Computational Engineering und Robotik

Zusammenfassung

Fabian Damken

6. März 2022

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhaltsverzeichnis

1 Einführung **5**
- 1.1 Beispiele . . . 5
- 1.2 Was ist ein Roboter? . . . 5
  - 1.2.1 Funktionale Anforderungen . . . 5
- 1.3 Was ist Robotik? . . . 5
- 1.4 Typischer Aufbau eines Roboterarms/-beins . . . 5
- 1.5 Begriffe . . . 5
  - 1.5.1 Simulation . . . 5
  - 1.5.2 Modell . . . 5
- 1.6 Aufgaben einer Simulation . . . 5

2 Mathematische Grundlagen und Notation **6**
- 2.1 Vektoren . . . 6
- 2.2 Matrizen . . . 6
- 2.3 Koordinatensysteme . . . 7
- 2.4 Positionsvektor eines Punktes . . . 7
- 2.5 Räumliche Anordnung eines Objektes . . . 8
- 2.6 Klassische Transformationsbeziehungen . . . 8
- 2.7 Homogene Transformationsbeziehungen . . . 8
- 2.8 Lösungsansätze für mathematische Modelle . . . 8
- 2.9 Numerik . . . 9
  - 2.9.1 Grundlegende Numerische Verfahren . . . 9
  - 2.9.2 Lösung nichtlinearer Gleichungsmodelle . . . 9
  - 2.9.3 Numerische Simulation . . . 11
  - 2.9.4 Numerische Simulation zeitkontinuierlicher Modelle . . . 11
- 2.10 Differentialgleichungen . . . 12
  - 2.10.1 Transformation auf System 1. Ordnung . . . 12
  - 2.10.2 Autonomisierung nichtautonomer DGL-Systeme . . . 13
  - 2.10.3 Richtungsfelder . . . 13
  - 2.10.4 Lösbarkeit . . . 13
  - 2.10.5 Gleichgewichtslösungen . . . 14
  - 2.10.6 Linearisierung um die Ruhelage . . . 14
  - 2.10.7 Lösung von $(\Delta x)' = A \cdot \Delta x$ . . . 15
  - 2.10.8 Stabilität von Differentialgleichungen . . . 15
  - 2.10.9 Zeitcharakteristik von Differentialgleichungen . . . 16
  - 2.10.10 Steife Differentialgleichung . . . 16
  - 2.10.11 Unstetige rechte Seite . . . 16

3 Modellung und Simulation mit nichtlinearen Gleichungsmodellen **17**
- 3.1 System und Modell: Notation . . . 17

3.2 Explizite vs. Implizite Gleichungsmodelle . . . 17
3.3 Aufbau von Roboterarmen/-beinen . . . 18
3.3.1 Koordinatensysteme . . . 18
3.4 Vorwärts-/Rückwärtsmodell . . . 18
3.5 Vorwärtskinematikmodell . . . 18
3.5.1 DH-Konvention . . . 19
3.5.2 Homogene Transformation in DH-Konvention . . . 19
3.6 Inverses Kinematikmodell . . . 20
3.6.1 Problematiken . . . 21
3.6.2 Berechnung . . . 21

4 Zeitkontinuierliche Modelle 22
4.1 Örtlich konzentrierter Systemzustand . . . 22
4.1.1 Beispiel: Linearer Schwinger . . . 22
4.2 Örtlich verteilter Systemzustand . . . 23
4.3 Beschreibung zeitkontinuierlicher Systeme . . . 23
4.3.1 Allgemeine Zustandsgleichung, Allgemeines Zustandsraummodell . . . 23
4.3.2 Lineare/Nichtlineare Systemdynamik . . . 24
4.4 Linearisierung um die Ruhelage . . . 24
4.5 Stabilität . . . 24
4.6 Regelung . . . 24
4.6.1 Steuerung und Regelung . . . 25
4.6.2 Regelung . . . 25
4.6.3 Lineares Feder-Masse-System . . . 26
4.6.4 PID-Bahnregelung eines Servomotors . . . 28

5 Numerische Simulation 30
5.1 Zahlendarstellung . . . 30
5.2 Rundungsfehler . . . 30
5.3 Fortpflanzung von Rundungsfehlern . . . 31
5.4 Kondition . . . 31
5.5 Numerische Stabilität . . . 32

6 Numerische Simulation zeitkontinuierlicher Modelle 33
6.1 Verschiedene Dynamikmodelltypen . . . 33
6.2 Numerische Integration . . . 33
6.2.1 Einschrittverfahren . . . 34
6.2.2 Explizites Euler-Verfahren . . . 34
6.2.3 Implizites Euler-Verfahren . . . 34
6.2.4 Heun-Verfahren . . . 34
6.2.5 Runge-Kutta-Verfahren 4. Ordnung . . . 35
6.2.6 Anmerkungen . . . 36
6.2.7 Schrittweitensteuerung . . . 36
6.3 Integration steifer Differentialgleichungen . . . 37
6.4 Integration von Zustands-Differentialgleichungen mit Unstetigkeiten . . . 37
6.4.1 Ursachen für Unstetigkeiten . . . 38
6.4.2 Schaltfunktionen . . . 38
6.4.3 Anmerkungen . . . 39

7 Teilschritte einer Simulationsstudie **40**
7.1 Problemspezifikation . . . 41
7.2 Modellierung . . . 41
7.2.1 Herleitung von Modellen . . . 41
7.2.2 Zustandsvariablen eines Modells . . . 42
7.2.3 Klassifikation der Zustandsänderungen . . . 43
7.3 Implementierung . . . 43
7.3.1 Klassifikation zeitkontinuierlicher Simulationswerkzeuge . . . 43
7.4 Anwendung . . . 43

8 Validierung **44**
8.1 Fehlerquellen . . . 44
8.2 Begriffe und Definitionen . . . 44
8.3 Vorgehensweise . . . 45
8.3.1 Validierung der Implementierung (D, L, V) . . . 45
8.3.2 Validierung des Modells (M) . . . 45
8.3.3 Validierung des Berechnungsverfahrens (D, L) . . . 45
8.3.4 Tests auf Plausibilität und Konsistenz . . . 45

9 Identifikation von Modellen **46**
9.1 Systemidentifikation . . . 46
9.1.1 Arten der und Modellbildung . . . 46
9.1.2 Hauptschritte der Systemidentifikation . . . 47
9.2 Parameteridentifikation . . . 47
9.2.1 Schema der Parameteridentifikation . . . 48
9.2.2 Gütekriterien . . . 48
9.2.3 Kalibrierung . . . 49

10 Physikalisch basierte Spiele **50**
10.1 Definition . . . 50
10.2 Game Loop . . . 50
10.3 Game Engine . . . 50
10.4 Physics & Collision Engine . . . 51
10.5 Modellierung eines Objektes . . . 52
10.6 Kontake, Kollisionen, Kräfte und Impulse . . . 52

11 Simulation autonomer Roboter **53**
11.1 Problemspezifikation . . . 53

12 Beispiele aus der Forschung **54**

1 Einführung

1.1 Beispiele

1.2 Was ist ein Roboter?

1.2.1 Funktionale Anforderungen

1.3 Was ist Robotik?

1.4 Typischer Aufbau eines Roboterarms/-beins

1.5 Begriffe

1.5.1 Simulation

1.5.2 Modell

1.6 Aufgaben einer Simulation

2 Mathematische Grundlagen und Notation

2.1 Vektoren

Vektor

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \in \mathbb{R}^3 = \mathbb{R}^{3\times 1} \quad \text{(Spaltenvektor)}$$

Nullvektor

$$0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Transponierter Vektor

$$p^T = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} \in \mathbb{R}^{1\times 3} \quad \text{(Zeilenvektor)}$$

(Euklidische) Norm eines Vektors

$$\|p\| = \sqrt{p_1^2 + p_2^2 + p_3^2}$$

Skalarprodukt zweier Vektoren $p, r \in \mathbb{R}^3$

$$\langle p, r \rangle = p^T p = p_1 r_1 + p_2 r_2 + p_3 r_3$$

Orthogonale Vektoren $p \perp r$

$$\langle p, r \rangle = p^T r = 0$$

Winkel Φ **zwischen zwei Vektoren** p, r

$$\cos(\Phi) = \frac{p^T r}{\|p\| \cdot \|r\|}$$

2.2 Matrizen

(3×3)**-Matrix**

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

Transponierte Matrix

$$R^T = \begin{bmatrix} r_{11} & r_{21} & r_{31} \\ r_{12} & r_{22} & r_{32} \\ r_{13} & r_{23} & r_{33} \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

Einheitsmatrix

$$I = E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

Nullmatrix

$$0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

Inverse Matrix R^{-1}

$$RR^{-1} = E = R^{-1}R$$

Orthonormale Matrizen Für orthonormale (orthogonale und normierte) Matrizen gilt

$$RR^T = E \implies R^T = R^{-1}$$

Außerdem gilt für orthonormale Matrizen:

$$1 = \det(E) = \det\left(RR^T\right) = \det(R)\det\left(R^T\right) = (\det(R))^2 \implies |\det(R)| = 1$$

Orthonormale Matrizen mit $\det(R) = +1$ heißen *Rechtssystem*, orthonormale Matrizen mit $\det(R) = -1$ heißen *Linkssystem*.

2.3 Koordinatensysteme

- Koordinatensystem dienen der Beschreibung von *Position* und *Orientierung* von Robotergelenken, Endeffektoren, Objekten in der Umgebung u.v.m..
- Oftmals werden *kartesische Rechtskoordinatensysteme* verwendet.
- Die Vektoren $e_x, e_y, e_z \in \mathbb{R}^3$, welche das Koordinatensystem aufspannen, sind zueinander orthogonale Einheitsvektoren, das heißt sie bilden eine Basis von $\mathbb{R}$.

2.4 Positionsvektor eines Punktes

- Die Position eines Punktes im Raum kann durch einen *Positionsvektor* angegeben werden.
- Im dreidimensionalen euklidischen Raum ist dies ein Vektor

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = p_x e_x + p_y e_y + p_z e_z$$

- **Notation:** Bei einem Punktvektor relativ zu einem Koordinatensystem S_a, wird dies mit ${}^a p$ notiert.

2.5 Räumliche Anordnung eines Objektes

- Zur Beschreibung der Lage eines Objektes wird ein *lokales Koordinatensystem* erstellt, welches fest mit dem Objekt verbunden ist.
- Die Lage dieses Koordinatensystems S_b wird mit ${}^a e_x, {}^a e_y, {}^a e_z$ bezüglich des äußeren Koordinatensystems S_a angegeben.
- So entstehen durch Verschiebung (Translation) und Verdrehung (Rotation) neue Koordinatensysteme.

2.6 Klassische Transformationsbeziehungen

- Eine *klassische Transformationsbeziehung* beim Wechsel von Punktkoordinaten bezüglich S_a zu S_b wir mit getrennter Translation und Rotation durchgeführt:

$$ {}^a p = {}^a r_b + {}^a R_b \, {}^b p $$

- ${}^a r_b$ beschreibt die Translation zwischen S_a und S_b.
- ${}^a R_b$ beschreibt die Rotation zwischen S_a und S_b.
- Die Koordinaten der Einheitsvektoren in S_b dargestellt in S_a entsprechen den Spalten der Rotationsmatrix ${}^a R_b$.

$$ {}^a R_b = \left[{}^a e_{xb} \mid {}^a e_{yb} \mid {}^a e_{zb} \right] $$

2.7 Homogene Transformationsbeziehungen

- Statt einer klassischen Transformation kann auch eine *homogene Transformation* durchgeführt werden.
- Diese Nutzt nur eine Matrix für die Translation und die Rotation.

$$ {}^a \hat{p} = \begin{bmatrix} {}^a p \\ \hline 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & {}^a R_b & & {}^a r_b \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \cdot \begin{bmatrix} {}^b p \\ \hline 1 \end{bmatrix} = {}^a T_b \cdot {}^b \hat{p} $$

- Die *homogene Transformationsmatrix* wird als ${}^a T_b$ bezeichnet.

