrow \mathbf{x}^* = \min_{\mathbf{x}} \| f(\mathbf{x}) \|^2$

4.3 Beispiele für hybride Systeme

Bisher strenge Trennung in

- kontinuierliche Systeme
  Darstellung kontinuierlicher Zustandsänderungen in
  - vielen naturwissenschaftlichen Anwendungen (Physik, Chemie, ...)
  - populationsbasierten Systemen mit sehr großer Population (Straßenverkehr, Bevölkerungsentwicklung)
- diskrete Systeme
  Darstellung im wesentlichen ereignisorientierter Zustandsänderungen in
  - vielen technischen Systemen (Computer, Kommunikation, Fertigung, ...)
  - diskretisierten kontinuierlichen Systemen

Einzelne Simulationsansätze

- etabliert und in vielen Systemen realisiert
- aber grundsätzlich verschieden
    - numerische Analyse von Differentialgleichungen im kontinuierlichen Fall
    - stochastische ereignisdiskrete Simulation im diskreten Fall

Kombination von kontinuierlicher und diskreter Simulation:

Wird diese benötigt?

Wie simuliert man?

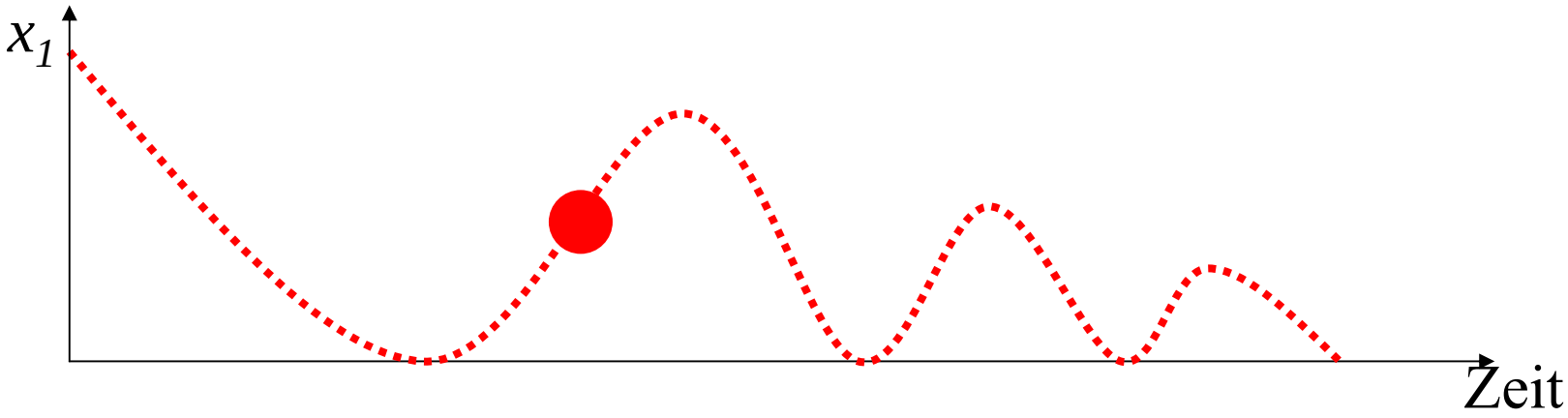
Wie modelliert und spezifiziert man?

$\Rightarrow$ Hybride Simulation: Inhalt des letzten Teils dieses Kapitels

Im Wesentlichen zwei Klassen von Beispielen:

- Kontinuierliche Modelle, bei denen Diskontinuitäten auftreten
  - Überlast in elektronischen Schaltungen
  - Kollision von Körpern
  - sehr steife Differentialgleichungssysteme
- Diskrete Steuerung kontinuierlicher Prozesse
  - digitale Steuerung
    - generiert zeitgesteuerte Ereignisse
    - reagiert auf Zustandsänderungen im kontinuierlichen Teil
  - kontinuierlicher Prozesse
    - schreitet fort
    - wird durch Steuerungsimpulse beeinflusst u.U. auch durch abrupte Zustandsänderungen

