Zusammenfassung EiCE

Release 0.8

Dominic Scheurer

March 03, 2011

INHALTSVERZEICHNIS

KAPITEL

ONE

DIFFERENTIALGLEICHUNGEN

1.1 Gewöhnliche Differentialgleichung

Die bestimmende Gleichung einer Funktion **einer unabhängigen Veränderlichen** heißt *gewöhnliche Differentialgleichung* genau dann, wenn die Gleichung von *der Funktion selbst* und mindestens *einer ihrer Ableitungen* abhängt.

Systeme mit *örtlich konzentrierten Zuständen* werden mit gewöhnlichen Differentialgleichungen (ODE = ordinary differential equation) beschrieben.

In der Vorlesung werden nur Systeme mit örtlich konzentrierten Systemzuständen bahandelt.

1.2 Partielle Differentialgleichung

Die bestimmende Gleichung einer Funktion **mehr als einer unabhängigen Veränderlichen** heißt *partielle Differentialgleichung* genau dann, wenn die Gleichung von *der Funktion selbst* und mindestens *einer ihrer Ableitungen* abhängt.

Systeme mit (lokal) *örtlich verteilten Zuständen* werden durch partielle Differentialgleichungen beschrieben.

1.3 Transformation auf System 1. Ordnung

Jedes System von Differentialgleichungen höherer Ordnung kann auf ein System 1. Ordnung transformiert werden, indem höhere Zeitableitungen als *weitere Zustandsvariablen* hinzugefügt werden.

Beispiel:

$$\ddot{x} + \frac{c}{m} \cdot x = 0$$

$$\Rightarrow x_1 := x, x_2 := \dot{x}, x := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Daraus folgt das System:

$$\dot{x} = f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} := \begin{pmatrix} x_2 \\ -\frac{c}{m} x_1 \end{pmatrix}$$

1.4 Autonomisierung nichtautonomer DGL-Systeme

Ein autonomes Differentialgleichungssystem ist ein System, in dem auf der rechten Seite die Zeit als Variable nicht mehr vorkommt. Jedes System kann durch Einführung einer weiteren Zustandsvariablen für die Zeit "autonomisiert" werden.

Gegeben sei ein nichtautonomes DGL-System mit dem Zustandsvektor $x = (x_1, \cdots, x_n)^T$; führe eine weitere Variable $x_{n+1} := t$ ein. Damit ist $\dot{x}_{n+1} := 1, x_{n+1}(0) := 0$.

Damit erhält man ein modifiziertes, autonomes DGL-System der Dimension n+1, welches für unsere Zwecke äquivalent zum ursprünglichen System der Dimension n ist.

1.5 Lösbarkeit

1.5.1 Richtungsfeld, Lösungstrajektorie

Liegt eine Differentialgleichung 1. Ordnung der Dimension 1 vor, $\dot{x} = f(x(t), t)$, ist für jeden zulässigen Punkt $(x, t) \in R^2$ eine Steigung durch $f(x(t), t)$ und durch die Menge aller Steigungen ein *Richtungsfeld* definiert.

Praktisch trägt man in ein gewöhnliches karthesisches Koordinatensystem an ausgewählte Punkte die dort definierte Steigung ein, indem man die Zeit- und Zustandskoordinaten in die gegebene Funktion f (rechte Seite der DGL) einsetzt und diesen Wert, als Steigung interpretiert, in Form eines Pfeilchens mit Stützpunkt am aktuellen Punkt einträgt.

Liegt ein Anfangswert x(0) vor, erhält man eine **Lösungstrajektorie** x(t), indem man von dem Anfangswert aus dem "Richtungsfeld folgt".

1.5.2 Lipschitz-Bedingung

Ein autonomes DGL-System 1. Ordnung mit Anfangswert hat genau dann eine *eindeutige Lösung*, falls für alle zulässigen x die *Lipschitz-Bedingung* mit geeigneter Lipschitzkonstante L > 0 erfüllt ist:

$$\| f(x_1) - f(x_2) \| \leq L \cdot \| x_1 - x_2 \|$$

Man kann sich nun (im eindimensionalen Falle und für $x_1 \neq x_2$) vorstellen, durch $\| x_1 - x_2 \|$ zu dividieren. Das Ergebnis sieht aus wie ein Differenzenquotient; die Konstante L ist dann interpretierbar als die *betragsmäßig größte Steigung* der Funktion f.

1.6 Lösung eines homogenen Systems von n > 1 DGLen mit imaginären Lösungen

Es liegt ein System von *n* linear gekoppelten Differentialgleichungen vor: $\dot{x(t)} = A \cdot x(t)$. Gewählt wird der übliche Lösungsansatz $x(t) = ce^{\lambda t}$, wobei $c \in \mathbb{R}^n$ und $\lambda \in \mathbb{C}$. Dann empfiehlt sich das folgende Vorgehen:

1. Berechne die **Eigenwerte** der Matrix *A* als Nullstellen des *charakteristischen Polynoms* von *A*: $det(\lambda_i I - A)$, wobei *I* die Identitätsmatrix der Dimension *n* ist.
2. Berechne die **Eigenvektoren** der Matrix *A* als Lösungen der Gleichung $\lambda_i c_i = A c_i$.
3. Die allgemeine Lösung ergibt sich als **Linearkombination der Realteile** der Teillösungen $c_i \cdot e^{\lambda_i t}$. Bei *imaginären Teillösungen* ist zu beachten:

4. Eine imaginäre Teillösung $x_{\mathbb{C}}(t) = c_i \cdot e^{\lambda_i t}$ liefert zwei *relle Lösungen* $\mathrm{R}(x_{\mathbb{C}}(t))$ und $\mathrm{I}(x_{\mathbb{C}}(t))$. Diese erhält man durch Anwendung der Eulerschen Formel $\mathrm{e}^{\mathrm{i}\,\varphi} = \cos(\varphi) + \mathrm{i}\,\sin(\varphi)$ und Trennung der Teile der Lösung, die imaginär sind (es kommt ein i vor) von denen, die rein reell sind. Anschließend werden beide Teile getrennt interpretiert, d.h. bei der Auswertung von $\mathrm{R}(x_{\mathbb{C}}(t))$ werden die imaginären Anteile ignoriert und entsprechend bei der Auswertung von $\mathrm{I}(x_{\mathbb{C}}(t))$ die reellen Anteile. Dabei kann, umgangssprachlich gesprochen, die imaginäre Einheit (i) einfach "weggelassen" werden, sodass sie bei $\mathrm{I}(x_{\mathbb{C}}(t))$ nicht mehr vorkommt (verlangt werden schließlich reelle Lösungen).

   Somit ergibt *jede imaginäre Teillösung* zwei reelle Lösungen, die mit eigenen Koeffizienten in die Linearkombination, die die allgemeine Lösung ergibt, mit eingehen. **Wichtig**: Teillösungen, die sich aus Eigenwerten und -vektoren ergeben, die das komplexe Komplement einer anderen, bereits aus- gewerteten Teillösung sind, liefern dieselben reellen Lösungen und müssen *nicht berücksichtigt* werden.

1.7 Variation der Konstanten

Mit der Methode der *Variation der Kontanten* lassen sich (Systeme) *inhomogener Differentialgleichungen erster Ordnung* lösen (erste Ordnung kann o.B.d.A. angenommen werden, siehe *Transformation auf System 1. Ordnung*).

Es liegt ein System der folgenden Art vor:

$$\dot{x(t)} = Ax(t) + b;\ x, b \in \mathbb{R}^n,\ A \in \mathbb{R}^n_n$$

Die Methode funktioniert mit den folgenden Schritten:

1. Lösung des *homogenen Problems* $\dot{x(t)} = Ax(t)$ mit üblichen Mitteln (siehe z.B. *Lösung eines homogenen Systems von n > 1 DGLen mit imaginären Lösungen*). Die Lösung dieses Problems (**homogene Lösung**) wird mit $x_h(t)$ bezeichnet.

2. Ermittelung der **speziellen Lösung** (auch "partikuläre Lösung") durch *Variation der Konstanten* aus der homogenen Lösung.

   Das heißt: Die Konstante (hier *C* genannt) aus dem homogenen Problem wird ersetzt durch eine Funktion *C(t)*. Nun wird die so modifizierte homogene Lösung nach der Zeit abgeleitet (die Ableitung von *C(t)* ist einfach *C'(t)*). Diese Ableitung wird für $\dot{x(t)}$ aus dem inhomogenen Problem, die modifizierte homogene Lösung selbst für $x(t)$ aus dem inhomogenen Problem eingesetzt. Nach der Einsetzung wird mit Integration und anderen Mitteln nach *C(t)* aufgelöst; dieses wird dann eingesetzt in die homogene Lösung. Das Ergebnis ist die spezielle Lösung $x_s(t)$.

3. Die allgemeine Lösung des inhomogenen Problems ergibt sich durch die Summe aus homogener und spezieller Lösung:

$$x(t) = x_h(t) + x_s(t)$$

KAPITEL

TWO

SIMULATION - GRUNDLAGEN

2.1 Schritte einer Simulationsstudie

Die grundlegenden Schritte einer Simulationsstudie sind

- Problemspezifikation
- Modellierung
- Implementierung
- Validierung
- Anwendung

2.1.1 Validierung

Validierung:

- Systematische Plausibilitätsprüfung
- Reduziert die Wahrschl. falscher Schlussfolgerungen aus der Simulation
- Beruht auf Sorgfältig ausgewählten Tests

Möglichkeiten zur Validierung:

- Vergleich mit Experimenten
- A-posteriori-Beobachtungen (also Beobachtungen *nach* Modelleinsatz, bspw. Zufriedenheitstest)
- Plausibilitätstest (Prüfung auf Konsistenz mit versch. Theorien)
- Modellvergleich (Vgl. der Simulationsergebnisse untersch. Modelle)

2.2 Ziele einer Simulation

Mit Computersimulationen kann man

- **Bekannte** Szenarien **verstehen** bzw. **nachvollziehen**
- **Bekannte** Szenarien **optimieren**
- **Unbekannte** Szenarien **vorhersagen**

2.3 Modellbegriff

Mögliche, allgemeine Definitionen für den Modellbegriff:

- (Vereinfachende) Abbildung einer (partiellen) Realität
- Abstrakte, logische und mathematische Darstellung der Objekte und Wechselbeziehungen in einem System
- Ersatzsystem, gebildet unter Annahmen und Idealisierung

Ein **abstraktes Modell** ist eine formale, üblicherweise mathematische Beschreibung des Modells.