2.8 Lösungsansätze für mathematische Modelle

Zur Lösung von mathematischen Modellen gibt mehrere Möglichkeiten:

Analytisch Das Problem mit exakt gelöst, es sind keine Vereinfachungen/Näherungen nötig. Dies ist nur in wenigen Fällen (effizient) möglich.

Heuristisch „Trial & Error“ gemäß einer bestimmten Strategie, vor allem nützlich bei Problemen der diskreten Optimierung.

Direkt-Numerisch Der numerische Algorithmus liefert exakte Lösungen (mglw. mit Rundungsfehlern). Dies ist ein klarer Algorithmus und eine Heuristik mehr, das Erreichen des Ziels ist stets sichergestellt.

Approximativ-Numerisch Iterative Näherungsverfahren für angenäherte Beziehungen (Gleichungen). Dies stellt den häufigsten Fall der Lösungsstrategien dar und ist der Hauptbestandteil der numerischen Simulation.

Analytische und Heuristische Verfahren werden im Vergleich zu den numerischen Verfahren selten eingesetzt, da die meisten Probleme analytisch zu schwer zu lösen sind und Heuristiken für die meisten Probleme nicht oder nur langsam konvergieren.

2.9 Numerik

2.9.1 Grundlegende Numerische Verfahren

2.9.2 Lösung nichtlinearer Gleichungsmodelle

Aufgabe Punkte $x_s \in \mathbb{R}^n$ finden, sodass

$$F(x_s) = 0 \iff \begin{aligned} F_1(x_1, \cdots, x_n) &= 0 \\ &\vdots \\ F_n(x_1, \cdots, x_n) &= 0 \end{aligned}$$

erfüllt ist.

Sämtliche Gleichungen können in diese Form überführt werden.

Fixpunktiteration

- Die Fixpunktiteration ist ein einfacher Algorithmus, um einen Fixpunkt einer Funktion zu finden.
- Somit muss die Gleichung $F(x) = 0$ in eine Fixpunktgleichung $x = g(x)$, bspw. $g(x) := F(x) + x$ umgebaut werden.
- Es muss außerdem ein Startwert x_0 gegeben sein, welcher eine Näherung der Lösung darstellt.
- Dann gilt für die Iteration:
$$x_{k+1} := g(x_k) \qquad k \in \mathbb{N}$$
- Das Verfahren terminiert, wenn es nahe genug bei der Lösung ist, die maximale Iterationsanzahl erreicht ist oder kein Fortschritt ersichtlich ist.
- Konvergenzbedingung: Sei $x_s = g(x_s)$, dann muss x_0 nahe genug bei x_s liegen und alle Eigenwerte von $\frac{\partial g}{\partial x}(x_s)$ müssen im Einheitskreis liegen (d.h. $\forall \lambda : |\lambda| < 1$).
Dann konvergiert $x_k \to x_s$.

Konvergenzhilfe: Relaxationsmatrix

- Da die Konvergenz nicht garantiert ist, kann eine *Relaxationsmatrix* $A \in \mathbb{R}^{n \times n}$ vorgeschaltet werden, mit deren Hilfe die Konvergenz verbessert werden kann:
$$g(x) := A \cdot F(x) + x$$

- Diese Relaxationsmatrix muss konstant und regulär sein.
- Als einfachster Ansatz kann eine Diagonalmatrix (oder sogar die Einheitsmatrix) verwendet werden.
- Eine optimale Relaxationsmatrix ergibt sich durch

$$A = -\left[\frac{\partial F}{\partial x}(x_s)\right]^{-1} \qquad F(x_s) = 0$$

- Durch die Wahl einer optimalen Relaxationsmatrix ist eine konvergente FPI immer möglich (sofern eine Lösung existiert).

Newton-Verfahren

- Das Newton-Verfahren ist ein Algorithmus, um eine Nullstelle einer Funktion zu finden.
- Das Newton-Verfahren löst somit eine Gleichung $F(x) = 0$.
- Hierzu muss ein Startwert x_0 gegeben sein, welcher eine Näherung der Lösung darstellt.
- Dann gilt für die Iteration:

$$x_{k+1} \coloneqq x_k + \Delta x_k \qquad \Delta x \coloneqq -\left[\frac{\partial F}{\partial x}(x_k)\right]^{-1} \cdot F(x_k)$$

  Die Berechnung von Δx glückt nur, wenn $\frac{\partial F}{\partial x}(x_k)$ regulär ist. Ansonsten muss die Gleichung

$$0 = F(x_k) + \frac{\partial F}{\partial x}(x_k) \cdot \Delta x_k$$

  gelöst werden.
- Das Verfahren terminiert, wenn es nahe genug bei der Lösung ist, die maximale Iterationsanzahl erreicht ist oder kein Fortschritt ersichtlich ist.
- Konvergenzbedingung: Sei x so, dass $F(x) = 0$ gilt, dann muss x_0 nahe genug bei x liegen.

Konvergenz: Schrittweitensteuerung

- Bei hoch-nichtlinearen Funktionen kann es passieren, dass über die Nullstele drüber iteriert wird.
- Durch eine geeignete Schrittweitensteuerung α kann dies verhindert werden (mit $0 < \alpha_{\text{min}} \leq \alpha_k \leq 1$):

$$x_{k+1} = x_k + \alpha_k \cdot \Delta x_k$$

- α sollte nur „so klein wie nötig“ und „so groß wie möglich“ sein, da nur für $\alpha \approx 1$ quadratische Konvergenz zu erwarten ist.
- Die Schrittweite kann beispielsweise durch Minimierung von

$$\varphi(\alpha) = \|F(x_k + \alpha \cdot \Delta x_k)\|_2^2 = \sum_{i=1}^{n} (F_i(x_k + \alpha \cdot \Delta x_k))$$

  bestimmt werden.

Berechnung der Jacobi-Matrix

- Die Berechnung der Jacobi-Matrix gestaltet sich oft schwierig.
- Außerdem: Ist ein Eintrag in der Jacobi-Matrix falsch, konvergiert das Newton-Verfahren nur schlecht oder gar nicht.
- Somit wird in der Praxis häufig *Vorwärtsdifferenzen-Approximation* verwendet.
- Ist $F(x)$ nichtlinear, so ver-n-facht sich der Berechnungsaufwand durch die VD-Approximation.
- Es gibt noch viele weitere Lösungsmethoden für die Jacobi-Matrix.
  - Untersuchung der Jacobi-Matrix auf *Dünnbesetztheit* und Überspringen der Einträge $x_{ij} \approx 0$. Berechnung der restlichen Einträge bspw. mit VD-Approximation.
  - Ersetzung der Jacobi-Matrix $\frac{\partial F}{\partial x}(x_k)$ mit einer *konstanten Matrix*, bspw. $\frac{\partial F}{\partial x}(x_0)$.
    * Das Verfahren ist damit nicht mehr quadratisch, sondern höchstens linear konvergent.
    * Sollte es dennoch konvergieren, ist es häufig schneller als ein „normales“ Newton-Verfahren.
  - *Schrittweise Aufaddierung („Updates“)* einer Approximation der Jacobi-Matrix („Quasi-Newton-Verfahren“).

Vergleich FPI $\leftrightarrow$ Newton

Fixpunktiteration

- Ausgeglichene globale Konvergenz, aber schlechte lokale Konvergenz.
- Geringer Rechenaufwand pro Iteration.
- Geringer Implementationsaufwand.

Newton-Verfahren

- Schlechte globale Konvergenz, aber sehr gut lokale Konvergenz.
- Hoher Rechenaufwand pro Iteration.
- Hoher Implementationsaufwand.

2.9.3 Numerische Simulation

Siehe Kapitel 5.

2.9.4 Numerische Simulation zeitkontinuierlicher Modelle

Siehe Kapitel 6

2.10 Differentialgleichungen

Allgemeiner Aufbau eines AWP 1. Ordnung:

$$\dot{x}(t) = f(x(t), u(t)) \qquad x(0) = x_0 \in \mathbb{R}^n$$

$f(x, u)$ Funktion der DGL.

x Zustand des Systems zu einem Zeitpunkt t.

u Stellgrößen zu einem Zeitpunkt t.

eine solche Differentialgleichung mit Anfangswert (Anfangswertproblem) kann auch als

$$x(t) = x_i + \int_0^t f(x(\tau), u(\tau))\, d\tau$$

geschrieben werden.

2.10.1 Transformation auf System 1. Ordnung

Jedes System von gewöhnlichen Differentialgleichungen höherer Ordnung kann in ein System 1. Ordnung transformiert werden.

Bei nichtlinearen System kann das System nicht in der Form $\dot{x} = Ax + b$ aufgeschrieben werden, sondern wird in der Form $\dot{x} = A(x)$ notiert, wobei A ein Vektor ist.

Beispiel: Schwingung einer Masse an einer Feder

- Ausgangsdifferentialgleichung 2. Ordnung:

$$\ddot{x} + \frac{k}{m}x = 0$$

- Umformung in eine explizite DGL:

$$\ddot{x} + \frac{k}{m}x = 0 \iff \ddot{x} = -\frac{k}{m}x$$

- Einführung weiterer Zustandsvariablen zur Transformation:

$$\hat{x}_1 := x$$
$$\hat{x}_2 := \dot{x}$$
$$\hat{x} := \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix}$$

- Transformation in ein DGL-System 1. Ordnung der Form $\dot{\hat{x}} = A\hat{x}$:

$$\begin{bmatrix} \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \cdot \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix}$$

2.10.2 Autonomisierung nichtautonomer DGL-Systeme

Jedes nicht-autonome System von DGL kann mit Einführung einer neuen Zeitvariablen autonomisiert werden.

Beispiel *Dieses Beispiel wandelt zeitgleich ein DGL-System 2. Ordnung ist ein System 1. Ordnung um.*

- Das folgende nichtautonome DGL-System 2. Ordnung soll in ein autonomes System 1. Ordnung umgewandelt werden:

$$\begin{aligned} 0 &= \frac{1}{2}\dot{x}_3 - \ddot{x}_2 + 5x_1 \\ 1 &= -1\ddot{x}_3 + x_1 + \dot{x}_2 - \frac{4}{5}t \\ 0 &= \dot{x}_3 - 3x_2 + \ddot{x}_1 + t \end{aligned}$$

- Zuerst werden die einzelnen Gleichungen in explizite DGL umgeformt:

$$\begin{aligned} \ddot{x}_2 &= \frac{1}{2}\dot{x}_3 + 5x_1 \\ \ddot{x}_3 &= \frac{1}{2}x_1 + \frac{1}{2}\dot{x}_2 - \frac{2}{5}t - \frac{1}{2} \\ \ddot{x}_1 &= 3x_2 - \dot{x}_3 - t \end{aligned}$$

- Als nächstes werden weitere Zustandsvariablen für die Transformation eingeführt:

$$\begin{aligned} &\hat{x}_1 \coloneqq x_1, \quad \hat{x}_2 \coloneqq x_2, \quad \hat{x}_3 \coloneqq x_3 \\ &\hat{x}_4 \coloneqq \dot{x}_1, \quad \hat{x}_5 \coloneqq \dot{x}_2, \quad \hat{x}_6 \coloneqq \dot{x}_3 \\ &\hat{x}_7 \coloneqq t \\ &\hat{x} \coloneqq \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \hat{x}_3 & \hat{x}_4 & \hat{x}_5 & \hat{x}_6 & \hat{x}_7 \end{bmatrix}^T \end{aligned} \qquad \text{(Autonomisierung)}$$

- Nun wird das System autonomisiert und in ein System 1. Ordnung der Form $\dot{\hat{x}} = A\hat{x} + b$ transformiert:

$$\begin{bmatrix} \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \\ \dot{\hat{x}}_3 \\ \dot{\hat{x}}_4 \\ \dot{\hat{x}}_5 \\ \dot{\hat{x}}_6 \\ \dot{\hat{x}}_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 0 & 0 & -1 & -1 \\ 5 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & -\frac{2}{5} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -\frac{1}{2} \\ 1 \end{bmatrix}$$

2.10.3 Richtungsfelder

2.10.4 Lösbarkeit

- Die allgemeine Lösung von $\dot{x} = f(x)$, $x \in \mathbb{R}^n$ hängt von n Integrationskonstanten ab, die z.B. durch n Anfangswerte festgelegt werden: $x(0) = x_0 \in \mathbb{R}^n$.