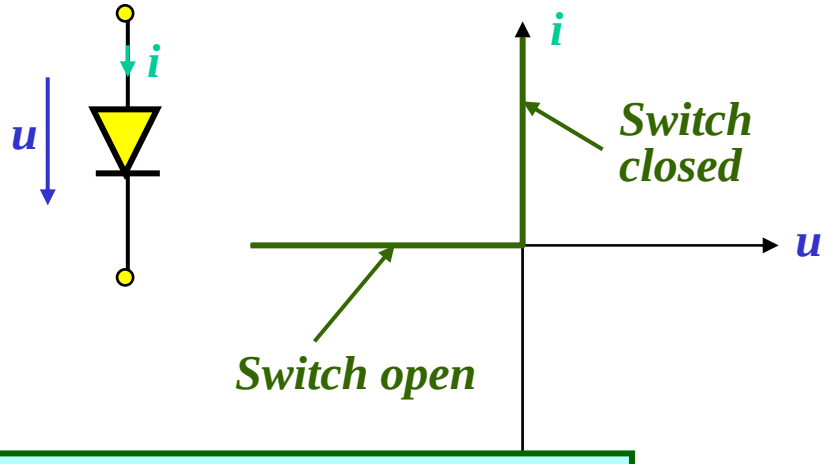
Aufspringender Ball:



Beschreibung:

- x_1 sei die vertikale Position des Balls ($x_1 \geq 0$)
  uns interessiert nur die Höhe, nicht die horizontale Position
- x_2 Geschwindigkeit des Balls, also $\dot{x}_1 = x_2$
- Geschwindigkeitsänderung durch Erdbeschleunigung vorgegeben
  also $\dot{x}_2 = -g$
- Richtungsänderung beim Aufprall des Balls
  falls $x_1 = 0$ dann $x_2 = -c \cdot x_2$ (wobei $0 \leq c \leq 1$)

Verhalten einer idealen Diode:

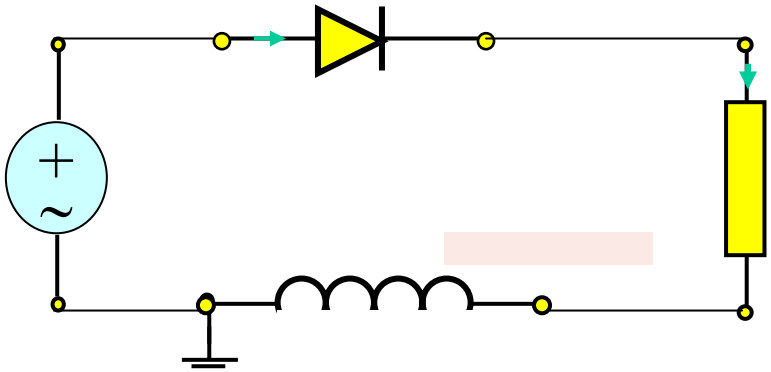


When $u < 0$, the switch is open. No current flows through.

When $u > 0$, the switch is closed. Current may flow. The ideal diode behaves like a short circuit.

open = u < 0 ;
0 = if open then i else u ;

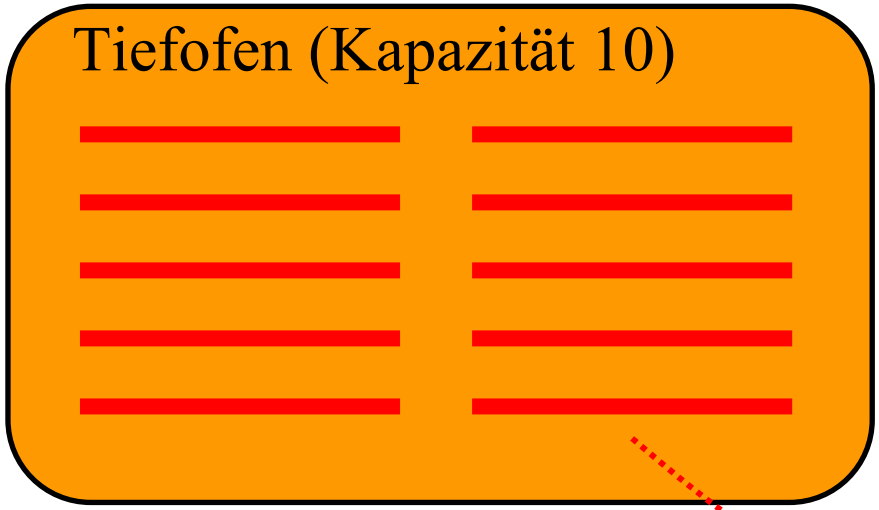
Beispiel:



Tiefofen in einem Walzwerk

Entfernen eines Barrens aus dem Ofen, falls seine Temperatur ≥ 380 ^{0}C

Stahlbarren im Warteraum



Ankunftsstrom
Poisson mit Rate λ
Ankunftstemperatur:
200 ^{0}C

Temperatur im Warteraum
25 ^{0}C

Temperatur im Ofen y(t) ≤ 500 ^{0}C

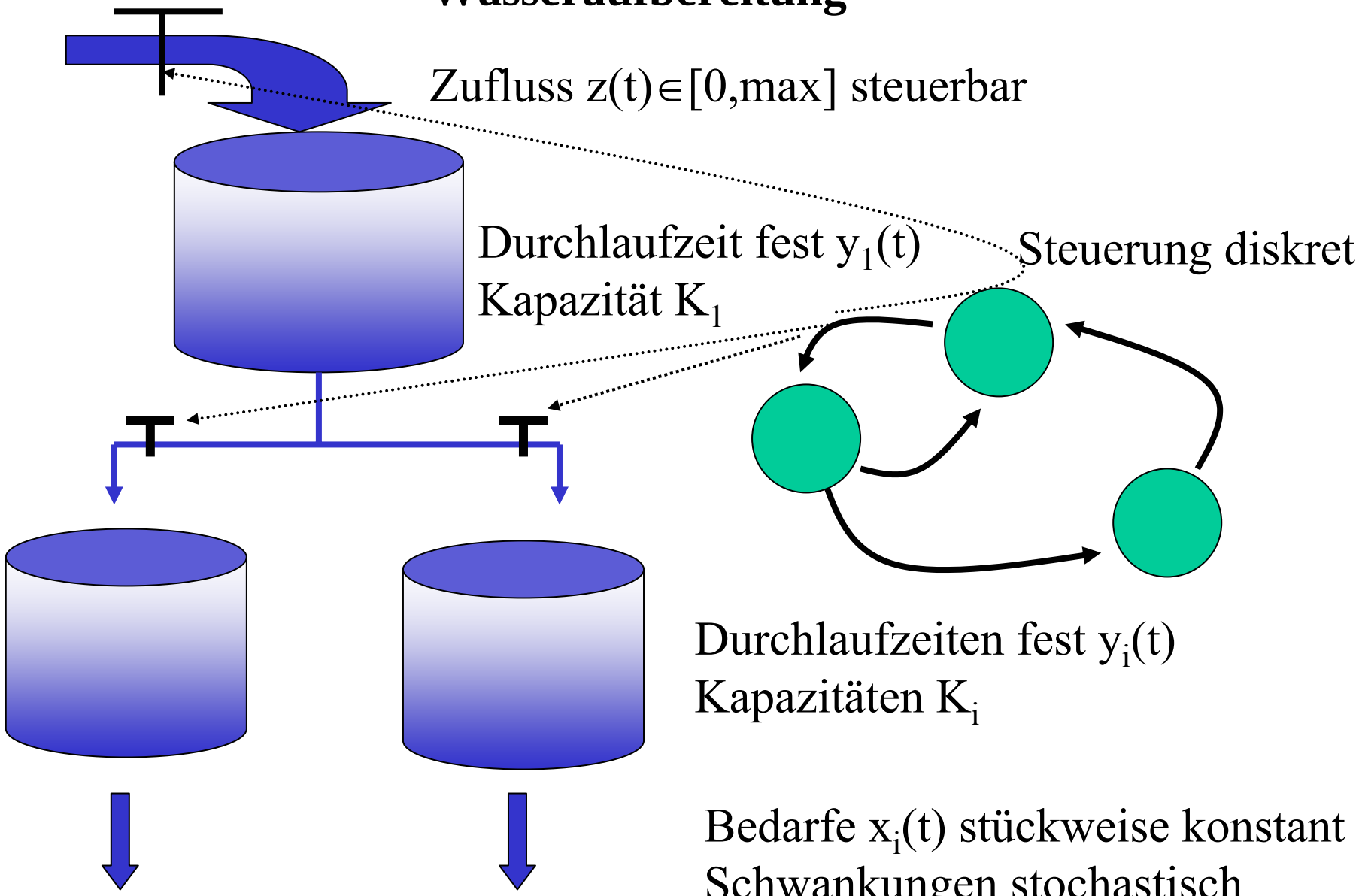
Temperaturänderungen:

$$\dot{x}_i(t) = (u(t) - x_i(t))/7$$

$$\dot{y}(t) = 500 - \sum_{i \text{ im Ofen}} (y(t) - x_i(t))/3$$

$x_i(t)$ Temperatur Barren i
$u_i(t)$ Umgebungstemperatur Barren i (25 oder y(t))