Die **mathematische Modellierung** ist der Prozess der formalen Herleitung und Analyse eines mathematischen Modells:

1. Informelle, **prosaische** Problembeschreibung
2. Semi-formale Beschreibung mit dem **Instrumentarium der Anwendungswissenschaft**
3. Streng formale, **konsistente** Modellbeschreibung

Modelle lassen sich in die folgenden Klassen einteilen:

- Zeitkontinuierlich, wertkontinuierlich
- Zeitdiskret, wertdiskret
- Zeitdiskret, wertkontinuierlich
- Ereignisdiskret, Wertdiskret
- Stochastisch

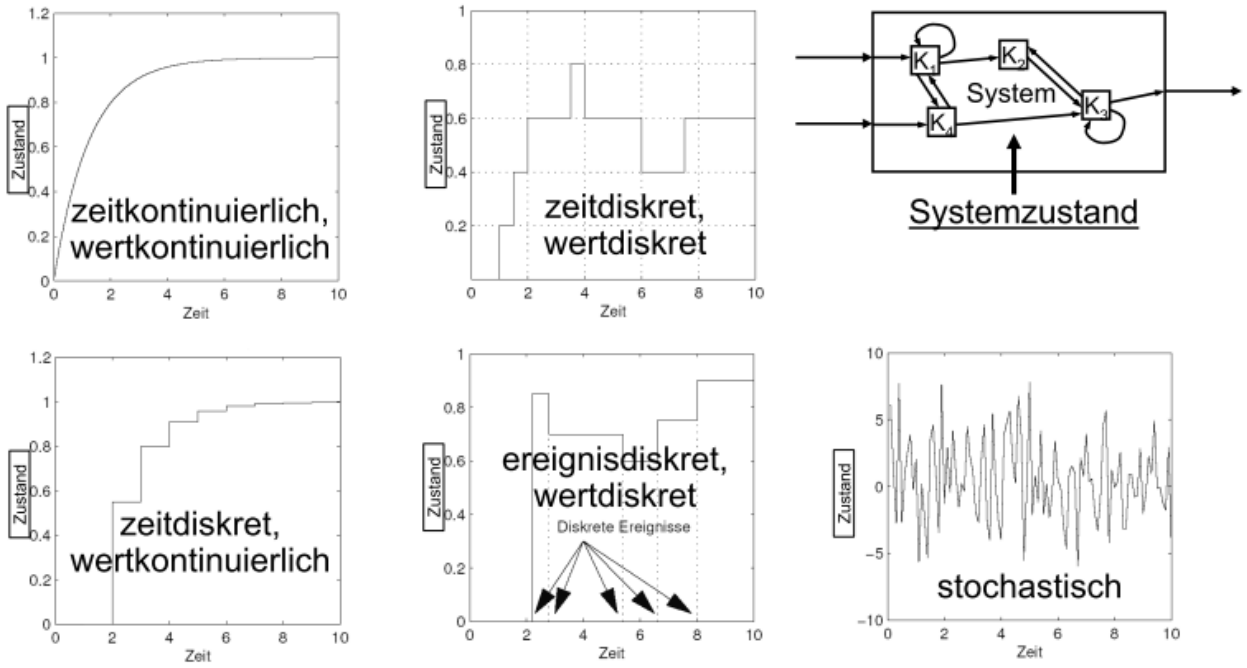

Abbildung 2.1: Modellklassifikation nach Kontinuität von Zustand und abhängiger Variable

2.4 Systemaufbau

Ein System besitzt

- Während des Simulationslaufs konstante **Systemparameter**
- Steuerbare (Stellgrößen) sowie nicht kontrollierbare (Störgrößen) **Systemeingänge**
- Systemausgänge (messbare / beobachtbare Größen)
- Einen **Systemzustand** (vollständige Charakterisierung des Modellverhaltens)

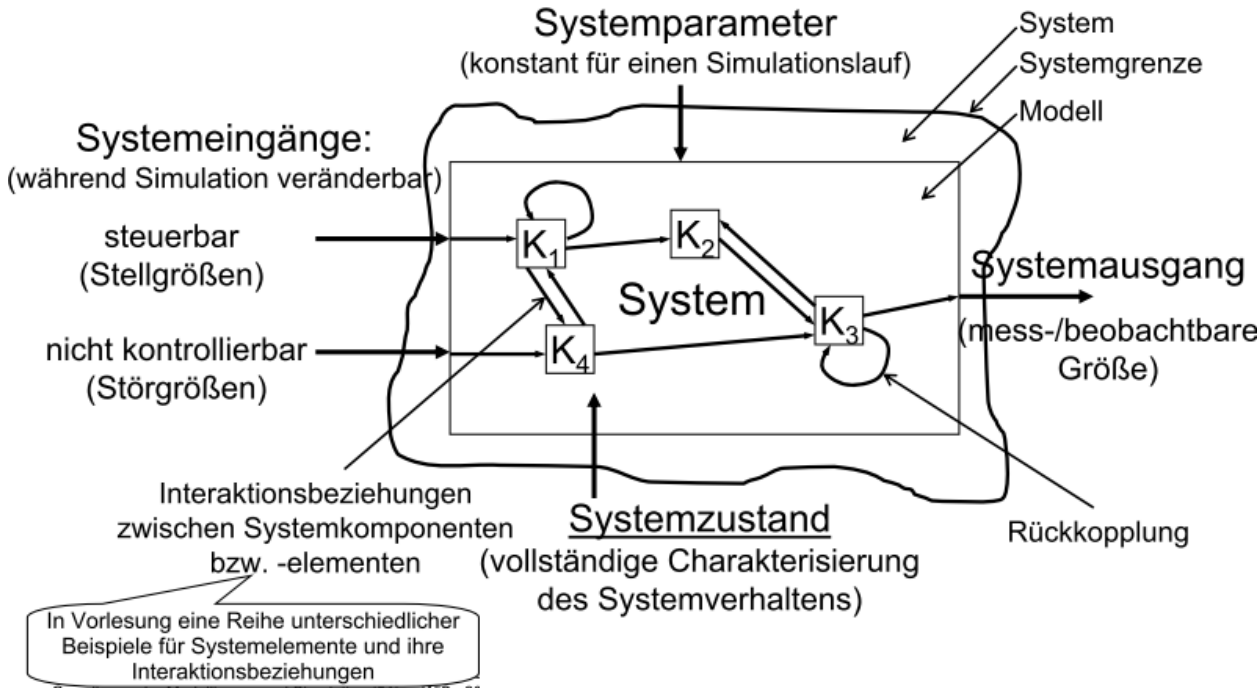

Abbildung 2.2: Modell, System, Eingänge / Ausgänge / Zustand: Skizze

KAPITEL

THREE

DISKRETE MODELLIERUNG UND SIMULATION

Es gibt unterschiedliche *"diskrete" Modelle*:

Zeitdiskrete Modelle: Die Zeit t wird diskretisiert: Nur zu ausgewählten Zeiten ist Beobachtung / Steuerung möglich.

Häufig ist ein zeitdiskretes Modell eine äquidistant zerlegte Variante des entsprechenden zeitkontinuierlichen Modells.

Nicht äquidistant => Häufig ereignis-diskrete Beschreibung bevorzugt.

Qualitative Modelle: Die Zeitachse ist *von Natur aus* diskret, nicht allerdings notwendigerweise äquidistant. Die abhängige Variable (Zustand) ist ebenfalls diskret(isiert). Bsp.: Ampelsignale.

Ereignis-diskrete Modelle: Zeit und Zustand oft **kontinuierlich**! Modellierung des eigentlich kontinuierlichen Übergangs eines Zustands in einen Folgezustande als diskretes Ereigbnis (falls nur *qualitative* Änderung von Interesse).

3.1 Ereignis-diskrete Modellierung und Simulation

Synonym: DEVS = Descrete Event Simulation.

Die DEVS ist einer der am häufigsten angewanden und etabliertesten Simulationsmethoden.

3.1.1 Terminologie

System: Ansammlung von *Objekten*, die *zeitabhängig* nach spezifizierten *Regeln* miteinander interagieren.

Modell: *Abstrakte*, *logische* und *mathematische* Darstellung der Objekte und *Wechselbeziehungen* in einem System.

Entität: Explizit dargestelltes Objekt in einem System (bspw. Kunde, Werkstück)

Attribut: Variable, die den *Zustand* einer Entität beschreibt

Ereignis: Beschreibung der Aktualisierung des Modellzustands

Ereignisliste: Nach Zeitpunkten aufsteigend geordnete Liste von *Zeitpunkt* und *Typ* der *geplanten Ereignisse*

Aktivität: Zeitlich erstreckter *Vorgang* zwischen *initiierenden* und abschließenden Ereignissen einer Operation, die *den Zustand einer Entität transformiert*.

Simulationsuhr: Variable, die den *aktuellen* Stand der *Simulationszeit* angibt.

Zeitführungsrouting: Prozedur zur *Auswahl des nächsten Ereignisses* und *Vorstellen der Simulationsuhr* auf den nächsten Ereigniszeitpunkt.

Ergebnisroutine: Prozedur zur *Erechnung* der statistischen *Schätzwerte der Ergebnisvariablen* (anhand statistischer Zähler) und zur Ausgabe des Ergebnisprotokolls (nach Simulationsende)

Steuerprogramm: Programmteil, der *wiederholt* die *Zeitführungsroutine* zur Bestimmung des nächsten Ereignistyps und die zugehörige *Ereignisroutine aufruft*, bis die *Simulation beendet* ist.

3.1.2 Funktionsweise

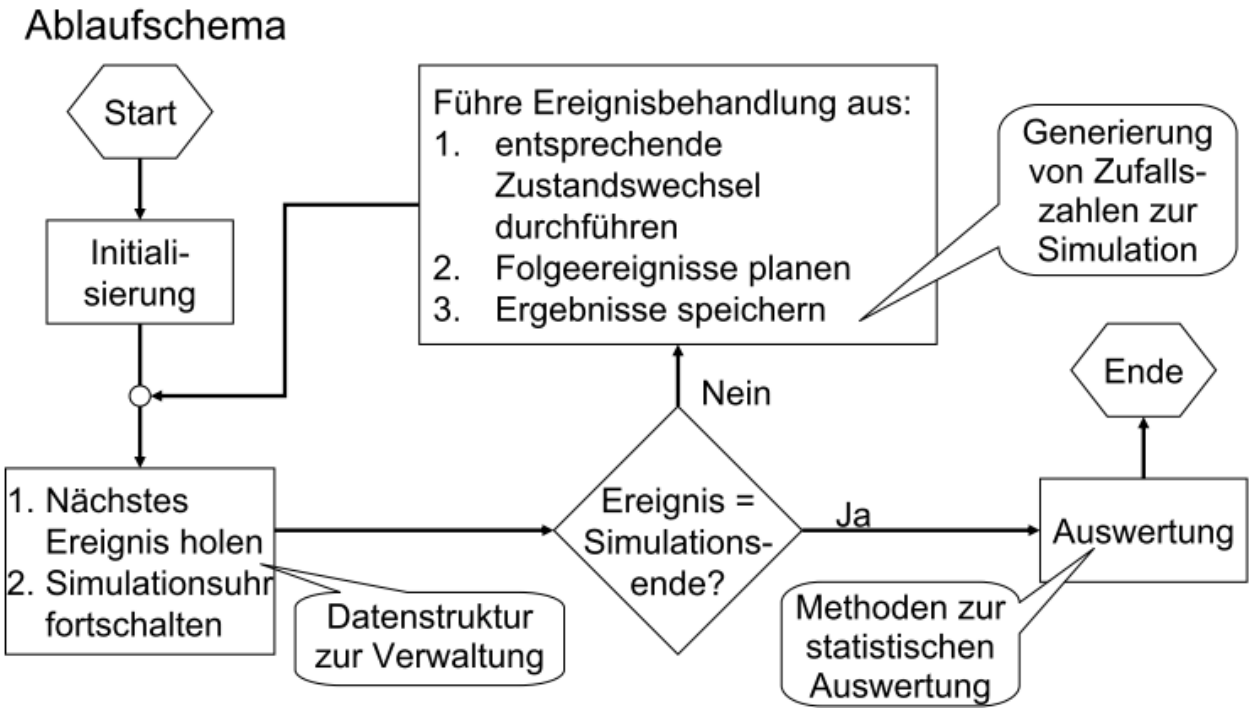

Abbildung 3.1: Funktionsweise der ereignisbasierten Simulation (DEVS) - Ablaufschema