- Allgemein hat ein *autonomes* DGL-System *1. Ordnung* mit Anfangswert eine *eindeutige Lösung* $x(t), t > 0$, falls die Funktion $f(x)$ Lipschitz-Stetig ist (Satz von Picard-Lindelöff).
- Die Funktion $f(x)$i st Lipschitz-Stetig gdw.

  $$\exists L > 0 : \forall x_1, x_2 \in \mathbb{R}^n : (\|f(x_1) - f(x_2)\| \leq L\|x_1 - x_2\|$$

  gilt mit einer beliebigen Vektornorm $\|\cdot\|$.

2.10.5 Gleichgewichtslösungen

- Ein Zustand x_s heißt *stationärer Zustand* (oder *Ruhezustand*), wenn $x(t) \to x_s$ mit $t \to \infty$ gelten.
- Sei $u(t) = u_s$ gegeben.
- Falls x_s existiert, muss $\dot{x}(t) \to 0$ für $t \to \infty$ und somit $0 = f(x_s, u_s)$ gelten.
- Dies entspricht einem (i.A. nichtlinearen) System von n Gleichungen f_j zur Bestimmung von n Unbekannten $x_{s,i}$.
  - Die Frage nach der Existenz eines Ruhezustandes und dessen Eindeutigkeit lässt sich also auf die Frage nach der Lösbarkeit des Gleichungssystems zurück führen.
  - Für ein lineares Gleichungssystem $0 = Ax_s + Bu_s \iff -Bu_s = Ax_s$ heißt dies, das System ist eindeutig lösbar, wenn A quadratisch und regulär ist (d.h. $\det(A) \neq 0$).
- Jede Lösung $x_s \in \mathbb{R}^n$ heißt *Gleichgewichtslösung*.

2.10.6 Linearisierung um die Ruhelage

- Oftmals ist es nicht (effizient) möglich, ein nichtlineares DGL-System $\dot{x} = f(x, u) \in \mathbb{R}^n$ regeln.
- Das System kann um die Ruhelage linearisiert werden, sodass es einfacher zu regeln ist.
- Der Prozess der Linearisierung stützt sich auf Taylor-Entwicklungen um (x_s, u_s) bis zum linearen Term.
- Die Linearisierung läuft nun wie folgt ab:
  - Sei $\dot{x} =: (f_1, \cdots, f_n) =: f$.
  - Nun werden folgende Jacobi-Matrizen gebildet:

    $$A \coloneqq \frac{\partial f}{\partial x}(x_s, u_s)$$
    $$B \coloneqq \frac{\partial f}{\partial u}(x_s, u_s)$$

    Es wird somit einmal die n-Dimensionale Funktion in Richtung x und einmal in Richtung u abgeleitet.
  - Damit ergibt sich die folgende, linearisierte, Differentialgleichung (mit Δu als Regelparameter):

    $$\begin{aligned}(\Delta x)' &= A \cdot \Delta x + B \cdot \Delta u \\ &= \frac{\partial f}{\partial x}(x_s, u_s) \cdot \Delta x + \frac{\partial f}{\partial u}(x_s, u_s) \cdot \Delta u\end{aligned}$$

- Eine Linearisierung ist nicht möglich, wenn der Ruhezustand mit einer Sprung- oder Knickstelle (oder einer anderen Ausnahmestelle) zusammen fällt oder in der engeren Umgebung liegt, da die Funktion in diesen Punkten nicht differenzierbar ist.

2.10.7 Lösung von $(\Delta x)' = A \cdot \Delta x$

- Eine DGL $(\Delta x)' = A \cdot \Delta x$ lässt sich allgemein über die Eigenwerte λ_i und den dazugehörigen Eigenvektoren v_i der Matrix A lösen.
- Die Lösung lautet dann $\Delta x = \sum_i x_i$ für alle i mit unterschiedlichen (wobei für $z \in \mathbb{C}$ gilt $z = \bar{z}$) λ_i und folgenden x_i:
  - Falls $\lambda_i \in \mathbb{R}$:
    $$x_i = c_i \cdot e^{\lambda_i t}$$
  - Falls $\lambda_i \in \mathbb{C} \setminus \mathbb{R}$
    $$\begin{aligned} x_i &= c_i \cdot l_{i,\mathrm{Re}}(t) + i \cdot d_i \cdot l_{i,\mathrm{Im}}(t) \\ l_{i,\mathrm{Re}}(t) &\coloneqq e^{-t}(\mathrm{Re}(v_i) \cdot \cos(t) - \mathrm{Im}(v_i) \cdot \sin(t)) \\ l_{i,\mathrm{Re}}(t) &\coloneqq e^{-t}(\mathrm{Re}(v_i) \cdot \sin(t) - \mathrm{Im}(v_i) \cdot \cos(t)) \end{aligned}$$
- Die Koeffizienten c_i, d_i ergeben sich aus den Anfangswerten.

2.10.8 Stabilität von Differentialgleichungen

- Die Eigenwerte λ_i, $i = 1, \cdots, n$ bestimmen die *Stabilität* des Systems und könne
  - reell,
  - konjugiert komplex,
  - einfach oder
  - mehrfach sein,
- Für die Stabilität ist das Vorzeichen des reellen Teils $\mathrm{Re}(\lambda_i)$ entscheidend:
  - Bei *nur negativen reellen Eigenwerten* unterliegt das System einer aperiodischen Dämpfung und ist somit *stabil*.
  - Bei *mindestens einem reellen echt-positivem Eigenwert* wächst die Eigenbewegung mit $t \to \infty$ und das System ist somit *instabil*.
  - Bei *nur komplexen Eigenwerten mit* $\mathrm{Re}(\lambda_i) < 0$ liegt eine gedämpfte Oszillation vor, das System ist somit *stabil*.
  - Bei *mindestens einem komplexen Eigenwert mit* $\mathrm{Re}(\lambda_i s)$ liegt eine ungedämpfte Oszillation vor, das System ist somit *instabil*.
- Nach A. M. Ljapunov gilt:
  - Das System ist stabil, wenn alle Eigenwerte negative Realanteile aufweisen.
  - Das System ist instabil, wenn mindestens ein Eigenwert einen positiven Realanteil aufweist.
  - Es kann keine Aussage getroffen werden, wenn mindestens ein Eigenwert einen Realanteil Null aufweist und alle übrigen Eigenwerte einen negativen Realanteil aufweisen.

2.10.9 Zeitcharakteristik von Differentialgleichungen

Die Zeitcharakteristik eines linearen oder linearisierten Differentialgleichungssystems $(\Delta x)' = A \cdot \Delta x$ kann durch die Eigenwerte λ_i der Matrix A bestimmt werden. Dabei wird jedem Eigenwert λ_i eine Zeitcharakteristik T_i wird folgt zugewiesen:

$$T_i = \begin{cases} \frac{1}{|\lambda_i|} & \lambda_i \in \mathbb{R} \\ \frac{2\pi}{|\lambda_i|} & \lambda_i \in \mathbb{C} \wedge \mathrm{Re}(\lambda_i) = 0 \\ \min\left\{ \frac{1}{|\mathrm{Re}(\lambda_i)|}, \frac{2\pi}{|\mathrm{Im}(\lambda_i)|} \right\} & \lambda_i \in \mathbb{C} \wedge \mathrm{Re}(\lambda_i) \neq 0 \wedge \mathrm{Re}(\lambda_i) \neq 0 \end{cases}$$

Dann werden die minimalen und maximalen Zeitcharakteristika folgendermaßen zugewiesen:

$$T_{\min} = \min_i(T_i)$$
$$T_{\max} = \max_i(T_i)$$

2.10.10 Steife Differentialgleichung

- Eine stabile Differentialgleichung mit *sehr unterschiedlichen Zeitcharakteristika* wird *steife Differentialgleichung* genannt.
- Als Maß der Steifigkeit wird das Verhältnis der minimalen und maximalen Zeitkonstanten $\frac{T_{\max}}{T_{\min}}$ angesehen.
- Numerische Lösungen steifer Differentialgleichungen mit explizitem Integrationsverfahren benötigen hohe Rechenzeiten proportional zu $\frac{T_{\max}}{T_{\min}}$.
  Mit impliziten Verfahren kann dieses Problem gelöst werden.
- Ein solch steifes Verhalten ist zu erwarten, wenn für $\dot{x} = f(x, u)$ die Jacobi-Matrix $\frac{\partial f}{\partial x}(x, u)$ Eigenwerte λ_i mit Realanteil $\mathrm{Re}(\lambda) \ll 0$ sind.

2.10.11 Unstetige rechte Seite

- Ist die rechte Seite einer Differentialgleichung $f(x, u)$ unstetig, so kann dies zu Problem bei der numerischen Integration führen.
- Mit *Schaltpunkten* und *abschnittsweiser* Berechnung kann dies gelöst werden.
- Für weitere Informationen siehe Abschnitt 6.4.

3 Modellung und Simulation mit nichtlinearen Gleichungsmodellen

3.1 System und Modell: Notation

u Steuergrößen, kontrollierbar (Systemeingang)

z Störgrößen, nicht kontrollierbar (Systemeinang)

p Systemparameter, konstant während eines Systemlaufes

x Systemzustand, vollständige Charakterisierung des Systems

y Systemausgang

In diesem Kapitel wird angenommen, dass $z = 0$ und $y = x$ gilt.

3.2 Explizite vs. Implizite Gleichungsmodelle

Explizite Gleichungsmodelle Explizite Gleichungsmodelle sind zu der interessanten Variable x aufgelöst und können meist schnell berechnet werden.

$$x = F(u) \iff \begin{bmatrix} x_1 \\ \vdots \\ x_{n_x} \end{bmatrix} = \begin{bmatrix} F_1(u_1, \cdots, u_{n_u}) \\ \vdots \\ F_{n_x}(u_1, \cdots, u_{n_u}) \end{bmatrix}$$

wobei $F : \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$.

Implizite Gleichungsmodelle Implizite Gleichungsmodelle sind nicht zu der interessanten Variable x aufgelöst und erfordern meist mehr Rechenaufwand.

$$0 = F(x, u) \iff \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} F_1(x_1, \cdots, x_{n_x}, u_1, \cdots, u_{n_u}) \\ \vdots \\ F_{n_x}(x_1, \cdots, x_{n_x}, u_1, \cdots, u_{n_u}) \end{bmatrix}$$

wobei $F : \mathbb{R}^{n_x \times n_u} \to \mathbb{R}^{n_x}$.