Wasseraufbereitung



4.5 Hybride Simulationsansätze

- Hybride Simulation als Erweiterung diskreter Simulation, falls kontinuierlicher Teil einfach diskretisierbar
- Hybride Simulation als Erweiterung kontinuierlicher Simulation, falls diskontinuierliche Ereignisse spezifizierbar und ohne Ereignisliste (u.U. auch ohne Stochastik) realisierbar

Für allgemeine hybride Modelle sind spezielle Simulationsansätze notwendig (Kombination diskret & kontinuierlich)

Solange keine Interaktion zwischen diskretem und kontinuierlichem Teil auftritt, können die jeweiligen Simulationskonzepte Verwendung finden

$\Rightarrow$ zentraler Aspekt Interaktion zwischen den unterschiedlichen Teilen

Mögliche Interaktionsszenarien:

Zeitgesteuerte Ereignisse

- Ereignisse werden in der Ereignisliste gehalten und sind nach der Zeit ihres Eintretens geordnet
- Ereignisse können zu Zustandsänderungen im diskreten Teil führen
- Ereignisse können zu Diskontinuitäten im kontinuierlichen Teil führen
- Ereignisse werden zu Beginn der Simulation und während der Simulation durch andere Ereignisse generiert (Zeit eines neuen Ereignisses kann nicht in der Vergangenheit liegen!)

Zustandsgesteuerte Ereignisse

- Bei Über- oder Unterscheiten vorgegebener Schranken wird ein diskretes Ereignis ausgelöst
- Auswirkung des Ereignisses siehe oben
- Ereigniszeitpunkte ergeben sich implizit aus der Zustandstrajektorie (nicht vorhersehbar bzw. speicherbar)

Simulation zeitgesteuerter Ereignisse:

Sei t der aktuelle Zeitpunkt und t_e der Zeitpunkt des nächsten Ereignisses in der Ereignisliste

- Integrationsalgorithmus wird benutzt, um aus bekanntem x(t) den Wert $x(t_e)$ zu berechnen,
- falls im Intervall $[t, t_e)$ keine zustandsgesteuerten Ereignisse auftreten
  - normaler Ablauf des Algorithmus,
  - im letzten Schritt ist Δt so zu wählen, dass genau t_e erreicht wird
- diskretes Ereignis zum Zeitpunkt t_e wird ausgeführt und liefert neuen Zustand $x'(t_e)$
- $x'(t_e)$ wird als Startwert der Simulation bis zum nächsten Ereignis verwendet

Simulation eines Systems mit d Ereignissen entspricht d+1 separaten Simulationsläufen

Zustandsgesteuerte Ereignisse
Algorithmus für den eindimensionalen Fall, Ereignis falls x(t)=y

1. Ausführung eines Simulationsschrittes mit Schrittweite Δt
2. Falls $((x(t)<y \wedge x(t+\Delta t)>y) \vee (x(t)>y \wedge x(t+\Delta t)<y))$
   a. $\delta t=(|x(t)-y|)/(|x(t)-x(t+\Delta t)|) \cdot \Delta t$
   b. berechne $x(t+\delta t)$
   c. Falls $((x(t)<y \wedge x(t+\delta t)<y) \vee (x(t)>y \wedge x(t+\delta t)>y))$
      $t=t+\delta t$ und $\Delta t=\Delta t-\delta t$
   d. Falls $((x(t)<y \wedge x(t+\delta t)>y) \vee (x(t)>y \wedge x(t+\delta t)>y))$
      $\Delta t=\delta t$
   e. falls $|x(t+\Delta t)-y|<\varepsilon$
      $t = t + \Delta t$ und gehe zu 3.
   f. falls $|x(t)-y|<\varepsilon$
      gehe zu 3.
   g. gehe zu 2a
3. führe das diskrete Ereignis aus
4. Falls $t+\Delta t >$ Simzeit dann stop, sonst $t=t+\Delta t$ und gehe zu 1.