Der folgende Pseudocode-Ausschnitt verdeutlicht den **zentralen Ereignisalgorithmus**. Dabei sind L die Liste der Zeitpunkt-Ereignis-Paare und t die Simulationszeit:

```
// Initialisierung
L := { (t1,E1) };      // Startereignis
t := 0;                // Uhr starten
initializeSystem;      // Systemspezifische Startwerte

// Ereignisschleife
while (t < t_max) {
        t := t1;             // Nächstes Ereignis
        E := E1;
        L := L \ { (t1,E1) } // Entferne Ereignis aus Warteliste

        eventRoutine (E);    // Passende Ereignisroutine

        sort (L)             // Nach Zeitpunkten aufsteigend sortieren
}
```

Weitere Routinen sind die Initialisierungsmethode initializeSystem und die Ereignisorutinen eventRoutine, arrivalRoutine und departureRouting:

```
// Systeminitialisierung
function initializeSystem {
        N := 0;      // N -> Länge der Warteschlange
        S := _idle;  // S -> Zustand des Servers. Element von {_idle, _busy}
}

// Ereignisroutine
function eventRoutine (E) {
        switch E {
                A: arrivalRoutine;
                D: departureRoutine;
        }
}

// Ankunftsbehandlung
function arrivalRoutine {
        // Anlegen eines neuen Ankunfts-Ereignisses
        L := L <> { (t + exp (lambda),A) };       // Ankünfte sind exponentiell
                                                  // verteilt => exp (lambda)

        if (S = _busy) {
                N := N + 1;
        } else {
                S := _busy;

                // Anlegen eines neuen Verlassens-Ereignisses
                L := L <> { (t + norm (mu, sigma),D) }; // Belegungszeiten sind
                                                        // normal verteilt
        }
}

// Verlassensbehandlung
function departureRoutine {
        if (N = 0) {
                S := _idle;
        } else {
                N := N - 1;

                // Anlegen eines neuen Verlassens-Ereignisses
                L := L <> { (t + norm (mu, sigma),D) }; // Belegungszeiten sind
                                                        // normal verteilt
        }
}
```

3.2 Petrinetze

Ein Petrinetz ist ein abstraktes, formales Modell eines Informationsflusses; besonders geeignet zur Modellierung von **ereignisdiskreten** Systemen mit *nebenläufigen* Ereigniseintritten und gleichzeitiger *Beschränkung* an Art, Häufigkeit und Vorgängern dieser Ereignisse.

3.2.1 Definition

Beschrieben wird ein Petrinetz durch ein **Fünf-Tupel** $PN = (P, T, A, K, M')$ aus

- einer endlichen Menge P von **Plätzen**, die mögliche Zustände beschreiben (Darstellung als Kreis)
- einer endlichen Menge T von **Transitionen**, die mögliche Ereignisse beschreiben (Darstellung als Rechteck)
- einer endlichen Menge A von gerichteten **Kanten**, welche Plätze und Transitionen verbinden.
- einer Menge $K = \left\{k_1, k_2, \cdots, k_{|P|}\right\}$ mit $|K| = |P|$, die die maximale **Kapazität** der jeweiligen Plätze festlegt, wobei die Kapazität des Platzes i gleich k_i ist.
- einer Anfangsmarkierung M', die jedem Platz seine anfängliche Anzahl an Markierungen zuordnet. Auch hier ist $|M'| = |P|$.

3.2.2 Mathematische Darstellung

Mathematisch dargestellt werden Petrinetze durch eine Inzidenzmatrix W, die sich aus der Differenz zweier Matrizen W^- (Verbindungen eines Platzes mit einer Transition) und W^+ (Verbindungen einer Transition mit einem Platz) ergibt: $W = W^+ - W^-$. Bei allen Matrizen stehen die Zeilen für die Plätze und die Spalten für die Transitionen.

Achtung: Für jedes Petrinetz gibt es genau eine Inzidenzmatrix, aber nicht für jede Matrix $W = W^+ - W^-$ gibt es *genau ein* Petrinetz.

3.2.3 Simulation

Bei der Simulation wird neben den gegebenen Initialisierungsinformationen ein Vektor M, der Zustandsvektor, gespeichert; dieser repräsentiert die Anzahl von Markierungen m_i in allen Plätzen p_i zu diskreten Zeitpunkten r: $M(r) = [m_1(r), m_2(r), \cdots, m_n(r)]$.

Zur "**Schaltung**" einer Transition zur Änderung des Markierungsvektors müssen

1. In allen "Vor-Plätzen" ausreichend Markierungen verfügbar sein (beim Schalten wird an jedem "Vor-Platz" eine Markierung abgezogen und
2. In allen "Nach-Plätzen" die maximale Zahl an Markierungen (Kapazität) nicht überschritten werden.

3.2.4 Charakteristische Eigenschaften

Erreichbarkeit: Ein Zustand heißt erreichbar von einem Anfangszustand, falls eine Schaltsequenz vom Anfangszustand zu diesem existiert (-> Erreichbarkeitsgraph)

Beschränktheit, Safety: Ein Petrinetz heißt beschränkt, falls es an keinem Platz p_i jemals mehr als eine bestimmte Zahl k_i an Markierungen gibt.

Verklemmung (Deadlock): Eine Verklemmung eines Petrinetzes ist ein Zustand, in dem keine Transition mehr schalten kann.

Lebendigkeit: Eine Transition heißt *tot*, falls sie bei keiner Folgemarkierung mehr aktivierbar ist. Ein Petrinetz heißt *nicht-lebendig*, falls es mindestens eine tote Transition enthält.

3.3 Endliche Zustandsautomaten

Endliche Zustandsautomaten sind Grundbestandteile der Simulationssprache XABSL ("Extensible Agent Behavior Specification Language").

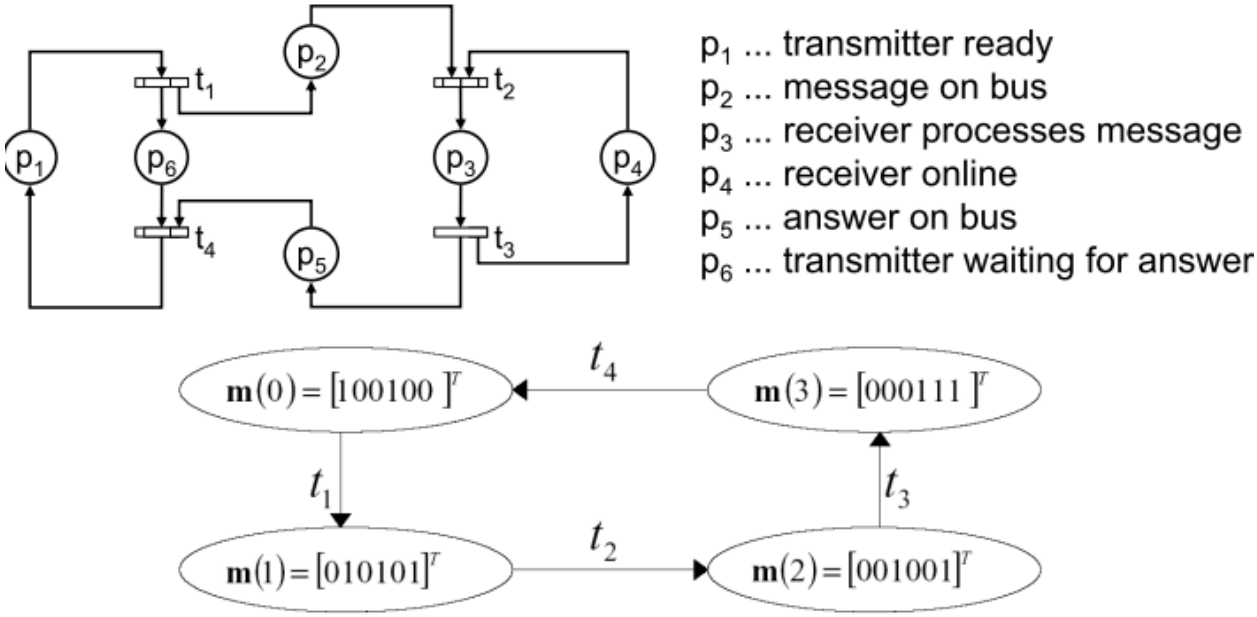

Abbildung 3.2: Erreichbarkeitsgraph eines Petrinetzes. Hier können alle Zustände von jedem Startzustand erreicht werden.

Ein endl. Zustandsautomat (engl. DFA, "deterministic finite automaton") ist spezifiziert als ein Fünf-Tupel $A = \{\Sigma, Q, q_0, \delta, Q_{akz}\}$, wobei für die Bedeutung der Tupel gilt (siehe Tabelle):

Σ	Alphabet
Q	Endliche, nicht-leere Zustandsmenge
q_0	Anfangszustand
$Q_{akz} \subseteq Q$	Menge der akzeptierenden Zustände
$\delta : Q \times \Sigma \rightarrow Q$	Übergangsfunktion

KAPITEL

FOUR

ZEITKONTINUIERLICHE MODELLIERUNG UND SIMULATION

4.1 Grundlagen

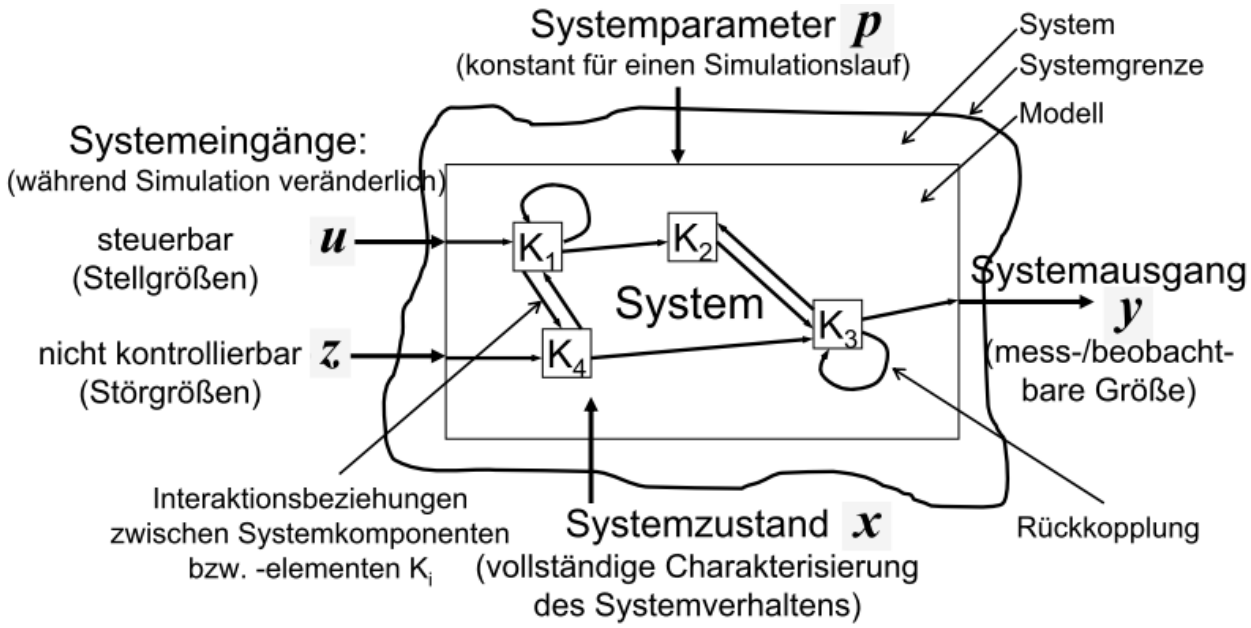

Abbildung 4.1: Zeitkontinuierliche Modellierung und Simulation. Systemvariablen p, x, u, z und y.