3.3 Aufbau von Roboterarmen/-beinen

- Ein Roboterarm/-bein ist eine offene und starre kinematische Kette mit
  - Einer Menge an starren Körpern, den *links*, die
  - durch Gelenke, den *joints*, in einer Kette verbunden sind.
- Überlicherweise haben die elementaren Gelenke nur einen Bewegungsfreiheitsgrad:
  - Drehgelenke, welche in einer Achse rotieren können und
  - Schubgelenke, welche ihre Länge ändern können
- Die Gelenkstellung des i-ten Gelenks wird angegeben als:

$$q_i = \begin{cases} \text{Winkel } \Theta_i & \text{bei Drehgelenken} \\ \text{Strecke} d_i & \text{bei Schubgelenken} \end{cases}$$

3.3.1 Koordinatensysteme

- Zur Abbildung der Positionen der Gelenke und des Endeffektors wird an jedem Gelenk ein Koordinatensystem befestigt.
- Mit diesen $n+1$ Koordinatensystemen kann die Vorwärtskinematik (die Hinrichtung) sehr einfach berechnet werden.
- Siehe hierzu 3.5.

3.4 Vorwärts-/Rückwärtsmodell

- Das *Vorwärtsmodell* eines Systems beschreibt, wie aus gegebenen Gelenkstellungen die Position eines Endeffektors berechnet werden kann.
- Das *Rückwärtsmodell* oder auch *inverses Modell* eines Systems beschreibt, wie aus gegebener Position eines Endeffektors die Gelenkstellungen berechnet werden können.

3.5 Vorwärtskinematikmodell

- Sei n die Anzahl der Gelenke der kinematischen Kette. Dann ist S_0 das Weltkoordinatensystem und S_n das Koordinatensystem des Endeffektors.
- Das *Vorwärtskinematikmodell* eines Roboters ordnet den gegebenen Gelenkstellungen $q_1, \cdots, q_n$ eine Rotation ${}^{n-1}R_n$ und Position ${}^{n-1}r_n$ des Endeffektors zu.
- Da an jedem Gelenk ein Koordinatensystem befestigt ist, kann die Vorwärtskinematik einfach durch Kombination der lokalen homogenen Transformationsmatrizen gebildet werden:

$${}^0T_1 \times \cdots \times {}^{n-1}T_n = \left[\begin{array}{ccc|c} & {}^0R_n & & {}^0R_n \\ 0 & 0 & 0 & 1 \end{array}\right]$$

  Dann beschreibt ${}^0R_n$ die Rotation und ${}^0r_n$ die Lage des Endeffektors bezüglich des Weltkoordinatensystems.

- Eine lokale Transformationsmatrix ${}^{i}T_{i+1}$ beschreibt die Lage des Koordinatensystems S_{i+1} in Bezug auf das Koordinatensystem S_i und stellt eine homogene Transformationsmatrix (siehe 2.7) dar.

3.5.1 DH-Konvention

Die Denavit-Hartenberg (DH) Konvention ist eine Konvention zur Angabe der Freiheitsgrade eines Gelenkes. Dazu haben die angebrachten Koordinatensysteme folgende Eigenschaften zu erfüllen:

- Die Koordinatensysteme liegen in den Bewegungsachsen.
- Die z_{i-1}-Achse liegt entlang der Bewegungsachse des Gelenks.
- Die x_i-Achse steht senkrecht zur z_{i-1}- und z_i-Achse, zeigt von ihr weg und hat einen Schittpunkt mit ihr.
- Die y_i-Achse muss mit z_i- und x_i-Achse ein Rechtskoordinatensystem bilden.

Sind alle diese Anforderungen erfüllt, so werden nur 4 Parameter benötigt, um die gesamte Bewegung zu beschreiben:

Bedeutung	Symbol	Definition	Wert
Gelenkwinkel	Θ_i	Winkel zwischen x_{i-1}- und x_i-Achse um die z_{i-1}-Achse.	Variabel, wenn Drehgelenk
Verschiebung	d_i	Entfernung des Ursprungs von S_{i-1} zur x_i-Achse entlang der z_{i-1}-Achse.	Variablen, wenn Schubgelenk
Länge	a_i	Entfernung zwischen der z_{i-1}- und z_i-Achse entlang der x_i-Achse.	Konstant
Verdrehung	α_i	Winkel zwischen z_{i-1}- und z_i-Achse um die x_i-Achse.	Konstant

Tabelle 3.1: DH-Parameter Beschreibung

3.5.2 Homogene Transformation in DH-Konvention

Die (homogene) Transformation ${}^{i-1}T_i$ zwischen zwei benachbarten Koordinatensystemen S_{i-1} und S_i mit DH-Konvention ergibt sich durch:

$$
\begin{aligned}
{}^{i-1}T_i &= \text{Rot}(z;\Theta_i)\cdot\text{Trans}(0,0,d_i)\cdot\text{Trans}(a_i,0,0)\cdot\text{Rot}(x;\alpha_i) \\
&= \begin{bmatrix} \cos(\Theta_i) & -\sin(\Theta_i) & 0 & 0 \\ \sin(\Theta_i) & \cos(\Theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos(\Theta_i) & -\sin(\Theta_i)\cos(\alpha_i) & \sin(\Theta_i)\sin(\alpha_i) & a_i\cos(\Theta_i) \\ \sin(\Theta_i) & \cos(\Theta_i)\cos(\alpha_i) & -\cos(\Theta_i)\sin(\alpha_i) & a_i\sin(\Theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

3.6 Inverses Kinematikmodell

- Sei n die Anzahl der Gelenke der kinematischen Kette. Dann ist S_0 das Weltkoordinatensystem und S_n das Koordinatensystem des Endeffektors.
- Das *inverse Kinematikmodell* eines Roboters ordnet einer gegebenen Endeffektorrotation ${}^{n-1}R_n$ und Endeffektorposition ${}^{n-1}r_n$ gewisse Gelenkstellungen $q_1, \cdots, q_n$, welche im Vorwärtskinematikmodell zu der gewollten Rotation/Position führen.
- Folgende Fragen tun sich für das inverse Modell auf:
  1. Existiert eine Lösung?
  2. Ist die Lösung eindeutig oder mehrdeutig?
  3. Wie kann die Lösung berechnet werden?
- Mit dem impliziten Gleichungsmodell

$$0 = \begin{bmatrix} {}^0\tilde{r}_{n,x} \\ {}^0\tilde{r}_{n,y} \\ {}^0\tilde{r}_{n,z} \\ {}^0\tilde{R}_{n,1,1} \\ \vdots \\ {}^0\tilde{R}_{n,3,3} \end{bmatrix} - \begin{bmatrix} {}^0r_{n,x}(q_1, \cdots, q_n) \\ {}^0r_{n,y}(q_1, \cdots, q_n) \\ {}^0r_{n,z}(q_1, \cdots, q_n) \\ {}^0R_{n,1,1}(q_1, \cdots, q_n) \\ {}^0\vdots \; R_{n,3,3}(q_1, \cdots, q_n) \end{bmatrix}$$

  könnte die Gelenkposition berechnet werden, allerdings ist die Berechnung sehr aufwändig, da das System 12 Unbekannte hat.
- Die 3×3-Rotationsmatrix hat 9 Matrixelemente, kann aufgrund der orthonormalität aber durch maximal 3 unabhängige Winkel α, β, γ beschrieben werden.
- Es gibt hier viele Möglichkeiten, die Winkel festzulegen, z.B. durch Euler-Winkel, Kardan-Winkel, ...
- Hier werden sogenannte Z-Y-Z-Winkel verwendet, bei denen die Winkel α, β, γ Rotationen um Z-Y-Z-Achsen beschreiben.
- Damit kann das Gleichungsmodell wie folgt dedupliziert und vereinfacht werden:

$$0 = \begin{bmatrix} {}^0\tilde{r}_{n,x} \\ {}^0\tilde{r}_{n,y} \\ {}^0\tilde{r}_{n,z} \\ \tilde{\alpha} \\ \tilde{\beta} \\ \tilde{\gamma} \end{bmatrix} - \begin{bmatrix} {}^0r_{n,x}(q_1, \cdots, q_n) \\ {}^0r_{n,y}(q_1, \cdots, q_n) \\ {}^0r_{n,z}(q_1, \cdots, q_n) \\ \tilde{\alpha}(q_1, \cdots, q_n) \\ \tilde{\beta}(q_1, \cdots, q_n) \\ \tilde{\gamma}(q_1, \cdots, q_n) \end{bmatrix}$$

- Die Lösung kann nun analytisch oder numerisch (z.B. mit dem Newton-Verfahren) berechnet/approximiert werden.

3.6.1 Problematiken

- Das Problem ist durch das vorgestellte Verfahren noch nicht gelöst, da nicht alle Gelenkstellungen möglich sind (ein Arm kann nicht durch den Roboter greifen) und es viele Lösungen geben kann.
  - Eine Lösung existiert somit nur, wenn die Position und Orientierung im *Arbeitsraum* des Roboters sind.
  - Als Faustregel für die Lösbarkeit gilt bei einem INVKIN-Modell mit 6 nichtlinearen Gleichungen für n unbekannte q_i:
    * Bei $n < 6$ gibt es im Allgemeinen keine Lösung, da es mehr weniger Variablen als Gleichungen gibt.
    * Bei $n = 6$ gibt es im Allgemeinen eine Lösung, da es genau so viele Variablen wie Gleichungen gibt.
    * Bei $n > 6$ gibt es im Allgemeinen viele Lösungen, da es mehr Variablen als Gleichungen gibt.

3.6.2 Berechnung

Die Gleichung kann auf unterschiedliche Arten berechnet werden:

- Explizite *analytische* Lösung mit einer Lösungsformel.
- Iterative *numerische* Verfahren wie das Newton-Verfahren.
- Kombinationen aus den obigen Verfahren.
- Spezialverfahren unter Ausnutzung der Struktur des Vorwärtskinematikmodells.

Für alle Arten der Berechnung gilt die Anforderung, dass die Lösung in Echtzeit berechnet werden muss und dass alle Lösungen berechnet werden können müssen.

4 Zeitkontinuierliche Modelle

4.1 Örtlich konzentrierter Systemzustand

- Systeme mit *örtlich konzentrierten* Zuständen werden durch *gewöhnliche* Differentialgleichungen beschrieben.
- Beispiele:
  - Zeitlicher Verlauf der Schwingung einer Masse an einer Feder
  - Zeitlicher Verlauf von Strom und Spannung in einer elektrischen Schaltung
  - u.v.m.

4.1.1 Beispiel: Linearer Schwinger

Abbildung 4.1 zeigt einen linearen Schwinger mit folgenden Attributen:

m Masse

M Motor (Antriebs-/Bremskraft $F_M(t)$ in x-Richtung)

k Federkonstante

d Reibungskonstante

Aus diesem Blockschaltbild lässt sich direkt die entsprechende Differentialgleichung ablesen:

$$m\ddot{x} = F_M(t) - d\dot{x} - kx \qquad \text{(Grundform)}$$

$$\iff \ddot{x} = \frac{F_M(t)}{m} - \frac{d}{m}\dot{x} - \frac{k}{m}x \qquad \text{(Normalisiert)}$$

$$\iff F_M(t) = m\ddot{x} + d\dot{x} + kx \qquad \text{(Motorkraft)}$$

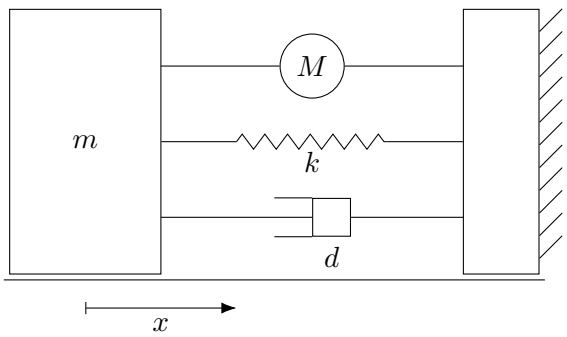

Abbildung 4.1: Linearer Schwinger: Blockschaltbild

Mit Hilfe der letzten, zur Motorkraft aufgelösten, Formel, kann die Bewegungsdifferentialgleichung auch auf mehrere Achsen (unter Vernachlässigung der Kopplungseffekte) erweitert werden:

$$F(t) = \underbrace{\begin{bmatrix} m_1 & 0 & \ddots \\ 0 & \ddots & 0 \\ \ddots & 0 & m_n \end{bmatrix}}_{M} \ddot{q} + \underbrace{\begin{bmatrix} d_1 & 0 & \ddots \\ 0 & \ddots & 0 \\ \ddots & 0 & d_n \end{bmatrix}}_{D} \dot{q} + \underbrace{\begin{bmatrix} k_1 & 0 & \ddots \\ 0 & \ddots & 0 \\ \ddots & 0 & k_n \end{bmatrix}}_{K} q$$

Dabei entsprechen $\ddot{q} \overset{\text{def}}{=} \ddot{x}$, $\dot{q} \overset{\text{def}}{=} \dot{x}$ und $q \overset{\text{def}}{=} x$ und $M \overset{\text{def}}{=} m$, $D \overset{\text{def}}{=} d$ und $K \overset{\text{def}}{=} k$ und $F(t) \overset{\text{def}}{=} F_M(t)$ in mehreren Achsen.