- Algorithmus ist für mehrere Dimensionen einfach erweiterbar
- x(t)=y kann nur bis auf eine Toleranz ε bestimmt werden
- es ist nicht feststellbar, ob x(t‘)=y für ein $t<t'<t+\Delta t$ erreicht wird, wenn sich das Vorzeichen der Differenz nicht ändert
- Automatische Schrittweitensteuerung sollte Abstand |x(t)-y| mit einbeziehen

4.6 Spezifikation hybrider Systeme

- Zahlreiche kontinuierliche Simulatoren erlauben Integration diskreter Ereignisse, oft aber keine vollständige ereignisdiskrete Modellierung
- Einzelne ereignisdiskrete Simulatoren erlauben Integration kontinuierlicher Anteile, oft muss das kontinuierliche Lösungsverfahren aber manuell integriert werden
- Einzelne Simulationsbibliotheken/-sprachen beinhalten Komponenten für diskrete und kontinuierliche Simulation, Kopplung ist aber in der Regel manuell zur realisieren

Wie betrachten hier

- Modelica als objektorientierte Sprache für kontinuierliche Systeme mit Erweiterungen zur hybriden Modellierung
- Hybride Petri Netze
- Hybride Automaten

Modelica:

Modelica ist eine objektorientierte Sprache primär zur Beschreibung kontinuierlicher Systeme.

Beispiel Schaltkreis:

```
connector Pin
  Voltage v ;
  flow Current i ;
end Pin
```

```
connect (Pin1, Pin2)
```

definiert implizit die Gleichungen

Pin1.v = Pin2.v und Pin1.i + Pin2.i = 0

```
partial class TwoPin
   Pin p, n ;
   Voltage v ;
   Current i ;
 equation
  v = p.v – n.v ;
  0 = p.i + n.i ;
  i  = p.i ;
end TwoPin ;
```

Definition eines Widerstandes:

```
class Resistor
   extends TwoPin ;
   parameter Real R(unit = „Ohm“) ;
 equation
   R · i = v ;
end Resistor ;
```

In ähnlicher Form werden weitere Elemente definiert

Definition von diskreten Ereignissen über if .. then .. else ...

Beispiel Diode:

```
class Diode
  extends TwoPin ;
equation
  open = u < 0 ;
  0 = if open then i else v ;
end Diode ;
```

Konstrukte sind ausreichend, um Diskontinuitäten in kontinuierlichen Modellen zu beschreiben, aber nicht, um komplexe diskrete Modelle zu beschreiben!

Bibliotheken existieren um z.B. einfache Petri-Netze oder State-Charts zu beschreiben

Nicht realisiert sind

- stochastische Funktionen
- Ereignisliste
- dynamische Objekterzeugung

Hybride Petri-Netze

Petri Netze sind ursprünglich ein Modell mit diskreter Zustandsbeschreibung und explizit ohne Zeitbegriff.
In der Folgezeit in unterschiedliche Richtungen weiterentwickelt, u.a. zu zeitbehafteten, stochastischen kontinuierlichen und hybriden Petri Netze