In der zeitkontinuierlichen Simulation werden Modelle durch allgemeine Zustandsgleichungen, d.h. Systeme von gewöhnlichen Differentialgleichungen 1. Ordnung beschrieben. Das **Zustandsraummodell** besteht aus einer **Systemfunktion** $\dot{x} = f(t, x, u, p)$ und einer **Ausgangsfunktion** $y = g(t, x, u, p)$:

$$\dot{x}(t) = \frac{dx(t)}{dt} = \begin{pmatrix} \dot{x_1}(t) \\ \vdots \\ \dot{x_n}(t) \end{pmatrix} = \begin{pmatrix} f_1(x(t), u(t), t) \\ \vdots \\ f_n(x(t), u(t), t) \end{pmatrix} = f(x(t), u(t), t)$$

$$y = g(x, u, t) = (g_1(x, u, t), \cdots, g_m(x, u, t))^T$$

Beachte: Es kann auch $m \neq n$ sein!

$x = (x_1, \cdots, x_n)^T$ sind Zustandsvariablen, $u = (u_1, \cdots, u_n)^T$ sind Steuervariablen, $y = (y_1, \cdots, y_m)^T$ sind Ausgangsgrößen.

Beachte: Es ist keine Einschränkung, dass Systeme **1. Ordnung** gefordert werden, siehe *Transformation auf System 1. Ordnung*.

4.2 Allgemeine, lineare Systemdynamik

Eine wichtige Teilklasse zeitkontinuierlicher Modelle ist die lineare Systemdynamik:

$$\dot{x} = f(x,u) = Ax + Bu$$
$$y = g(x,u) = Cx + Du$$

A, B, C und D sind konstante (oder rein zeitabhängige) Koeffizientenmatrizen, deren Dimensionen so beschaffen sind, dass sie mit dim(x) = n, dim(y) = m und dim(u) = k verträglich sind.

4.3 Stationäre Zustände

Für ein konstantes $u(t) = u_s$ bezeichnet man jede Lösung x_s der Gleichung $0 = f(x_s, u_s)$ als **Gleichgewichtslösung**, den zugehörigen Zustand als *stationären Zustand* oder **Ruhezustand**.

Zur Bestimmung der Gleichgewichtslösung unterscheidet man zwischen

Linearer Systemdynamik: $\dot{x} = Ax + Bu \Rightarrow Ax_s = -Bu_s$

Die eindeutige Lösung existiert hier genau dann, wenn $Rg(A) = n$ bzw. $det(A) \neq 0$ (*A* ist invertierbar).

Nichtlinearer Systemdynamik: $\dot{x} = f(x,u) \Rightarrow 0 = f(x_s, u_s)$

Hier können keine / eine / mehrere / unendlich viele Lösungen existieren.

4.4 Approximation der Jacobi-Matrix

Die Jacobi-Matrix der rechten Seite einer Differentialgleichung $\dot{x} = f(x,u)$ hinsichtlich der Zustandsvariablen x ist definiert als:

$$\frac{\delta f}{\delta x} = \frac{\delta f_i}{\delta x_j}_{i,j=1,\cdots,n} = \begin{pmatrix} \frac{\delta f_1}{\delta x_1} & \cdots & \frac{\delta f_1}{\delta x_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta f_n}{\delta x_1} & \cdots & \frac{\delta f_n}{\delta x_n} \end{pmatrix}$$

Man kann die Jacobi-Matrix einerseits wie oben gezeigt analytisch berechnen, andererseits aber auch numerisch annähern, z.B. mit dem **Vorwärtsdifferenzenquotienten**:

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

e_j ist hier der j-te Einheitsvektor der Dimension n.

Diese Approximation hängt sehr ab von der Wahl der Schrittweite δ_j, die man zum Beispiel wählen kann wie folgt: $\delta_j = \epsilon \left(1 + |x_j|\right)$. Je nach $|x_j|$ nennt man dies eine "relative" Schrittweite ($|x_j| >> 1$ "groß") bzw. eine "absolute" Schrittweite ($|x_j| << 1$ "klein").

Den Parameter ϵ nennt man "Toleranz" und wählt dafür bspw. die Wurzel der relativen Maschinengenauigkeit.

Die gesamte Approximation fordert im Wesentlichen n+1 Auswertungen von f. Dabei liefert sie (Faustregel) maximal *die Hälfte der gültigen Dezimalstellen* von f! Die Genauigkeit steigt nicht immer bei kleinerem δ; denn wenn dieser

Wert zu klein wird, steigt der Einfluss der Rundungsfehler (die aufgrund der endlichen Rechenaritmetik der Maschine aufkommen).

Weitere Arten der Approximation sind

Symbolisches Differenzieren: Keine Approximationsfehler (nur Rundungsfehler), lange Formeln, hoher Berechnungsaufwand.

Automatisches Differenzieren Vorwärts- oder Rückwärtsmodus; Keine Approximationsfehler (nur Rundungsfehler), bei Rückwärtsmodus effiziente Gradientenberechnung und (oft) effiziente Berechnung der Jacobi-Matrix.

4.5 Linearisierung um die Ruhelage

Oft sollten dynamische Systeme in einem vorgegebenen, stationären Betriebszustand arbeiten: Bei möglichst konstanten Eingangsgrößten u_s sollen möglichst konstante Ausgangsgrößen x_s erzeugt werden. Problem: Umgebungsstörungen drängen $x(t)$ vom idealen Verhalten x_s ab.

Untersucht werden daher Abweichungen von der Ruhelage Δx und eine DGL für Δx:

$$\dot{\Delta x} = \left|\frac{\delta f}{\delta x}\right|_{x_s,u_s} \cdot \Delta x + \left|\frac{\delta f}{\delta u}\right|_{x_s,u_s} \cdot \Delta u$$

An Ruhelagen bei Sprung-/ Knick- oder ähnlichen Ausnahmestellen kann *nicht* linearisiert werden!

4.6 Stabilität

Um zu Untersuchen, ob eine Gleichgewichtslage *statil* ist, werden die Eigenwerte der an der Gleichgewichtslage linearisierten systembeschreibenden Differentialgleichung berechnet und deren Realteile interpretiert;

Negative Realteile: Gedämpfte Oszillation, *stabiles* System

Mindestens ein positiver Realteil: *Instabiles* System.

Mindestens eine Polstelle hat Realteil 0: Mit der Näherungsgleichung allein ist keine Aussage über die Stabilität der Gleichgewichtslage möglich.

4.7 Zeitcharakteristik

Die Zeitcharakterisika T_i werden mit Hilfe der Eigenwerte λ_i des linearisierten Systems bestimmt, in Abhängigkeit von deren Beschaffenheit:

Beschaffenheit der λ_i	**Zeitcharakteristika**
Relles λ_i	$T_i = \frac{1}{\lvert\lambda_i\rvert}$
Rein imaginäres λ_i	$T_i = \frac{2\pi}{\lvert\lambda_i\rvert}$
Konjugiert komplexes λ_i	$T_i = min\left\{\frac{1}{\lvert Re(\lambda_i)\rvert}; \frac{2\pi}{\lvert Im(\lambda_i)\rvert}\right\}$

T_{max} und T_{min} sind die maximalen bzw. minimalen Zeitcharakteristika. Es existieren Faustregeln, wie man daraus eine *sinnvolle Simulationsdauer* abschätzen kann: Bei einem stabilen System vergeht bis zum Erreichen der Gleichgewichtslage ungefähr $t_f = 5 \cdot T_{max}$. Für die *Diskretisierungsschrittweite* sollte man $h = \Delta t \leq \alpha \cdot T_{min}$ wählen ($\frac{1}{20} \leq \alpha \leq \frac{1}{5}$.

4.8 Steife Systeme / Differentialgleichungen

Stabile Differentialgleichungen mit sehr *unterschiedlichen Zeitcharakteristika* werden steife Differentialgleichungen genannt (ein "steifes System" ist ein System mit einer beschreibenden steifen Differentialgleichung).

Das Verhältnis der minimalen und maximalen Zeitkonstangen nennt man **Steifheitsmaß**. Bei Steifheit gilt dafür üblicherweise, dass

$$\frac{T_{max}}{T_{min}} > 10^3 \cdots 10^7$$

Die Rechenzeit zur numerischen Lösung einer steifen DGL ist groß und proportional zum Steifheitsmaß; man verwendet angepasste *implizite Verfahren*.

4.9 Bilanzgleichungen

In einigen Szenarien ist es sinnvoll, die zeitliche Änderung einer Erhaltungsgröße (z.B. Masse, Energie, Impuls, ...) als *Differenz* der Summen der *Zugewinne* und *Verluste*, die die Änderung ergibt, darzustellen.

Bsp. Tankmodellierung: die zeitliche Änderung der Masse im Tank (also deren Ableitung nach der Zeit) ist gleich der Differenz von Zufluss und Abfluss in / aus dem Tank.

4.10 Linearisierung um eine Referenztrajektorie

Statt ein System um eine konstante Ruhelage x_s, u_s zu linearisieren, kann es auch um eine *zeitveränderliche Referenztrajektorie* linearisiert werden; das heißt im Wesentlichen, dass aus x_s, u_s jetzt $x_s(t), u_s(t)$ wird und $\dot{x_s}(t) = f\left(x_s(t), u_s(t)\right)$ gilt.

Hier werden die Abweichungen von der Referenztrajektorie durch das lineare DGL-System

$$\dot{\Delta x} = A(t) \cdot \Delta x + B(t) \cdot \Delta u$$

bestimmt. Die Koeffizientenmatrizen sind in diesem Fall *zeitveränderlich* (die Linearisierung ist interpretierbar als Linearisierung um eine zeitveränderliche Ruhelage). Die Matrizen A und B sind wie folgt beschaffen:

$$\begin{aligned} A(t) &= \left|\frac{\delta f}{\delta x}\right|_{x_s,u_s} \\ &= \left(\frac{\delta f_i(x_s(t), u_s(t))}{\delta x_k}\right)_{i,k=1,\cdots,n} \\ B(t) &= \left|\frac{\delta f}{\delta u}\right|_{x_s,u_s} \\ &= \left(\frac{\delta f_i(x_s(t), u_s(t))}{\delta u_k}\right)_{i,k=1,\cdots,n} \end{aligned}$$

4.11 Regelung

Vorab: Man unterscheidet bei der Benutzung des englischen Fachbegriffs "Control" je nach Kontext:

(open loop / feedforward) control	Steuerung
(closed loop / feedback) control	Regelung

4.11.1 Dämpfung

Beim Vorliegen einer *linearen* Differentialgleichung mit *konstanten Koeffizienten* kann deren Lösung explizit berechnet werden. Die Nullstellen der **charakteristischen Gleichung** bestimmen das "Dämpfungsverhalten" der Masse. Beachte zum Beispiel die folgende "normalisierte" Differentialgleichung (es wurde nur durch den ersten Koeffizienten geteilt):

$$\ddot{x(t)} + \frac{b}{a}\dot{x(t)} + \frac{c}{a}x(t) = 0$$

Mit Einsetzen des *allgemeinen Lösungsansatzes* $x(t) = ce^{\lambda t}$ (mit geeigneten Konstanten c und λ) erhält man (nach Division durch $e^{\lambda t}$) die charakteristische Gleichung

$$\lambda^2 + \frac{b}{a}\lambda + \frac{c}{a} = 0$$

deren Nullstellen λ_1, λ_2 man analytisch berechnen kann:

$$\lambda_{1,2} = -\frac{b}{2a} \pm \frac{\sqrt{b^2 - 4ac}}{2a}$$

Dann lässt sich die folgende Fallunterscheidung über die Art der Nullstellen ("Pole") durchführen:

$\lambda_{1,2}$ ***einfache, reelle*** **Nullstellen:** Dann ist $b^2 - 4ac > 0 \Leftrightarrow b^2 > 4ac$, daraus folgt, dass (bei positiven Parametern) die Pole negativ sind. Aufgrund dessen geht für große t die Lösung der Differentialgleichung ($x(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}$) gegen Null. Dieses Systemverhalten nennt man **überkritisch gedämpft** (nicht oszillierende, aber langsame Systemantwort).