4.2 Örtlich verteilter Systemzustand

Örtlich verteilte Systemzustände werden durch *partielle* Differentialgleichungen beschrieben, die mehrere unabhängige Variablen haben.

Beispiele:

- Strömungsdynamik
- Elektromagnetische Felder
- u.v.m.

Diese Art von Systemen wird nicht weiter behandelt.

4.3 Beschreibung zeitkontinuierlicher Systeme

4.3.1 Allgemeine Zustandsgleichung, Allgemeines Zustandsraummodell

Allgemein werden die Systeme mit gewöhnlichen Differentialgleichungssystemen 1. Ordnung beschrieben:

$$\dot{x}(t) = \frac{dx(t)}{dt} = \begin{bmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} = \begin{bmatrix} f_1(x(t), u(t), t) \\ \vdots \\ f_n(x(t), u(t), t) \end{bmatrix} = f(x(t), u(t), t)$$

$$y = g(x, u, t) = \begin{bmatrix} g_1(x, u, t) \\ \vdots g_n(x, u, t) \end{bmatrix}$$

Mit den Zustandsgrößen x, den Steuervariablen (Stellgrößen) u und den Ausgangsgrößen y.

- Die große Aufgabe bei der Erstellung eines Modells ist es, die Funktion f zu finden und die Steuervariablen u mit geeigneten Werten zu belegen.
- Dabei wird die Modellgleichung f meistens aus physikalischen, mechanischen, ... Gesetzen hergeleitet.
- Steuervariablen müssen natürlich nicht immer explizit auftreten.

Zeitcharaktersitik

Siehe 2.10.9 für die Definition der hier verwendeten Variablen.

Aus den minimalen/maximalen Zeitcharakteristika $T_{\min}/T_{\max}$ ist die Simulationsdauer sinnvoll abschätzbar (als Faustregel):

- Stabiles System: $t_f = 5T_{\max}$
  (Bis zum erreichen der Gleichgewichtslage)
- Instabiles System: x_t sodass $x(t_f) \geq M$
  (Danach nicht mehr so interessant)

Daraus lässt sich auch eine geeignete Diskretisierungsschrittweite annähern:

$$h = \Delta t \leq \alpha T_{\max}, \quad \alpha \approx \frac{1}{20} \text{ bis } \frac{1}{5}$$

4.3.2 Lineare/Nichtlineare Systemdynamik

Für die Regelungstechnik sehr wichtig sind *lineare Systemdynamiken*, die den allgemeinen Aufbau

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

haben mit $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $u \in \mathbb{R}^l$ und konstanten oder nur Zeitabhängigen Matrizen $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times l}$, $C \in \mathbb{R}^{m\times n}$ und $E \in \mathbb{R}^{m\times l}$.

4.4 Linearisierung um die Ruhelage

- Das Verfahren der Linearisierung selbst ist in 2.10.6 beschrieben.
- Eine Linearisierung um die Ruhelage ist von Nöten, damit die Regelung des Systems (um in der Ruhelage zu verweilen) vereinfacht wird.

4.5 Stabilität

- Ein *stabiles System* ist asymptotisch stabil, d.h. die Veränderungen werden immer kleiner, bis sie schließlich verschwinden.
- Eine mathematische Definition und die Bestimmung, ob ein System stabil ist, ist in 2.10.8 gegeben.

4.6 Regelung

- Die Regelungstechnik beschäftigt sich damit, die Stellgrößen u auf Basis der Systemausgänge y anzupassen, sodass ein stabiler Zustand erreicht wird.
  Vergleich: Regelung der Temperatur des Wassers in einer Badewanne.
- Gewünscht ist, dass das dynamische Modell eine „virtuelle“ Ruhelage x_{soll} (Sollwert) oder $x_{\text{soll}}(t)$ (Solltrajektorie) annimmt.

- Hierzu wird eine geeignete Steuerung $u(t)$ oder Regelung $u(x)$ gesucht, sodass der Sollwert/die Solltrajektorie angenommen stabil wird.

4.6.1 Steuerung und Regelung

Abgrenzung

Die *Steuerung* steuert ein Modell ohne Kenntnis über den Systemausgang (open-loop/feedforward control).

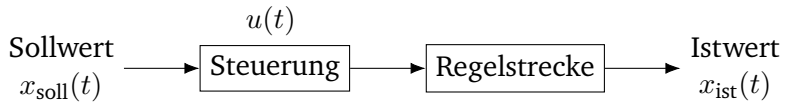

Abbildung 4.2: Steuerung

Die *Regelung* steuert ein Modell mit Kenntnis über den Systemausgang (closed-loop/feedback control).

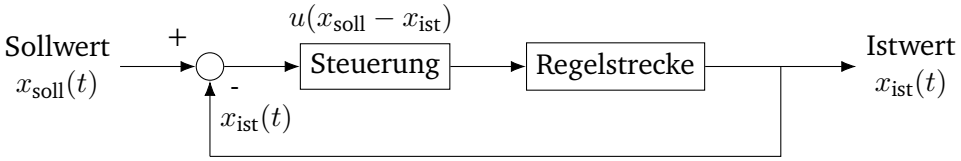

Abbildung 4.3: Regelung

4.6.2 Regelung

- Der Sollwert/die Sollwerttrajektorie soll durch Regelung der Steuerungsparameter u in Abhängigkeit vom aktuellen Zustand x_{ist} stabil angenommen werde.
- Anders Ausgedrückt: Die Stellwerte $u(x)$ müssen so gefunden werden, dass der Realteil aller Eigenwerte der linearisierten Bewegungsdifferentialgleichung um x_{soll} kleiner 0 ist und die Regelung somit stabil ist (nach Ljapunov).
- nach *Regelungssystem* besteht somit aus:
  - Einem *Messglied* zur Messung der regelbaren Größen (der Systemausgänge).
  - Einem *Stellglied* zum Einbringen der Steuerung.
  - Einem *Regler* mit *Regelgesetz* zur Bestimmung der Steuerung aus den Daten des Messglieds.
- Es nun die Frage, *wann* eine solche Regelung existiert und *wie* eine stabile oder sogar optimale Steuerung/Regelung bestimmt werden kann? Hierzu gibt es keine allgemeingültigen Antworten, im folgenden wird nur die Regelung eines linearen Feder-Masse-Systems betrachtet.

4.6.3 Lineares Feder-Masse-System

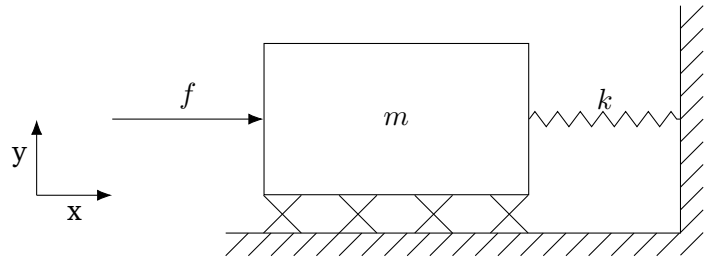

Abbildung 4.4: Lineares Feder-Masse-System

m Masse, $m > 0$

k Federsteifigkeit, $k > 0$

b Reibungskoeffizient, $b > 0$

f Antriebs- oder Bremskraft

Bewegungsgleichung $m\ddot{x}(t) + b\dot{x}(t) + kx(t) = f(t)$

Untersuchung des Bewegungsverhaltens

- Berechnung des Bewegungsverhaltens von $x(t)$ mit Hilfe von:
  - gegebener Anfangsposition $x(0)$,
  - gegebener Anfangsgeschwindigkeit $\dot{x}(t)$ und
  - gegebenem Antriebskraftverlauf $f(t)$.
- Weitere Annahmen:
  - Antriebs-/Bremskraft $f(t) = 0$
  - Ruhelage $x_{\text{stat}}(t) = 0$
    Viele mechanische Systeme sind in einer Ruhelage, wenn keine Antriebs-/Bremskraft wirkt ($\frac{d}{dt}x(t) \to 0$ für $t \to \infty$).

Erwartetes Bewegungsverhalten

- Variante 1
  - Die Federkraft ist schwach (k ist „klein“).
  - Die Reibungskraft ist groß (b ist „groß“).

  $\Longrightarrow$ Die Masse kehrt von einer Auslenkung $x(0) \neq x_{\text{stat}}$ nur langsam in die Ruhelage zurück.
- Variante 2
  - Die Federkraft ist groß (k ist „groß“).
  - Die Reibungskraft ist schwach (b ist „klein“).

  $\Longrightarrow$ die Masse oszilliert nach einer Auslenkung mehrmals schnell hin und her, bevor sie in die Ruhelage zurückkehrt.

Berechnung des Bewegungsverhaltens

Mit der Lösung der Differentialgleichung

$$\ddot{x} + \frac{b}{m}\dot{x} + \frac{k}{m}x = 0$$

durch den allgemeinen Lösungsansatz $x = ce^{\lambda t}$ ergeben sich die Nullstellen (Pole)

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

Die Lage dieser Pole bestimme das Bewegungsverhalten der Masse:

- $\lambda_{1,2}$ sind einfache, reelle Nullstellen $\lambda_{1,2}$, wenn

$$b^2 - 4mk > 0 \iff b^2 > 4mk \implies \lambda_{1,2} < 0$$

  - Hat die Lösung die Form $x = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}$ mit durch Anfangswerte bestimme Konstanten c_1, c_2.
  - Wegen $\lambda_{1,2}$ gilt $x(t) \to 0$ für $t \to \infty$.
  - Dieses Verhalten wird *überkritisch gedämpft* genannt.

- $\lambda_{1,2}$ ist eine doppelte, reelle Nullstelle $\lambda_{1,2} = -\frac{b}{2m} < 0$, wenn

$$b^2 - 4mk = 0 \iff b^2 = 4mk \implies \lambda_{1,2} = -\frac{b}{2m} < 0$$

  - Hat die Lösung die Form $x = c_1 e^{\lambda_1 t} + c_2 t e^{\lambda_2 t}$ mit durch Anfangswerte bestimme Konstanten c_1, c_2.
  - Wegen $\lambda_{1,2}$ gilt $x(t) \to 0$ für $t \to \infty$.
  - Dieses Verhalten wird *kritisch gedämpft* genannt und ist das schnellstmögliche Verhalten, um ohne Oszillation in die Ruhelage überzugehen.