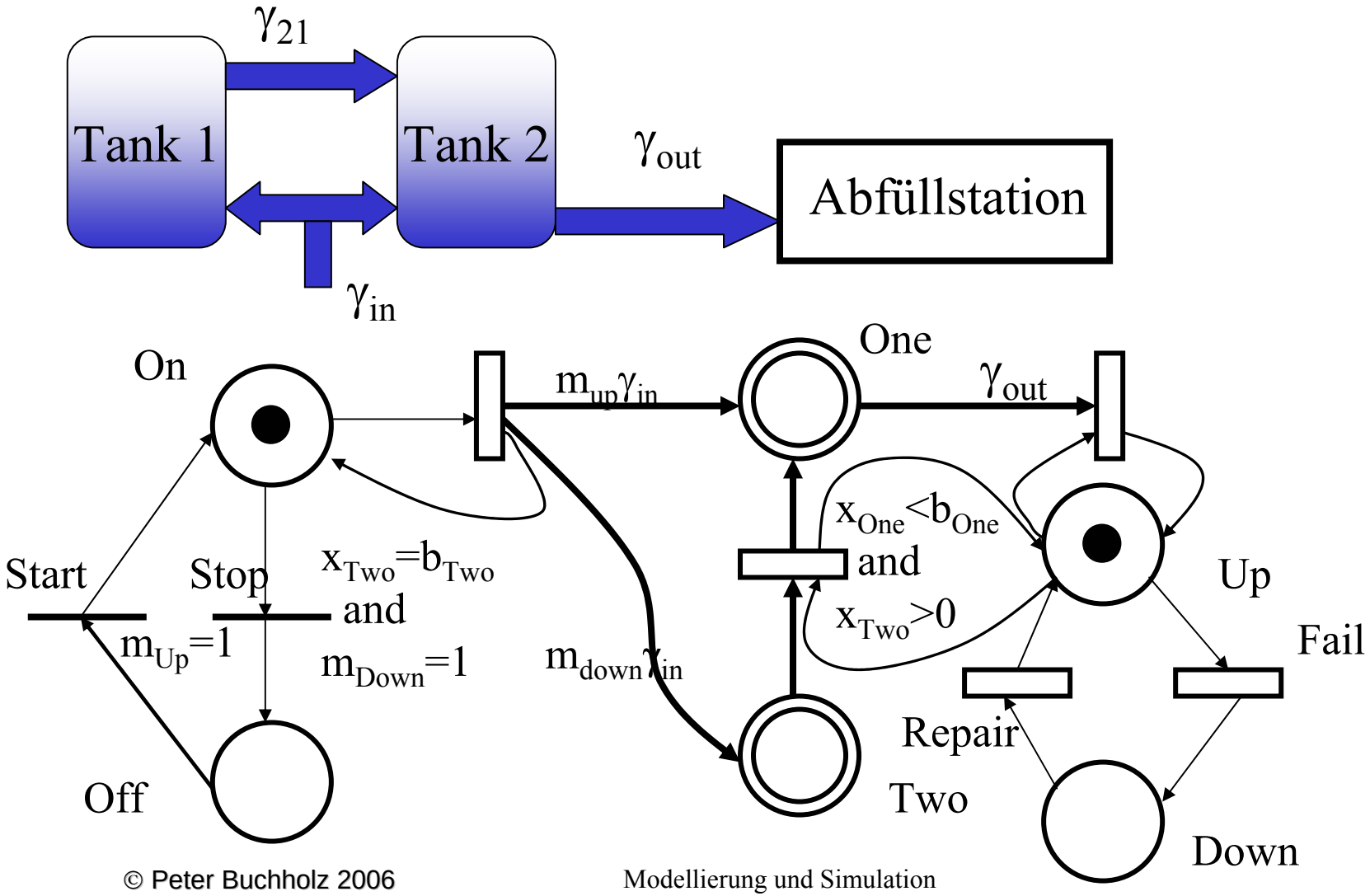
$PN=(S,T,F,W,M_0)$ mit

- S endliche Stellenmenge
- T endliche Transitionsmenge
- $F \subseteq S \times T \cup T \times S$
- M_0 initiale Markierung

Für spezielle Klassen Erweiterung der Definition!

Beispiel: Fluide stochastische Petri-Netze (andere Klassen existieren)

Einfaches Beispiel zwei Tanks



Hybride Automaten:

- Automatemodelle sind in der Informatik in unterschiedlichen Varianten wohl etabliert (zeitlos, stochastisch, probabilistisch, timed, ...)
- hybride Automaten entstanden als Erweiterung von timed automata
  sie vereinen
  - kontinuierliches Verhalten in den Zuständen
  - Ereignisse durch Zustandsübergänge

Def. hybrider Automat (eine von vielen Varianten):

$H=(Q, X, Init, Inv, f, E, G, J, \Sigma)$ mit

- Q endliche Zustandsmenge
- $X \subseteq \mathbb{R}^n$ kontinuierlicher Variablenraum
- $Init \subseteq L \times L$ initiale Zustandsmenge
- $Inv: L \rightarrow 2^X$ Zustandsinvariante
- $f: Q \rightarrow (X \rightarrow X)$ Änderungsfunktion kontinuierlicher Variablen
- $E \subseteq L \times L$ Transitionsmenge
- $G: E \rightarrow L^X$ Guard-Menge
- $J: E \rightarrow (X \rightarrow X)$ Diskontinuitätsfunktion
- S endliche Menge von Transitionslabeln