$\lambda_{1,2}$ ***doppelte, reelle*** **Nullstelle:** Dann ist $b^2 = 4ac$ und $\lambda_{1,2} = \frac{-b}{2a} < 0$; die Lösung hat die selbe Form wie im vorherigen Fall und ist wiederrum abklingend. Dieses Systemverhalten nennt man **kritisch gedämpft** (schnellste, nicht oszillierende Systemantwort).

$\lambda_{1,2}$ ***einfache, komplexe*** **Nullstellen:** Dann ist $b^2 < 4ac$ und

$$\begin{aligned}\lambda_{1,2} &= -\frac{b}{2a} \pm i\frac{\sqrt{|b^2 - 4ac|}}{2a} \\ &:= \lambda_{real} \pm i \cdot \lambda_{imag}\end{aligned}$$

Der Realteil der allgemeinen Lösung der Differentialgleichung lautet somit

$$x(t) = c_1 e^{\lambda_{real} t} \cos(\lambda_{imag} t) + c_2 e^{\lambda_{real} t} \sin(\lambda_{imag} t)$$

Da $\lambda_{real} < 0$ geht auch hier die Lösung der DGL gegen Null für große t, allerdings mit einer um die Ruhelage *oszillierenden Bewegung*. Dieses Verhalten nennt man **unterkritisch gedämpft**.

Üblicherweise wir eine *kritische Dämpfung* gewünscht: Schnellstmögliche Rückkehr in die Ruhelage ohne oszillierende Bewegungen.

Todo

- Proportional-Differential-Regelung linearer Systeme zweiter Ordnung
- Feedback-Linearisierung
- Sollwerttrajektorien-Folgeregelung

Siehe: unterlagen-3_3_13-EinfCE.pdf

KAPITEL

FIVE

NUMERISCHE SIMULATION

5.1 Gleitpunktzahlen - IEEE 754

5.1.1 Grundlagen

Gleitpunktzahlen bestehen nach IEEE 754 Standard aus einem *Vorzeichenbit* sowie aus, je nach Genauigkeit (Single / Double) einer festgelegten Anzahl an Bits, die die *Mantisse* sowie den *Exponenten* darstellen. Da die normalisierte Mantisse stehts größer 1 ist, wird deren erstes Bit nicht gespeichert und ist nur implizit vorhanden.

Steht *S* für das Vorzeichen (-1 oder +1), *M* für den abgespeicherten Wert der Mantisse ($0 \leq M < 1$) und *E* für den Wert des Exponenten, so ergibt sich der Wert der dargestellten Zahl durch

$$(-1)^S \cdot (1 + M) \cdot 2^{E-bias}$$

Folgende Tabelle zeigt die genauen Werte / Längen für Single / Double Präzission:

Wert	Single	Double
Länge Mantisse	23 bit	52 bit
Länge Exponent	8 bit	11 bit
Bias	127	1023

5.1.2 Grenzen

Gleitpunktzahlen dieser Art haben diverse Nachteile:

- Es gibt nur *endlich viele* Gleitpunktzahlen
- Es gibt *keine beliebig großen / kleinen* Zahlen
- Es gibt keine *beliebig benachbarten* Zahlen
- Gleitpunktzahlen sind *unregelmäßig verteilt*
- Nach Rechenoperationen muss i.d.R. gerundet werden

Den *maximalen relativen Abstand* zweier Zahlen in einem Gleitpunktsystem nennt man **relative Maschinengenauigkeit** (auch Maschinenepsilon, `macheps` oder nur `eps`). Es gilt:

$$\left| \frac{x - rd(x)}{x} \right| \leq \texttt{eps} = \frac{1}{2}\beta^{1-t}$$

Der Teil rechts des Gleichheitszeichens ist *nicht* den Vorlesungsfolien entnommen. Der Parameter β bezeichnet die Basis der Gleitkommadarstellung (hier: 2), *t* ist die Mantissenlänge. Die Formel lässt sich allerdings anhand der in den Folien angegebenen Werte für IEEE 754 Single/Double precision verifizieren.

5.1.3 Gleitkomma-Arithmetik: Addition

Addition von Gleitkommazahlen verläuft wie folgt:

1. **Angleichen der Komma-Position** durch Rechts-Shift der Zahl mit dem *kleineren Exponenten*. Hier kann Genauigkeit verloren gehen (bei fester Mantissenlänge).
2. Eigentliche **Addition**. Hier können zwei Vorkommastellen resultieren.
3. **Normalisieren** des Ergebnisses durch Rechts-Shift, anpassen des Exponenten (möglicher Überlauf!).
4. **Runden** der Mantisse auf verfügbare Stellenzahl (Verlust von Genauigkeit).

5.1.4 Rundungsfehler

Für den relativen Rundungsfehler gilt:

$$\varepsilon(x) := \frac{x - rd(x)}{x} \Leftrightarrow rd(x) = x(1 - \epsilon(x))$$
$$|\varepsilon(x)| \leq \texttt{eps}$$

Sei $gl(\cdots)$ die dem Inhalt der Klammern entsprechende Gleitpunktoperation, dann gilt in IEEE-Arithmetik:

$$gl(x + y) = (x + y) \cdot (1 + \varepsilon_1)$$
$$gl(x - y) = (x - y) \cdot (1 + \varepsilon_2)$$
$$gl(x \times y) = (x \times y) \cdot (1 + \varepsilon_3)$$
$$gl(x/y) = (x/y) \cdot (1 + \varepsilon_1)$$

Interessant ist nun die Fortpflanzung dieser Rundungsfehler: Wie wirken sich Rundungsfehler in den Eingangsdaten x einer Berechnung $y = f(x)$ auf das Ergebnis aus?

Ein Maß dafür ist die so genannte **Kondition**. Diese bestimmt den Grad der Auswirkung der Rundungsfehler; "schlecht konditioniert" bedeutet, dass kleine Abweichungen in der Eingabe (Rundungsfehler) zu großen Änderungen in der Ausgabe führen (immer!).

Die für f spezifischen Verstärkungsfaktoren, die **Konditionszahlen**, hängen nur von f ab, nicht vom verwendeten Algorithmus / der verwendeten Arithmetik. Definiert ist die Konditionszahl einer eindimensionalen Funktion f wie folgt:

$$cond_f(x) = \left| \frac{x}{f(x)} \cdot \frac{\delta f(x)}{\delta x} \right|$$

Erhält man bei einem *gut konditionierten Problem* trotzdem Ergebnisse mit großen Rundungsfehlern, ist das Berechnungsverfahren **numerisch instabil**. Oft ist ein Berechnungsschritt enthalten, der zu einer großen Auslöschung führt. Dann allerdings ist es unter Umständen möglich, das Problem durch algebraische Umformungen (Ausklammern, Brüche erweitern, etc.) in ein *numerisch stabiles* Berechnungsverfahren umzuformen.

Definition: Werden die relativen Eingabefehler eines *gut konditionierten Problems* durch ein Berechnungsverfahren *nicht weiter vergrößert*, so heißt das Verfahren **numerisch stabil**.

Note: Besondere Vorsicht gilt bei

- Subtraktion von nahezu gleich großen Zahlen
- Berechnungen mit relativ großen Zwischenwerten, wobei das Ergebnis relativ klein ist.

5.2 Berechnung nichtlinearer Gleichgewichtslösungen

Betrachtet werden nichtlineare Anfangswertprobleme $\dot{x} = f(x)$ mit $x(0) = x_0$, gesucht wird die Gleichgewichtslösung x_s, sodass $x(t) \to x_s$ für $t \to \infty$.

5.2.1 Differentialgleichungslöser

Idee: Verwende ein Verfahren zur *numerischen Lösung von Differentialgleichungen* (siehe *Numerische Lösung nichtlinearer Zustandsdifferntialgleichungen*). Iteriere so lange, bis ein stationärer Zustand erreicht ist. Dazu muss x_s ein stabiler stationärer Punkt und der Anfangswert x_0 nahe genug bei x_s sein, da dieses Verfahren sonst nicht konvergiert.

5.2.2 Fixpunktiteration

Die Gleichung $0 = f(x)$ muss umgewandelt werden in eine *Fixpunktgleichung* $x = g(x) \Leftrightarrow 0 = f(x)$. Eine Möglichkeit (nicht immer die beste) ist $g(x) := f(x) + x$.

Die Fixpunktiterationsvorschrift ist dann gegeben wie folgt:

$$x_{k+1} := g(x_k),\ k = 0, 1, 2, \cdots$$

Konvergenzbedingung: Sei $x_s = g(x_s)$ (der Fixpunkt von g), der Anfangswert x_0 nahe genug bei x_s und die Eigenwerte von $\frac{\delta g}{\delta x}(x_s)$ im Einheitskreis (d.h. der Betrag dieses Wertes < 1). Dann konvergiert das Verfahren gegen x_s.