- $\lambda_1 = \lambda_2$ ist eine einfache, komplexe Nullstelle $\mathrm{Re}(\lambda_{1,2}) < 0$, wenn

$$b^2 - 4mk < 0 \iff b^2 < 4mk \implies \lambda_{1,2} = -\frac{b}{2m} \pm i\frac{\sqrt{|b^2 - 4mk|}}{2m} = \mathrm{Re}(\lambda) \pm \mathrm{Im}(\lambda)$$

  - Hat die Lösung die Form $x = c_1 e^{\mathrm{Re}(\lambda)t}\cos(\mathrm{Im}(\lambda)t) + c_2 e^{\mathrm{Re}(\lambda)t}\sin(\mathrm{Im}(\lambda)t)$ mit durch Anfangswerte bestimmte Konstanten c_1, c_2.
  - Wegen $\mathrm{Re}(\lambda) < 0$ gilt $x(t) \to 0$ für $t \to \infty$.
  - Dieses Verhalten wird *unterkritisch gedämpft* genannt und die Masse oszilliert mehrmals um die Ruhelage x_{stat}, bevor sie vollständig in die Ruhelage übergeht.

Gewünschtes Bewegungsverhalten

- Gewünscht ist eine *kritische Dämpfung*, da diese am schnellsten in die Ruhelage zurück kehrt und keine Oszillation auftritt.

PD-Regelung des Feder-Masse-Systems

Der Name PD-Regelung stammt von dem verwendeten Proportions- und Differentialanteil, welche in das Regelgesetz einfließen.

- Ist das natürliche Bewegungsverhalten des Systems nicht wie gewünscht (zu weniger Federkraft, zu viel Federkraft, etc.), so muss das System geregelt werden.
- Dies wird mit Hilfe von geeigneten Sensoren zur Messung der Position und der Geschwindigkeit, der Verwendung einer geeigneten Antriebskraft f und einem geeigneten *Regelgesetz* zur Bestimmung von f geregelt.
- Das Bewegungsverhalten soll so modifiziert werden, dass eine kritische Dämpfung eintritt.
- Ein Ansatz ist folgende Antriebsformel mit den Regelparametern k_v und k_p:

$$f(t) = \underbrace{-k_p x(t)}_{\text{P-Teil}} \underbrace{-k_v \dot{x}(t)}_{\text{D-Teil}}$$

- Eingesetzt in Bewegungsdifferentialgleichung $f = m\ddot{x} + b\dot{x} + kx$ ergibt sich:

$$\begin{aligned} -k_p x - k_v \dot{x} &= m\ddot{x} + b\dot{x} + kx \\ \iff 0 &= m\ddot{x} + \underbrace{(b + k_v)}_{:=\hat{b}} \dot{x} + \underbrace{(k + k_p)}_{:=\hat{k}} x \end{aligned}$$

- Womit sich die Nullstellen $\lambda_{1,2} = -\frac{\hat{b}}{2m} \pm \frac{\sqrt{\hat{b}^2 - 2m\hat{k}}}{2m}$ ergeben.
- Dann ist es nach Abschnitt 4.6.3 notwendig, dass $\hat{b}^2 = 4m\hat{k}$ gilt. Damit Regelfehler bei Instabiltät nicht verstärkt gedämpft werden (k_v und k_p können positiv und negativ gewählt werden), muss außerdem $\hat{b} = 2\sqrt{m\hat{k}} > 0$ gelten.

$$\hat{b}^2 = 4m\hat{k} \quad \text{und} \quad \hat{b} = 2\sqrt{m\hat{k}} > 0$$

- Damit kann das Verhalten auf einen konstanten Sollwert x_d geregelt werden, ähnlich ist dies für eine Sollwerttrajektorie $x_d(t)$ möglich.

Mehr Dimensionen und Nichtlineare Systeme

- Es gibt ähnliche Ansätze, um mehr Dimensionen und auch nichtlineare Systeme zu regeln.
- Dabei ist immer das Ziel, die Polstellen so zu legen, dass ein stabiles System auftritt.

4.6.4 PID-Bahnregelung eines Servomotors

- Die PID-Regelung ist eine Erweiterung eines PD-Reglers, welche zusätzlich zum Proportions- und Differentialanteil noch einen Integralanteil enthält.
- Dabei ist der allgemeine Aufbau des PID-Regelgesetzes:

$$u(t) = u_d(t) + k_P \underbrace{e(t)}_{\text{P-Fehler}} + k_I \underbrace{\int_0^t e(s)\,ds}_{\text{I-Fehler}} + k_V \underbrace{\dot{e}(t)}_{\text{D-Fehler}}$$

- Soll-Position $q_d(t)$ (Gelenkwinkelverlauf)
- Ist-Position $q(t)$
- Soll-Moment $u_d(t)$
- Stellmoment $u(t)$ (Motorantriebsmoment)
- Regelabweichung $e(t) = q_d(t) - q(t)$

- Intuition führ die P-/I-/D-Anteile:

  P-Anteil Überwachung des Ist-Zustandes; Sofortige Reaktion; Überschwingen

  I-Anteil Überwachung der Vergangenheit; Reagiert auf Fehler der Regelung (z.B. Überschwingen)

  D-Anteil Überwachung der „Zukunft“; Reagiert auf Störungen

5 Numerische Simulation

- Um ein gegebenes numerisches Simulationsproblem zu lösen, ist es nötig, eine Implementierung zu finden, welche Rundungsfehler möglichst wenig verstärkt.
- Vorgehen
  1. *Analyse* der Berechnungsaufgabe, ob diese gut oder schlecht *konditioniert* ist.
  2. *Auswahl* eines mathematisch äquivalenten und *numerisch stabilen* Algorithmus, der Rundungsfehler nicht zusätzlich verstärkt.
  3. *Vorsicht* bei beispielsweise:
     - Subtraktion zwei annähernd gleicher Zahlen (Auslöschung).
     - Berechnungen mit relativ großen Zwischenwerten, wenn das Ergebnis klein ist (Auslöschung der Nachkommastellen).
- Treten bei der numerisch stabilen Implementierung eines schlecht konditionierten Problem noch immer Fehlerverstärkung auf, so kann:
  - Die Berechnung mit höherer Genauigkeit durchgeführt werden.
  - Das Ausgangsproblem so modifiziert werden, dass eine bessere Kondition erreicht wird.

5.1 Zahlendarstellung

Die Darstellung von Ganz- und Fließkommazahlen sollte hinreichend bekannt sein und wird hier nicht erneut erläutert.

Siehe https://de.wikipedia.org/wiki/IEEE_754.

5.2 Rundungsfehler

- Da nur eine endliche Menge an Gleitpunktzahlen g verfügbar ist, muss jede reelle Variable x auf die nächstliegende Maschinenzahl abgebildet werden.
- Diese Abbildung wird *Rundung* genannt und kann durch $\forall g : |x - \mathrm{rd}(x)| \leq |x - g|$ beschrieben werden.
- Nach IEEE 754 gibt es folgende Rundungsarten:

  R1 Immer aufrunden.
  Wichtig für Intervallarithmetik.

  R2 Immer abrunden.
  Wichtig für Intervallarithmetik.

  R3 Runden durch abschneiden.

R4 Runden zur nächsten geraden Gleitpunktzahl.
Stellt das Standard-Verfahren dar.

- Damit ergibt sich der relative Rundungsfehler $\varepsilon(x)$:

$$\varepsilon(x) := \frac{x - \mathrm{rd}(x)}{x} \iff \mathrm{rd}(x) = x(1 - \varepsilon(x)) \text{wobei} |\varepsilon(x)| \leq \mathrm{eps} = \varepsilon_{\text{mach}}$$

- eps (Maschinenepsilon) stellt hierbei die Maschinengenauigkeit dar. Der maximale Abstand zweier benachbarten Zahlen beträgt dann $2\,\mathrm{eps}$.
- Das Ergebnis einer arithmetischen Funktion (Addition, Subtraktion, Multiplikation, Division) ist im Allgemeinen keine Maschinenzahl, auch wenn die Operanden Maschinenzahlen sind.
- Um diese Ungenauigkeiten abzubilden, gilt für $x + y$, $x - y$, $x \cdot y$, $\frac{x}{y}$ in IEEE-Arithmetik:

$$\begin{aligned} \mathrm{gl}(x + y) &= (x + y)(1 + \varepsilon_1) \\ \mathrm{gl}(x - y) &= (x - y)(1 + \varepsilon_2) \\ \mathrm{gl}(x \cdot y) &= (x \cdot y)(1 + \varepsilon_3) \\ \mathrm{gl}\left(\frac{x}{y}\right) &= \left(\frac{x}{y}\right)(1 + \varepsilon_4) \\ &\text{mit } |\varepsilon_i| \leq \mathrm{eps} \end{aligned}$$

5.3 Fortpflanzung von Rundungsfehlern

- In der IEEE-Arithmetik gelten Assoziativ- und Distributivgesetz nicht.

$$\begin{aligned} \mathrm{gl}(\mathrm{gl}(x + y) + z) &\neq \mathrm{gl}(x + \mathrm{gl}(y + z)) \\ \mathrm{gl}(a \cdot \mathrm{gl}(b + c)) &\neq \mathrm{gl}(\mathrm{gl}(a \cdot b) \cdot \mathrm{gl}(a \cdot c)) \end{aligned}$$

- Eine Möglichkeit zur Fehleranalyse ist, jeden Ausdruck $\mathrm{gl}(x \diamond y)$ durch $(x \diamond y)(1 + \varepsilon)$ zu ersetzen. Daraus ergibt sich eine Darstellung, in der die von den Eingangparametern abhängigen Verstärkungsfaktoren ersichtlich werden.

5.4 Kondition

- Kernfrage: Wie wirken sich Rundungsfehler in den Eingangsdaten x einer Berechnung $y = f(x)$ auf das Ergebnis aus?
- *Absoluter Fehler* der *Eingangsdaten*: $\Delta x_j := |\tilde{x}_j - x_j|$
- *Relativer Fehler* der *Eingangsdaten*: $\varepsilon_{x_j} := \frac{\Delta x_j}{x_j}$
- *Absoluter Fehler* des *Ergebnisses*: $\Delta y_i := |\tilde{y}_i - y_i| = |f_i(\tilde{x}) - f_i(x)|$
- *Relativer Fehler* des *Ergebnisses*: $\varepsilon_{y_i} := \frac{\Delta y_i}{y_i} \approx \sum_{j=1}^{n} \left(\frac{x_j}{f_i(x)} \frac{\partial f_i(x)}{\partial x_j} \varepsilon_{x_j} \right) =: \sum_{j=1}^{n} (\alpha_j \varepsilon_{x_j})$
- Die Verstärkungsfaktoren α_j heißen *Konditionszahlen*.

  - Sind diese „groß“, so ist das Problem schlecht konditioniert.
  - Sind diese „klein“, so ist das Problem gut konditioniert.
- Intuitiv lässt sich sagen, dass „schlecht konditioniert“ heißt, dass kleine Änderungen der Eingabewerte große Änderungen des Ergebnisses bewirken.
- Die Konditionszahlen hängen nur von der Funktion ab und nicht davon, wie die Funktion ausgewertet wird!