Beispiel: Pumpe

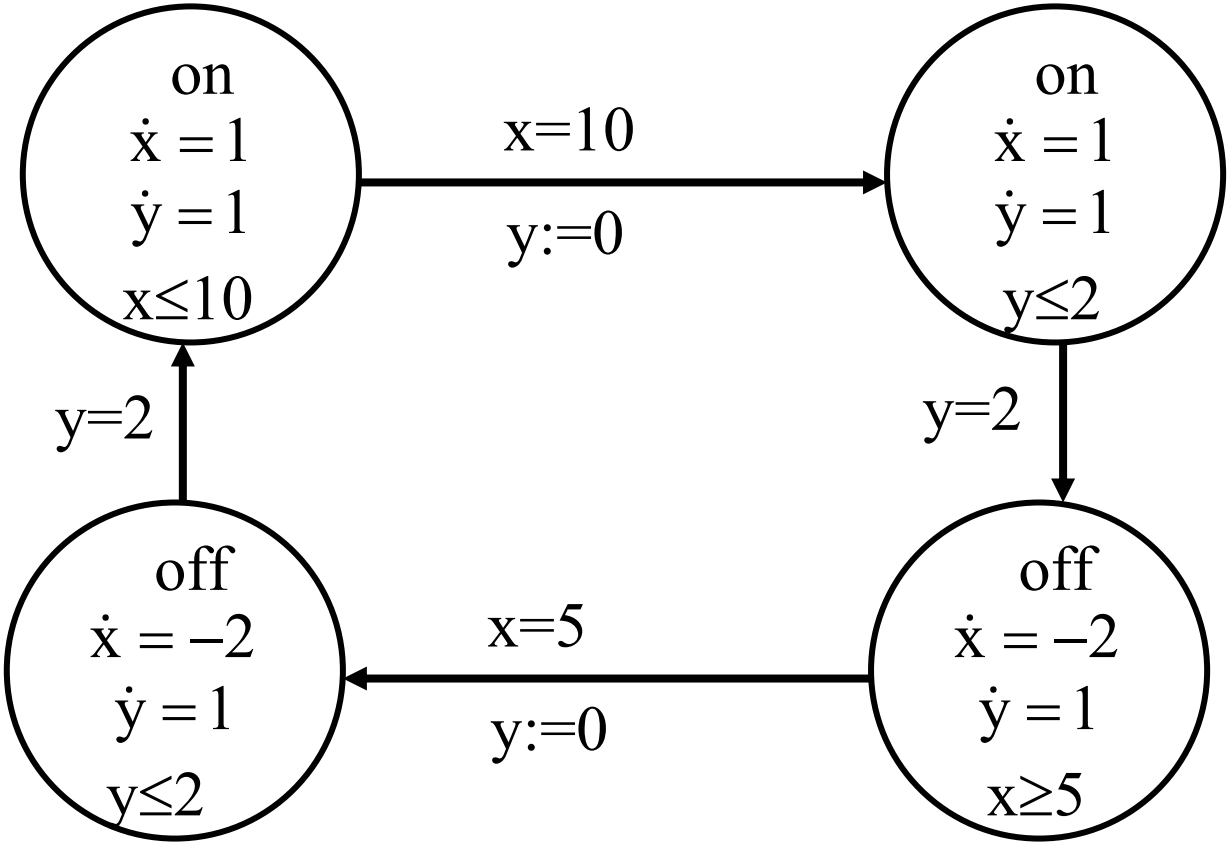
Pumpe on Wasser steigt um 1cm pro Sek.,

off Wasser fällt um 2cm pro Sek.

Signalübertragung zur Pumpe dauert 2 Sek.

x Wasserhöhe, y Zeit seit letzter Sendung

Ziel: Wasserspiegel zwischen 2 und 12 cm



Hybride Automaten

- existieren auch mit Erweiterungen zur stochastischen Modellierung
- erlauben in einigen Varianten auch die Komposition zu Automatennetzen (Varianten, in denen dynamisch neue Automaten erzeugt werden, existieren aber bisher nicht)
- wurden oftmals in eingeschränkten Versionen untersucht, so dass analytische zustandsbasierte Analysetechniken anwendbar sind, Ziele dann Nachweis, dass bestimmte Zustände nicht eintreten können bzw. alle erreichbaren Zustände bestimmte Invarianten erfüllen
- lassen sich zwar in einigen Werkzeugen spezifizieren (Simulink, Matlab u.a.) werden aber nicht primär unter dem Gesichtspunkt der Spezifikation von Simulationsmodellen untersucht