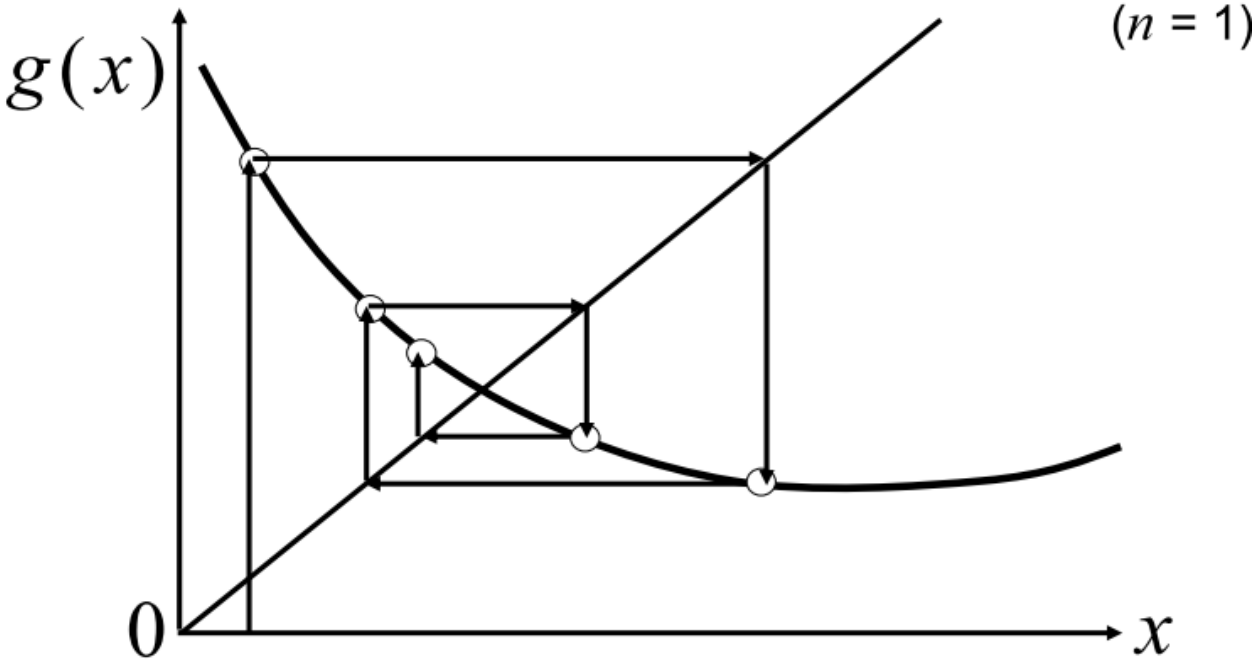

Abbildung 5.1: Eindimensionale Veranschaulichung der Fixpunktiteration.

Problem: Die Konvergenz der einfachen Fixpunktiteration ist nicht garantiert. Man kann die Konvergenzwahrscheinlichkeit erhöhen, indem man eine konstante und reguläre **Relaxationsmatrix** $A \in \mathbb{R}^n_n$ als Faktor hinzuzieht; diese

führt zu kleineren Eigenwerten und damit schnellerer "Kontraktion". Für diese Matrix gilt:

$$0 = f(x) \Leftrightarrow 0 = A \cdot f(x)$$
$$\Leftrightarrow x = x + A \cdot f(x) := g(x)$$

Die *optimale Relaxationsmatrix*, mit der die Fixpunktiteration fast immer möglich ist, ist die folgende Matrix A:

$$A = -\left[\frac{\delta f}{\delta x}(x_s)\right]^{-1} \text{ mit } f(x_s) = 0$$

5.2.3 Newton-Verfahren

Das Newton-Verfahren ist ein spezielles Fixpunktverfahren mit der Iterationsvorschrift

$$x_{k+1} := x - \left[\frac{\delta f}{\delta x}(x)\right]^{-1} \cdot f(x)$$

Kritisch ist die Wahl des Startvektors. Zu beachten ist, dass es sehr vorteilhaft ist, wenn dieser nahe bei der Lösung liegt; außerdem kann es Probleme geben, wenn der Startvektor nahe bei Polstellen oder stark schwankenden Funktionsbereichen liegt. Im Zweifelsfall hilf ausprobieren und beobachten der Ergebnisse. Im Idealfall verdoppelt sich in jeder Iteration die Anzahl der korrekten Nachkommastellen: Das Newton-Verfahren *konvergiert quadratisch* (lokal).

Ein Problem des Newton-Verfahrens ist, dass die Jacobi-Matrix berechnet und bei jedem Schritt neu ausgewertet werden muss. Es gibt verschiedene Ansätze, um das Verfahren durch unterschiedliche Berechnung der Jacobi-Matrix zu beschleunigen und die Lösungskomplexität zu verringern (vor allem interessant, wenn f nicht-linear, da dann der Berechnungsaufwand für jede Auswertung der Matrix groß ist):

- Untersuchung der Jacobi-Matrix auf Dünnbesetztheit, Approximation der von 0 verschiedenen Einträge durch *Vorwärtsdifferenzen-Approximation*
- Ersetzen der Jacobi-Matrix durch eine *konstante Matrix*, zum Beispiel an der Stelle des Anfangswertes. Dann aber nur noch *höchstens lineare Konvergenz*, bei gegebener Konvergenz jedoch *häufig schneller*
- *Quasi-Newton-Verfahren*: Schrittweises "Aktualisieren" der aktuellen Matrix nach speziellen Verfahren.

5.2.4 Vergleich der Verfahren

Die beiden letzten vorgestellten Verfahren (Newton und Fixpunktiteration) haben unterschiedliche Vorteile. Es muss, je nach Art des zu lösenden Problems, anhand derer Abgewogen werden.

Aspekt	Einfache Fixpunktiteration	Einfacher Newton
Lokale Konvergenz	-	++
Globale Konvergenz	0	-
Rechenaufwand pro Schritt	++	-
Implementierungsaufwand	++	-

5.3 Numerische Lösung nichtlinearer Zustandsdifferntialgleichungen

Die Grundidee der numerischen Integration ist die Approximation des Integrals durch eine Abschätzung - in der einfachsten Form ein Rechteck, das "unter die Kurve" gelegt wird. Bei den Integrationsverfahren unterscheidet man zwischen *Einschritt-*, *Mehrschritt-*, und Extrapolationsverfahren, wobei hier nur die Einschrittverfahren behandelt werden. Zusätzlich existieren *implizite* und *explizite* Verfahren.

Zur Auswahl und zur Bewertung von numerischen Integrationsverfahren betrachtet man im Allgemeinen

- Rechenaufwand (Berechnungseffizienz)
- Genauigkeit (Approximationsfehler)
- Eignung für steife Systeme (Stabilität)
- Implementierungsaufwand (nur bei Eigenimplementierung / Verwendung einer Bibliothek)

5.3.1 Einschrittverfahren

Gegeben sei $x_k \approx x(t_k)$, gesucht ist das Ergebnis des nächsten Iterationsschritts $x_{k+1} \approx x(t_{k+1})$.

Der allgemeine Ansatz für Einschrittverfahren ist

$$x_{k+1} = x_k + h \cdot \Phi\left(t_k, x_k, x_{k+1}, h; f\right)$$

Die einzelnen Verfahren unterscheiden sich dabei in der Wahl der Funktion Φ. Stets zu beachten ist, dass, wenn f an der Stelle x_k ausgewertet werden soll und x_k die Näherung von $x(t_k)$ darstellt, die Funktion an dem *Punkt* $(t_k; x_k)$ berechnet wird (es wird die Steigung dieses Punktes im *Richtungsfeld* berechnet. f ist in diesem Kontext quasi eine Funktion mit *zwei* Variablen).

Für Einschrittverfahren muss die folgende *Konsistenzbedingung* erfüllt sein:

$$\lim_{h \to 0} \Phi\left(t_k, x_k, x_{k+1}, h; f\right) = f(x_k) \tag{5.1}$$

5.3.2 Explizites Euler-Verfahren

Bei gegebenem Anfangswert $x(t_0) = x_0$ ist die Vorschrift des expliziten Euler-Verfahrens

$$x_{k+1} = x_k + h \cdot f_k \tag{5.2}$$

Dieses Verfahren ergibt sich durch die Wahl von $\Phi := f(x_k)$ aus der Vorschrift der Einschrittverfahren.

Das explizite Euler-Verfahren ist sehr schnell, allerdings auch ungenau; hohe Genauigkeit erfordert sehr kleine Schrittweite.

Geometrische Veranschaulichung: Im Punkt $(t_k; x_k)$ wird eine Gerade über den Zeitraum *h* in Richtung des Pfeils an dieser Stelle im Richtungsfeld gezeichnet. Dies wird wiederholt, bis die Endzeit erreicht ist.

5.3.3 Implizites Euler-Verfahren

Beim impliziten Euler-Verfahren wird der Wert von f an der Stelle der *nächsten* Näherung x_{k+1} berücksichtigt. Die dazugehörige Formel lässt sich nicht exakt, sondern nur *näherungsweise* lösen (siehe *Berechnung nichtlinearer Gleichgewichtslösungen*):

$$\begin{aligned} x_{k+1} &= x_k + h \cdot f(x_{k+1}) \\ \Rightarrow 0 &= x_{k+1} - x_k - h \cdot f(x_{k+1}) \end{aligned}$$

Die nächste Näherung x_{k+1} ergibt sich als die Ausgabe eines Verfahrens zur näherungsweisen Lösung von nichtlinearen Gleichungen angewendet auf die umgestellte Iterationsvorschrift (sodass auf einer Seite "0" steht - es wird die Nullstelle gesucht).

Dieses Verfahren ist ähnlich ungenau wie das explizite Euler-Verfahren, ist aber stabiler für *steife Differentialgleichungen* (siehe *Steife Systeme / Differentialgleichungen*).

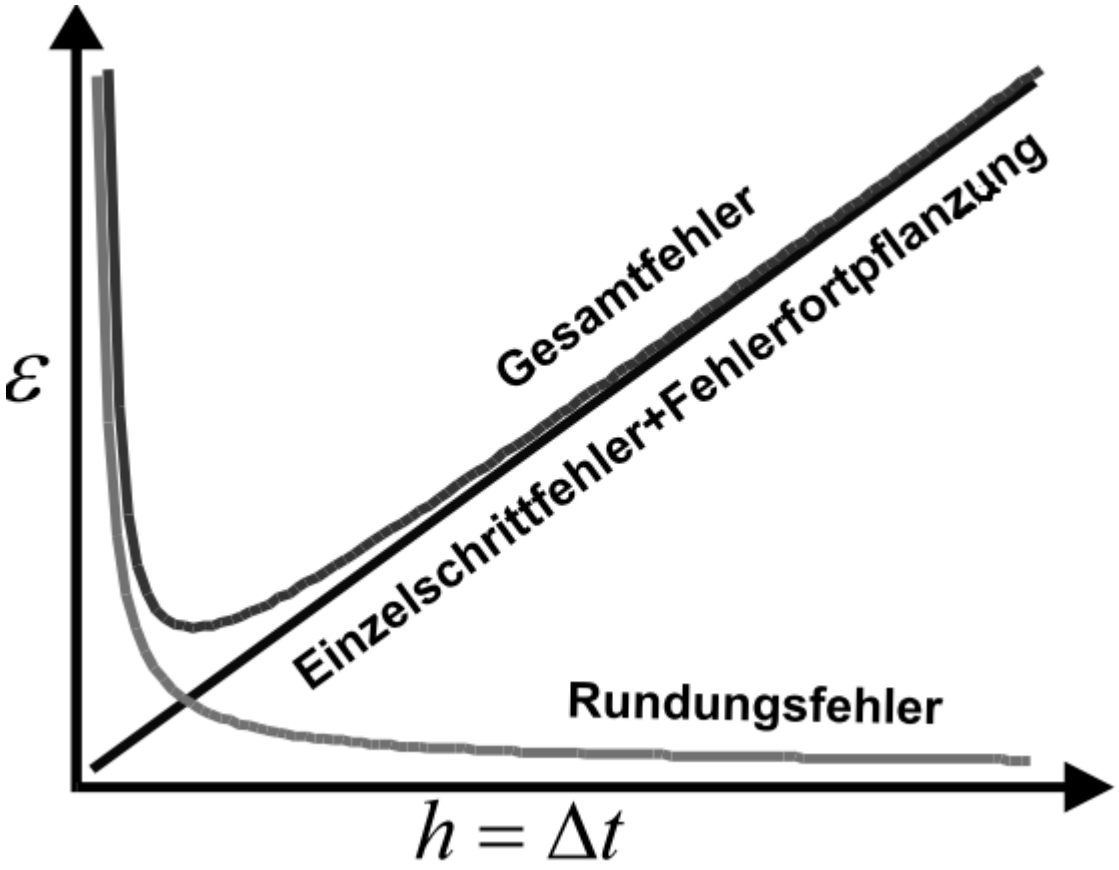

Abbildung 5.2: Die Entwicklung der Fehler beim expliziten Euler-Verfahren in Abhängigkeit der gewählten Schrittweitengröße.