5.5 Numerische Stabilität

- Die *numerischer Stabilität* eines Algorithmus gibt Aufschluss darüber, wie sich eine Implementierung bei Störung der Eingangsvariablen verhält.
- Das heißt, die numerische Stabilität ist (lapidar gesprochen) die Konditionszahl einer Implementierung.
- Definition: Werden die relativen Eingabefehler eines gut konditionierten Problems $y = f(x + \Delta x)$ durch ein Berechnungsverfahren nicht verstärkt, so ist das Verfahren *numerisch stabil*.
  Ein Berechnungsverfahren, welches trotz guter Konditionszahl schlechte Ergebnisse liefert, heißt *numerisch instabil*.
- Effizient Lösbar $\subset$ Numerisch Stabil Lösbar $\subset$ Gut Konditioniert $\subset$ Alle Probleme

6 Numerische Simulation zeitkontinuierlicher Modelle

6.1 Verschiedene Dynamikmodelltypen

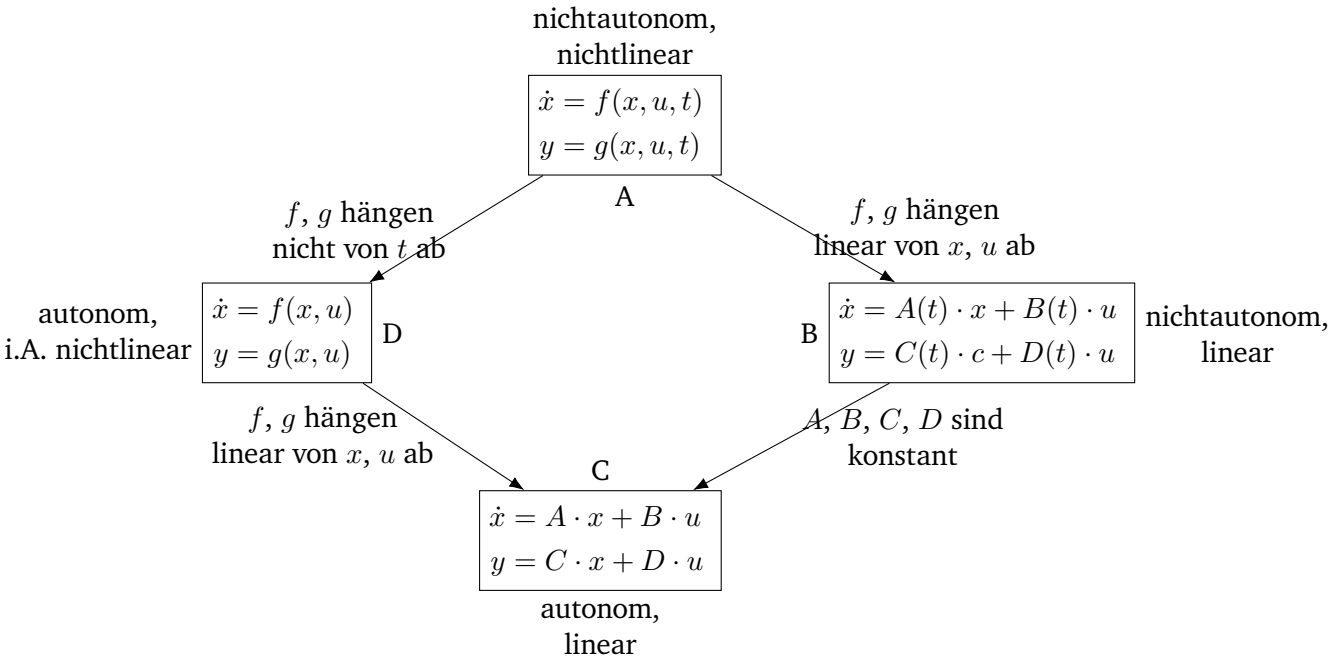

Abbildung 6.1: Dynamikmodelltypen

- Differentialgleichung von Typ A und D sind im allgemeinen nur numerisch lösbar.

6.2 Numerische Integration

- Bei der numerischen Integration wird versucht, eine schwer analytisch zu integrierene Funktion $g(x)$ oder eine Differentialgleichung $\dot{x}(t) = f(x(t))$ zu approximieren.
- Hierbei spielen vor allem folgende Faktoren eine Rolle:
  - Rechenaufwand (Berechnungseffizienz)
  - Genauigkeit (Approximationsfehler)
  - Eignung für steife Systeme (Stabilität)
  - Implementierungsaufwand (Eigenentwicklung oder Verwendung einer Bibliothek)

6.2.1 Einschrittverfahren

- Ein Einschrittverfahren zur Approximation eines Anfangswertproblems $\dot{x} = f(x)$, $x(0) = x_0$ hat den allgemeinen Ansatz:
$$x_{k+1} = x_k + h_k \cdot \Phi(t_k, x_k, x_{k+1}, h; f)$$
mit gegebenem $x_k \approx x(t_k)$.
- Als Konsistenzbedingung muss $\lim_{h \to 0} \Phi(t_k, x_k, x_{k+1}, h; f) = f(x_k)$ gelten.

6.2.2 Explizites Euler-Verfahren

- Das *explizite Euler-Verfahren* ist ein Einschrittverfahren zur Approximation von Differentialgleichungen.
- Es gilt $\Phi = f(x_k)$, eingesetzt in den allgemeinen Ansatz:
$$x_{k+1} = x_k + h_k \cdot f(x_k)$$
- **Rechenaufwand:** Gering, eine Auswertung von f je Schritt.
- **Genauigkeit:** $\mathcal{O}(h_k)$
$\implies$ Für eine hohe Genauigkeit sind sehr kleine Schrittweiten nötig.
$\implies$ Stabilitätsprobleme.

6.2.3 Implizites Euler-Verfahren

- Das *implizite Euler-Verfahren* ist ein Einschrittverfahren zur Approximation von Differentialgleichungen, welches sehr viel bessere Stabilitätseigenschaften als das explizite Euler-Verfahren hat.
- Es gilt $\Phi = f(x_{k+1})$, eingesetzt in den allgemeinen Ansatz:
$$x_{k+1} = x_k + h_k \cdot f(x_{k+1})$$
- Somit erfordert jeder Schritt eine (approximative) Lösung der nichtlinearen Gleichung
$$0 = x_{k+1} - x_k - h_k \cdot f(x_{k+1})$$
- **Rechenaufwand:** Sehr viel höher als bei dem explizitem Newton-Verfahren.
- **Genauigkeit:** $\mathcal{O}(h_k)$
$\implies$ Für eine hohe Genauigkeit sind sehr kleine Schrittweiten nötig. Im Gegensatz zum expliziten Newton-Verfahren ist das implizite aber deutlich stabiler.

6.2.4 Heun-Verfahren

- Das Heun-Verfahren ist ein Prädikator-Korrektor Verfahren und ein 2-stufiges Einschrittverfahren zur Approximation von Differentialgleichungen.
- Prädikator-Schritt: $x_{k+1}^p = x_k + h_k \cdot f(x_k)$ (explizites Euler-Verfahren)

- Korrektor-Schritt: $x_{k+1} = x_k + h_k \cdot \frac{1}{2}\big(f(x_k) + f(x_{k+1}^p)\big)$ (Mittlung des Gradienten)
- Insgesamt ist das Heun-Verfahren ein zweistufiges Einschrittverfahren:

$$\begin{aligned} s_1 &= f(x_k) \\ s_2 &= f(x_k + h_k \cdot s_1) \\ \Phi &= \frac{1}{2}\big(s_1 + s_2\big) \end{aligned}$$

  eingesetzt in den allgemeinen Ansatz:

$$x_{k+1} = x_k + \frac{h_k}{2}(s_1 + s_2)$$

- **Rechenaufwand:** Gering, zwei Auswertungen von f je Schritt.
- **Genauigkeit:** $\mathcal{O}(h^2)$, sehr viel besser als das implizite/explizite Euler-Verfahren.

6.2.5 Runge-Kutta-Verfahren 4. Ordnung

- Das Runge-Kutta-Verfahren ist ein 4-stufiges Einschrittverfahren zur Approximation von Differentialgleichungen.
- Das RK4-Verfahren setzt sich wie folgt zusammen:

$$\begin{aligned} s_1 &= f(x_k) \\ s_2 &= f(x_k + h_k \cdot \alpha_1 \cdot s_1) \\ s_3 &= f(x_k + h_k \cdot \beta_1 \cdot s_1 + h_k \cdot \alpha_2 \cdot s_2) \\ s_4 &= f(x_k + h_k \cdot \gamma_1 \cdot s_1 + h_k \cdot \beta_2 \cdot s_2 + h_k \cdot \alpha_3 \cdot s_3) \\ \Phi &= \delta_1 \cdot s_1 + \delta_2 \cdot s_2 + \delta_3 \cdot s_3 + \delta_4 \cdot s_4 \end{aligned}$$

- Nun wird versucht, die Koeffizienten α_1, α_2, α_3, β_1, β_2, γ_1, δ_1, δ_2, δ_3, δ_4 so zu bestimmen, dass die Konsistenzbedingung erfüllt ist und der Approximationsfehler möglichst gering wird (d.h. $\mathcal{O}(h^4)$) $\implies \delta_1 + \delta_2 + \delta_3 + \delta_4 = 1$.
- Klassischerweise werden folgende Koeffizienten genutzt:

$$\begin{aligned} \alpha_1 = \alpha_2 &= \frac{1}{2} \\ \alpha_3 &= 1 \\ \beta_1 = \beta_2 = \gamma_1 &= 0 \\ \delta_1 = \delta_4 &= \frac{1}{6} \\ \delta_2 = \delta_3 &= \frac{1}{3} \end{aligned}$$

- Was zu folgendem, klassischem, RK4-Verfahren führt:

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

  eingesetzt in den allgemeinen Ansatz:

$$x_{k+1} = x_k + \frac{h_k}{6}(s_1 + 2s_2 + 2s_3 + s_4)$$

- **Rechenaufwand (klassisches RK4):** Vier Funktionsauswertungen von f je Schritt.
- **Genauigkeit (klassisches RK4):** $\mathcal{O}(h^4)$ falls f 4 Mal stetig differenzierbar ist innerhalb von (t_k, r_{k+1}).

6.2.6 Anmerkungen

- Bei gegebener Schrittweite h_k:
  - Der Approximationsfehler bei Verfahren höherer Ordnung (Heun, RK4) ist sehr viel geringer.
  - Dafür ist der Rechenaufwand entsprechend höher.
- Die Auswahl eines Integrationsverfahrens ist abhängig von Anforderung an die Genauigkeit:
  - Je Höher die Anforderung an die Genauigkeit, desto höher sollte die Ordnung des Verfahrens sein. Beispiel: Bei einer Genauigkeitsanforderung von 10^{-4} kann das Heun-Verfahren, bei 10^{-8} sollte besser ein RK7- oder RK8-Verfahren verwendet werden.
  - Für die meisten Anforderungen in ein Verfahren 1. Ordnung (Euler-Verfahren) zu ungenau.
- Allerdings verstärken sich Rundungsfehler bei sinkender Schrittweite stark.

6.2.7 Schrittweitensteuerung

- Eine konstante Schrittweite $h_k = h$ ist
  - *ungenau*, wenn sich die gesuchte Lösung lokal sehr stark ändert und
  - *ineffizient*, wenn sich die gesuchte Lösung lokal sehr wenig ändert.
- Abhilfe schafft hier eine Schrittweitensteuerung, um
  - h_k so „groß wir möglich“ aber „so klein wie nötig“ zu wählen sodass
  - der lokale Approximationsfehler unterhalb einer vorgegebenen Fehlerschranke bleibt.
- Es gibt unterschiedliche Varianten der Schrittweitensteuerung, die üblichen sind:
  1. Ein Einschrittverfahren der Ordnung p mit s Stufen.

- Berechnung von 2 Näherungen für x_{k+1}, einmal mit der Schrittweite h_k und einmal mit zwei Schritten mit jeweils der Schrittweite $\frac{h_k}{2}$.
- Daraus wird der lokale Fehler geschätzt.
- Dann wird ein neuer Schrittweitenvorschlag $\hat{h}_k$ produziert, sodass der Fehler unter der vorgegebenen Schranke liegt.
- Ist der lokale Fehler unter der vorgegebenen Schranke, wird der Schritt akzeptiert (mit der durch $\frac{h_k}{2}$ berechneten Näherung) und es wird mit dem nächsten Schritt fortgefahren.
- Ansonsten wird der Schritt mit dem neuen Schrittweitenvorschlag $\hat{h}_k$ wiederholt.
- **Berechnungsaufwand:** $s + 2s$ Auswertungen von f.