5.3.4 Euler-Verfahren: Mehrdimensionaler Fall

Auch für Systeme von Differentialgleichungen lassen sich diese Verfahren anwenden; hier werden die einzelnen Komponenten für sich angenähert.

Sei $\tilde{x}(t_k)$ die aktuelle Näherungslösung am Zeitpunkt t_k, gesucht ist $\tilde{x}(t_{k+1})$.

Für $i = 1, 2, \cdots, dim(x)$ ist beim expliziten Euler-Verfahren

$$\tilde{x_i}(t_{k+1}) = \tilde{x_i}(t_k) + h_k \cdot f_i(\tilde{x}(t_k))$$

...und beim impliziten Euler-Verfahren:

$$\tilde{x_i}(t_{k+1}) = \tilde{x_i}(t_k) + h_k \cdot f_i(\tilde{x}(t_{k+1}))$$

5.3.5 Heun-Verfahren

Das Heun-Verfahren ist ein zweistufiges Einschritt-/ Prädikator-Korrektor-Verfahren. Anschaulich betrachtet berechnet man zuerst wie beim expliziten Euler-Schritt die Steigung am Ausgangspunkt und den Punkt, zu dem diese führt (das wäre schon x_{k+1} beim expliziten Euler), mittelt diese Steigung dann jedoch mit der am soeben errechneten "Temporärpunkt". Die so berechnete Steigung ist die, die benutzt wird, um vom Punkt (t_k, x_k) aus die nächste Näherung zu berechnen (siehe Grafik *Veranschaulichung des Heun-Verfahrens*).

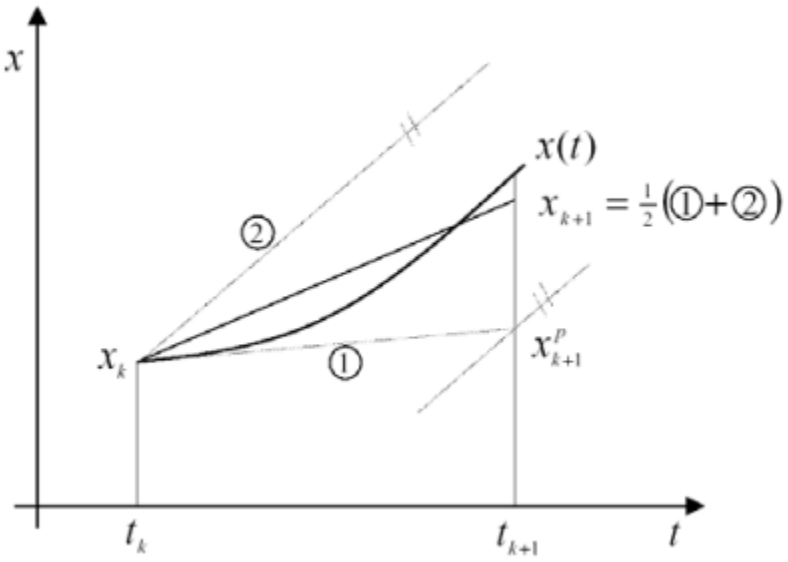

Abbildung 5.3: Veranschaulichung des Heun-Verfahrens

Zuerst erfolgt der *Prädikator-Schritt* (vgl. explizites Euler-Verfahren, Formel (5.2)):

$$x^p_{k+1} = x_k + h_k \cdot f(x_k)$$

Sodann erfolgt der *Korrektor-Schritt*, in dem die Mittelung stattfindet:

$$x_{k+1} = x_k + h_k \cdot \frac{1}{2} \left(f_k + f(x^p_{k+1}) \right)$$

Der Approximationsfehler liegt hier in $O(h^2)$, die Genauigkeit ist also höher als bei den Euler-Verfahren ($O(h^1)$).

5.3.6 Explizites 4-stufiges Runge-Kutta-Verfahren

Der allgemeine Ansatz für das explizite 4-stufige Runge-Kutta-Verfahren besitzt 10 unbekannte Koeffizienten, die so bestimmt werden müssen, dass die Konsistenzbedingung (5.1) erfüllt ist.

Eine Möglichkeit ist das "klassische" Runge-Kutta-Verfahren 4. Ordnung:

$$
\begin{aligned}
s_1 &= f(x_k) \\
s_2 &= f(x_k + \frac{h}{2} \cdot s_1) \\
s_3 &= f(x_k + \frac{h}{2} \cdot s_2) \\
s_4 &= f(x_k + h \cdot s_1) \\
x_{k+1} &= x_k + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4) \\
&\text{wobei } (s_1 + 2s_2 + 2s_3 + s_4) = \Phi
\end{aligned}
$$

Der Approximationsfehler liegt in $O(h^4)$. Erforderlich sind allerdings vier Auswertungen von f pro Iterationsschritt.

5.3.7 Integration von Differentialgleichungen mit Unstetigkeiten

Numerische Integrationsverfahren (siehe *Numerische Lösung nichtlinearer Zustandsdifferntialgleichungen*) fordern, dass die zu integrierende Funktion mindestens so oft *stetig differenzierbar* ist wie die Ordnung des Verfahrens (z.B.: Euler: einmal, klassisches Runge-Kutta: viermal). Unglücklicherweise sind Probleme aus der Realität oft auch unstetig, z.B. bei *unstetiger Steuerung* (z.B. Ventil auf / zu, Gangschaltung) oder einer *modellinhärenten Unstetigkeit* (z.B. Stoßvorgänge (hüpfender Ball, Roboterarmkollision), Reibung, oder bei strukturvariablen Systemen (Dimensionsänderungen)).

Im folgenden wird angenommen, dass f *abschnittsweise* mehrfach *stetig differenzierbar* ist; an den Übergängen jedoch womöglich nicht (unstetig, stetig aber nicht differenzierbar, usw.). Weiterhin werden die Übergänge durch *formulierbare Ereignisse* (Events) ausgelöst, die *zustandsabhängig* oder *zeitgesteuert* sein können.

Schaltfunktionen

Das Problem kann num mittels sogenannter *Schaltfunktionen* angegangen werden; die Schaltzeitpunkte werden als (einfache) Nullstellen *reellwertiger* Schaltfunktionen charakterisiert. Bei der numerischen Integration wird das Vorzeichen dieser Funktionen betrachtet; fand ein Wechsel statt, so muss der dazwischenliegende Schaltzeitpunkt mit gegebener Genauigkeit bestimmt und abschnittsweise integriert werden. Dies resultiert in einer Kombination aus numerischer Integration und (eindimensionaler) Nullstellensuche. Voraussetzung dabei ist, dass die Schrittweite klein genug ist, sodass kein *doppelter Vorzeichenwechsel* stattfindet.

Mathematisch formuliert:

Die Schaltzeitpunte $t_{s,i}, i = 1, \cdots, n_s$ werden durch Nullstellen n_q der rellwertigen Schaltfunktionen

$$q_l(x(t_{s,i}), ts_i) = 0, l \in \{1, \cdots, n_q\}$$

modelliert. Bsp.: $q_1 = d - x(t), q_2 = d + x(t)$.

Allgemeine Vorgehensweise

Um Probleme mit möglichen Unstetigkeiten zu behandeln, kann man dem folgenden Konzept folgen:

1. Modell (Differentialgleichung) auf mögliche unstetigkeiten in der Zustandsvariablen x, der rechten Seite oder den ersten Ableitungen (Jacobi-Matrix) der rechten Seite überprüfen.

2. Bestimmung der Schaltzeitpunkte (events). Dabei Fallunterscheidung über die Arte der Schaltpunkte:
   (a) Zeitgesteuert: Schaltzeitpunkte $t_{s,i}$ sind "a priori" (von Anfang an) bekannt
   (b) Zustandsabhängig: Schaltzeitpunkt als Nullstelle einer Schaltfunktion

$$q_k(x(t_{s,i}), t_{s,i}) = 0, k \in \{1, \cdots, n_q\}$$

3. Numerische Integration von einem Schaltpunkt zum nächsten. Halte dabei die Schaltpunkte so genau wie erforderlich ein.
4. Das Wiederaufsetzen nach einem gefundenen Schaltpunkt gestaltet sich wie ein normales, "neues" Anfangswertproblem. Der Anfangswert für *x* ergibt sich aus einer Zustandsübergangsbedingung, z.B. vom Typ

$$x(t_{s,i} + 0) = \xi\left(x(t_{s,i} - 0), t_{s,i}\right)$$

Wird einfach über Schaltpunkte integriert, d.h. diese nicht entsprechend wie oben skizziert behandelt, kann üblicherweise *hohe / gute Genauigkeit* nicht erreicht werden. Das Euler-Verfahren (z.B. ein einer "Game Physics Engine") merkt nichts; ein Integrator mit Schrittweitensteuerung stellt fest, dass sich ein Teil des stark ändert und verringert die Schrittweite; ist die minimal erlaubte Schrittweite erreicht, gibt der Integrator entweder auf (gewünschte Genauigkeit kann nicht erreicht werden) oder nimmt den Fehler in Kauf.

5.3.8 Integration steifer Differentialgleichungen

Steife Systeme (siehe *Steife Systeme / Differentialgleichungen*) lassen sich mit expliziten Verfahen nur mit *unnötig höher Rechenzeit* und *sehr kleinen Schrittweiten* ausreichend genau integrieren; es gilt für die Schrittweite *h*, dass sie die Bedingung

$$\forall \lambda_i . h < \frac{2}{|\lambda_i|}$$

einhalten muss. Da die Beträge der Eigenwerte bei steifen Problemen üblicherweise zumindest bei einem Eigenwert groß sind, ist die Schrittweite sehr klein.

Zur Berechnung sind *implizite Verfahren* (mit geeignetem "Stabilitätsgebiet" geeignet; auf explizite Verfahren sollte in diesem Falle verzichtet werden.

KAPITEL

SIX

TEILSCHRITTE EINER SIMULATIONSSTUDIE

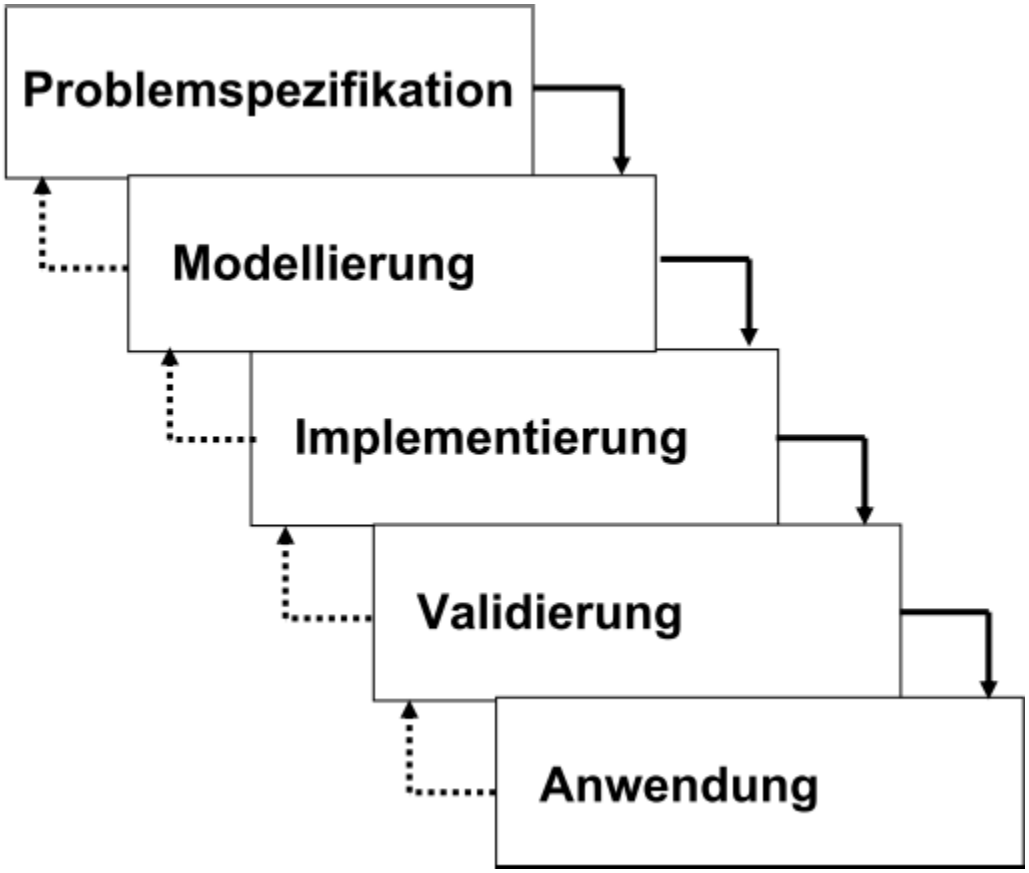

Abbildung 6.1: Teilschritte einer Simulationsstudie

Eine Simulationsstudie ist aufgeteilt in die Schritte Problemspezifikation, Modellierung, Implementierung, Validierung und Anwendung (siehe Abb. *Teilschritte einer Simulationsstudie*).

6.1 Problemspezifikation

In dieser Phase findet die

- Aufgabenformulierung
- Kriterienfestlegung
- Datenerhebung

statt. Dabei wird die prinzipielle Realisierbarkeit überprüft sowie die physikalischen und sonstigen Rahmenumstände des Modells untersucht; außerdem werden Zielsetzungen der Untersuchung festgelegt.

6.2 Modellierung

Im Zuge der Modellierung wird

- Die *Struktur* des Modells festgelegt
- Die *Modellgleichungen* aufgestellt
- *Vereinfachungen* am Modell vorgenommen

Bei der Modellierung werden Gesetzt der Domäne (z.B. physikalische, mechanische Gesetze) verarbeitet mit dem Ziel, *Differentialgleichungen erster Ordnung* zu erhalten. Dazu müssen nach der eigentlichen Modellierung ggf. noch Vereinfachungen vorgenommen werden.

Der Ausgangspunkt der Modellbildung ist die *Festlegung der Zustandsvariablen*. Diese

- sind zeitabhängige Größen
- legen die aktuelle Konfiguration des Systems exakt fest
- legen den zukünftigen Verlauf genau fest
- sind nicht redundant

6.3 Implementierung

Die Implementierung der Simulation besteht grob aus

- Auswahl oder Entwicklung des *Berechnungsverfahrens*
- *Programmierung* von Modell und Berechnungsverfahren
- *Visualisierung* von Berechnungsergebnissen
- Laufzeitoptimierung

6.4 Validierung

Die Validierung ist eine *systematische Plausibilitätsprüfung*, Untersuchung der Übereinstimmung von Modell, Simulation und Realität: Fehlersuche, Konsistenzprüfung, Daten-, Parameterabgleich.

Mögliche Fehler sind Modellierungsfehler (z.B. zu große Vereinfachungen), Approximationsfehler des iterativen Berechnungsverfahrens, Rundungsfehler, oder natürlich Programmier- und Implementierungsfehler. Bei der Plausibilitätsprüfung spielt die Visualisierung / Animation eine wichtige Rolle.

Berücksichtigt werden muss, dass das Simulationsmodell aus der *Implementierung* eines *Modells* und einem *Berechnungsverfahren* besteht; d.h. bei der Validierung müssen Implementierung, Modell und Berechnungsverfahren auf Fehler untersucht werden.

6.4.1 Implementierung

Hierbei kann zuerst die syntaktische Fehlerfreiheit, z.B. durch Debugging, festgestellt werden. Zusätzlich kann man für (einfache) Spezialfälle das "naturgetreue" Verhalten der Implementierung untersuchen. Mittels eines solchen Spezialsfalls lässt sich auch die numerische Korrektheit der Implementierung überprüfen.

6.4.2 Modell

Es muss untersucht werden, ob die *Modellannahmen* für die zu untersuchenden Fragestellungen zulässig und logisch konsistent sind (also z.B. nicht Starrkörpermodell für Materialermüdungen anwendbar). Desweiteren wird überprüft, ob das Modell detailliert genug ist und die grundlegende Struktur stimmt.

Zuletzt müssen auch die gewählten Modellparameter korrekt sein (diese lassen sich z.B. anhand von Messwerten "tunen").

6.4.3 Berechnungsverfahren

Ist das Berechnungsverfahren geeignet (z.B. nicht explizites Verfahren bei steifen Problemen)? Wie verhält es sich mit Approximationsfehlern (Schrittweitensteuerung vs. kostante Schrittweite etc.)? Wie groß ist der Einfluss von Rundungsfehlern (Untersuchung auf Auslöschung)?

6.5 Anwendung

Nimmt für Entwickler wenig Zeit in Anspruch. In Simulationsläufen können z.B. Parameter und die Struktur variiert werden sowie Vorraussagen und Optimierungen versucht werden.

KAPITEL

SEVEN

MODULARE MODELLBILDUNG

Definition Modul: *Elementare* Einheit eines Systems mit *definiertem Verhalten* und *Schnittstellen* zur Umgebung.

Schnittstellen sind *benannt*, ggf. *gerichtet* und *einheitsbehaftet*.

Mit Hilfe von vorgegebenen Modulbausteinen lassen sich für ein modular modelliertes Problem *automatisch Gleichungen generieren*. Dabei gilt

- Die Summe gekoppelter Flussvariablen (an einem Kopplungspunkt) ist 0
- Gekoppelte Potentialvariablen sind gleich
- Zu den benutzten Modulen sind üblicherweise weitere Gleichungen (Modulgleichungen) angegeben, die zusätzlich automatisch benutzt werden können (Bsp.: Grafik *Elemente für elektrische Schaltungen. Modulare Modellbildung*).

Bei der automatischen Gleichungsgenerierung entsteht eine *sehr große Anzahl an Gleichungen*, ein so genanntes **Differential-algebraisches System (DAE)**. Viele Koppelgleichungen sind leicht eliminierbar, nichtlineare Gesetze unter Umständen nicht. Selbst nach *vollständiger Elimination* ist das resultierende ODE-System (System gewöhnlicher Differentialgleichungen) noch sehr unhandlich.

Allgemein ist ein DAE-System

$$\dot{x}_d = f(x_d, x_a),\ x_d \text{ differentielle Variablen}$$
$$0 = h(x_d, x_a),\ x_a \text{ algebraische Variablen}$$

Mit $dim(f) = dim(x_d)$ sowie $dim(h) = dim(x_a)$.

Das Lösen von DAEs entspricht dem Lösen von Differentialgleichungen mit Nebenbedingungen.

Prinzipien für die automatische Modellgenerierung:

1. *Zerlegung* des Systems in *Module* mit Ein/Ausgangs-Schnittstellen
2. Zuordnung einer *Potential/Flussvariablen pro Schnittstelle*
3. Formulierung von *Gleichungen pro Modul* unter *ausschließlicher Benutzung* von Schnittstellen / Konstanten
4. *Kopplung* zweier Module über (kompatible) Schnittstellen induziert eine *Potential-* und eine *Flussgleichung*

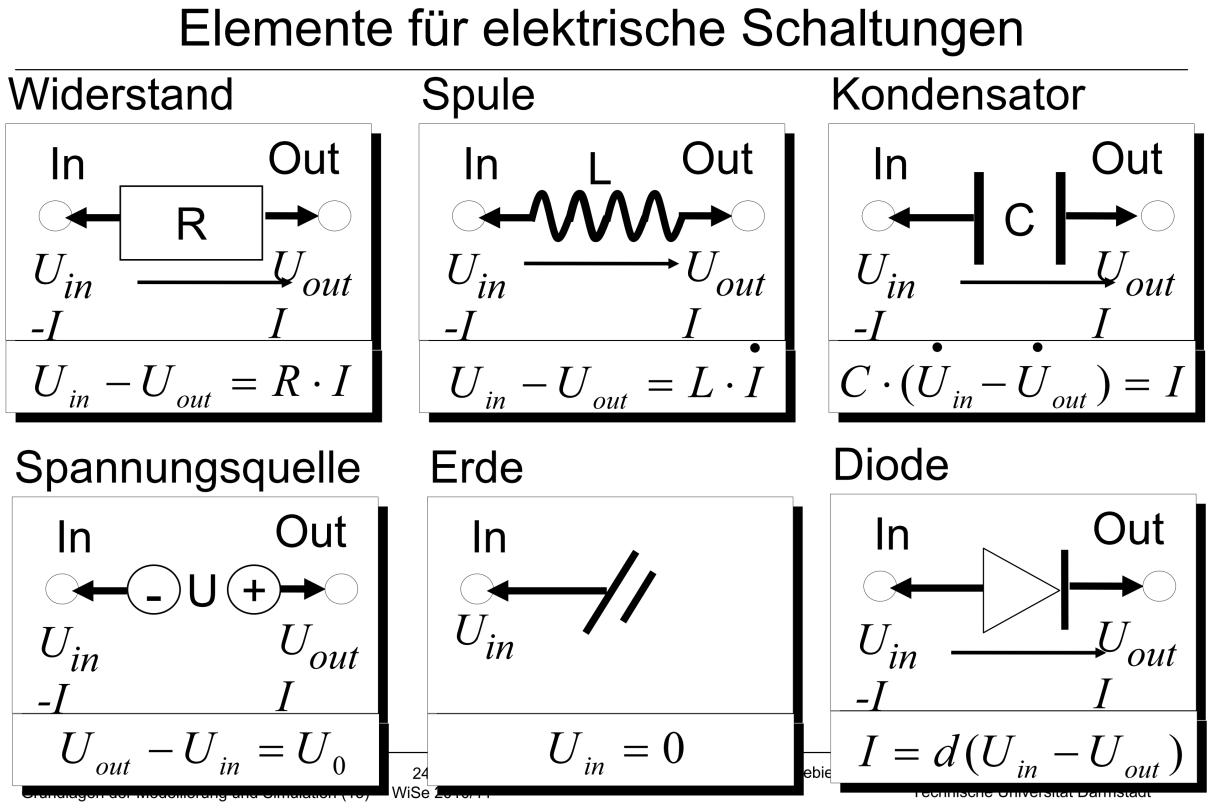

Abbildung 7.1: Elemente für elektrische Schaltungen. Modulare Modellbildung