2. Zwei verwandte Verfahren unterschiedlicher Ordnungen $p, \tilde{p}$.
   - Berechnung von 2 Näherungen für x_{k+1} für die selbe Schrittweite h_k.
   - Daraus wird der lokale Fehler geschätzt.
   - **Berechnungsaufwand:** Bei geeigneter Wahl der verfahren unterscheiden sich nur die δ_i-Koeffizienten, sodass nur $s + 1$ Auswertungen nötig sind.
   - Der Ansatz ist sehr viel effizienter als der erste, dafür ist (bei geringer Ordnung) das erste Verfahren etwas robuster.

6.3 Integration steifer Differentialgleichungen

Für eine Definition von steifen Differentialgleichungen siehe 2.10.10.

- Steife Differentialgleichungen sind im allgemeinen schwer zu lösen.
- Zur Berechnung von Lösungen sind ausschließlich implizite Verfahren geeignet.

6.4 Integration von Zustands-Differentialgleichungen mit Unstetigkeiten

- Die bisherigen Verfahren fordern, dass eine Differentialgleichung $\dot{x} = f(x, u, t)$ mindestens so oft *stetig differenzierbar* ist, wie die Ordnung des Verfahrens.
  Ansonsten führt dies zu einem massivem Genauigkeitsverlust.
- Aber: In der realen Welt ist nicht alles stetig differenzierbar.
- Annahmen:
  - Die Funkion f ist abschnittsweise stetig differenzierbar.
  - An den Übergängen der Abschnitte ist f möglicherweise unstetig aber nicht differenzierbar.
  - Die Übergänge werden durch formulierbare Ereignisse (Events) ausgelöst, die zustands- oder zeitabhängig sein können.

6.4.1 Ursachen für Unstetigkeiten

Ursachen für Unstetigkeiten in realen Systemen können beispielsweise sein:

- *Stoßvorgänge:* Position ändert sich stetig, aber nicht differenzierbar.
  Beispiele: Hüpfender Ball; Kollision in physikalisch-basierter Animation
- *Reibung in mechanischen Systemen:* Übergänge zwischen Gleit- und Haftreibung sind meist unstetig.
- *Strukturvariable Systeme:* Die Anzahl der Freiheitsgrade des Zustands ändert sich.
  Beispiele: Abheben eines Roboterbeins vom Boden; Verlust eines Gelenkantriebs eines Roboterarms
- *Approximation von Teilmodellen von* f mit stückweise stetigen Funktionen.
  Beispiel: Lineare Interpolation von Tabellendaten
- *Hysterese:* Ein Teil von f hängt von der aktuellen Vorgeschichte von x ab.
- *Unstetige zeitabhängige Stellgrößen* u*:* Zeitabhängige Eingangsfunktion mit Unstetigkeiten zu bekannten Zeitpunkten (Zeitereignisse)
  Beispiele: Ventil auf/zu; Gangschaltung
- u.v.m.

6.4.2 Schaltfunktionen

- Die *Schaltzeitpunkte* $t_{s,i}$ werden im Allgemeinen als (einfache!) *Nullstellen* n_q reeller *Schaltfunktionen*

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

  modelliert.

- Bei der numerischen Integration müssen nun die *Vorzeichen* der Schaltfunktion beobachtet werden:
  - Wenn zwischen der Näherung $x_k \approx x(t_k)$ und $x_{k+1} \approx x(t_{k+1})$ ein Vorzeichenwechsel in mindestens einer Schaltfunktion auftritt (d.h. die Schaltfunktion schaltet), dann muss:
  - Der dazwischen liegende Schaltpunkt bestimmt werden (mit der vorgegebenen Genauigkeit und der vom Verfahren implizit vorgegebenen Genauigkeit).

    $\rightarrow$ Kombination auf numerischer Integration und numerischer Nullstellensuche.

- Voraussetzung: Die Schrittweite h_k muss klein genug sein, sodass kein doppelter Vorzeichenwechsel innerhalb eines Schrittes stattfindet.

Vorgehensweise

1. Das Modell (die Differentialgleichung) auf **mögliche Unstetigkeiten** in x, der rechten Seite oder den ersten Ableitungen der rechten Seite überprüfen.
2. Bestimmung der **Schaltzeitpunkte** $t_{s,i}$, $i = 1, \cdots, n_s$:

   Zeitgesteuert Zu offensichtlich bekanntem $t_{s,i}$.

   Zustandsabhängig Modellierung 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 unter „genauer“ Einhaltung der Schaltpunkte.
   Möglicherweise spezielle Integrationsverfahren mit Schaltpunktsuche verwenden.

4. **Wiederaufsetzen** nach einem Schaltpunkt (wie ein normales, neues Anfangswertproblem).
   Anfangswert für x aus einer Zustandsübergangsbedingung, zum Beispiel vom Typ $x(t_{s,i}+0) = \xi(x(t_{s,i}-0), t_{s,i})$.

6.4.3 Anmerkungen

Was passiert, wenn der Integrator einfach über die Schaltstelle drüber iteriert?

- Ein *Integrator mit konstanter Schrittweite* merkt gar nichts hiervon.
- Ein *Integrator mit Schrittweitensteuerung* merkt, dass das Modell sich stark ändert.
  - Somit wird die Schrittweite verringert bis die minimale Schrittweite erreicht wurde.
  - Häufig stoppen diese Verfahren nun, da die minimale Schrittweite erreicht wurde, die Genauigkeitsanforderung aber nicht erfüllt werden kann.
  - Alternativ schreitet das Verfahren fort und nimmt den Fehler in kauf.

Dies führt in allen Fällen dazu, dass die gewünschte Genauigkeit nicht erreicht wird.

7 Teilschritte einer Simulationsstudie

Die Teilschritte einer Simulationsstudie sind:

1. Problemspezifikation
    - Aufgabenformulierung
    - Kriterienfestlegung
    - Datenerhebung
    - Siehe 7.1.
2. Modellierung
    - Strukturfestleung
    - Modellgleichungen
    - Modellvereinfachung
    - Siehe 7.2 und 9.
3. Implementierung
    - Auswahl oder Entwicklung eines Berechnungsverfahrens (für das jeweilige Model)
    - Programmierung von Modell und Berechnungsverfahren
    - Visualisierung von Berechnungsergebnissen
    - Laufzeitoptimierung
    - Siehe 7.2
4. Validierung
    - Systematische Plausibilitätsprüfung („Stimmen Modell und Simulation?“)
    - Fehlersuche
    - Konsistenzprüfung
    - Daten-, Parameterabgleich
    - Siehe 8 und 9.
5. Anwendung
    - Simulationsläufe
    - Parametervariation
    - Strukturvariation
    - Vorhersage und Optimierung
    - Siehe 7.4.

Einiger dieser Schritte werden in folgenden tiefer behandelt.

7.1 Problemspezifikation

Kernfrage: Was ist der Zweck der Simulation?

1. Ein bekanntes Szenario *verstehen/nachvollziehen*.
2. Ein bekanntes Szenario *optimieren*.
3. Ein unbekanntes Szenario *vorhersagen*.
4. Wieso nicht die Realität verwenden, welche bessere Ergebnisse liefert?
   - Zu groß, klein (Galaxien, Moleküle)
   - Zu schnell, zu langsam (Flugzeuge, Population, Klima)
   - Noch nicht gebaut (Flugzeuge, Brücken, Roboter)
   - Zu gefährlich (Kernphysik, Humanmedizin)
   - Nicht Experimentierbar (Volkswirtschaft, Ökosysteme)
   - Zu teuer (Auto Crash Test, Raketen, Roboter)
   - Zu stark gestört (Börsenkurse, Wirtschaft)

$\Longrightarrow$ Eine Simulation eignet sich, wenn das reale System nicht oder nur unter großem Aufwand zur Verfügung steht.

7.2 Modellierung

- Die zu entwickelnden Strukturen und Gleichungen zielen nur auf spezifische Aufgaben und Ziele ab! Für unterschiedliche Aufgaben werden unterschiedliche Modelle benötigt.
- Die reale Welt wird aufgeteilt in das zu untersuchende System und seine Umwelt, mit der es über Ein- und Ausgaben in Wechselwirkung steht.
- Das Modell ist ein Ersatzsystem, welches unter Annahmen und Idealisierungen gebildet wird.
- Modellierung bedeutet, durch Abstraktion wichtige Bestandteile und Eigenschaften hervorzuheben und unwichtiges zu entfernen.

7.2.1 Herleitung von Modellen

Bei der Herleitung von Modellen muss ein Gleichgewicht zwischen den folgenden Anforderungen gefunden werden:

1. Um Ergebnisse am Modell nutzen zu können, muss eine ausreichend genaue Modellierung mit der für den Simulationszweck relevanten Parametern durchgeführt werden.
2. Um das Modell einfacher handhaben zu können als die Realität, müssen Details weggelassen werden (Idealisierung und Abstraktion).

Allerdings ist vorher meistens nicht bekannt, welche Merkmale relevant sind. Die Auswirkungen dieser sind im Allgemeinen nicht vorhersagbar.

Bei der Herleitung von Modellen müssen die folgenden Fragen beantwortet werden können:

1. *Was* genau soll modelliert werden?
   Beispiel: Der gesamte Arbeitsraum (Geometrie, Kinematik) oder nur die Kräfte (Kinetik)?
2. Welche *Größen* spielen eine Rolle (qualitativ) und wie groß ist deren Einfluss (quantitativ)?
   Beispiele: Gelenkwinkel, Motorspannungen; Gravitation eines Himmelskörpers

   Achtung: Im allgemeinen sind diese Größen alles andere als offensichtlich!
3. In welchem *Beziehungsgeflecht* stehen die als relevant identifizierten Größen miteinander?
   - Qualitative vs. Quantitative Abhängigkeit (wenn-dann, Vorzeichen oder konkrete Größen der Abhängigkeiten)
   - Typischerweise sind die Beziehungen sehr kompliziert, z.B. Kontaktkräfte zwischen Roboterhand und unterschiedlichen Objekten
4. Mit welchen *Instrumenten* lassen sich die Wechselwirkungen und Abhängigkeiten beschreiben?
   - Algebraische Gleichungen und Ungleichungen
   - Systeme gewöhnlicher Differentialgleichungen (mit einer unabhängigen Variablen, meistens der Zeit)
   - Systeme partieller Differentialgleichungen (mehr als eine unabhängige Variable, meistens zeit und Ort)
   - Automaten, Zustandsübergangsdiagramme
   - Graphen
   - Wahrscheinlichkeitsverteilung
   - Fuzzy Logic
   - Gaußsche Prozesse
   - Neuronale Netze
   - u.v.m.
5. Welche Gestalt hat die *resultierende Berechnungsaufgabe* zur Lösung der Modellbeziehung?
   - Finden *einer Lösung* zu gegebenem Gleichungssystem
   - Finden *der Lösung* zu gegebenem Gleichungssystem
   - Lösen einer Existenzaufgabe
   - Lösen einer beschränkten Optimierungsaufgabe
   - Ermittlung von Flaschenhälsen und Störenfrieden

7.2.2 Zustandsvariablen eines Modells

- Zeitabhängige Größen
- Exakte Festlegung der aktuellen Konfiguration eines Systems

- Festlegung des zukünftigen Verlaufs
- Nicht redundant

Die Festlegung der Zustandsvariablen ist der Ausgangspunkt jeder Modellbildung.

7.2.3 Klassifikation der Zustandsänderungen

- Zeit-, Wertkontinuierlich
- Zeit-, Wertdiskret
- Zeitdiskret, Wertkontinuerlich
- Ereignisdiskret, Wertdiskret
- Stochastisch

7.3 Implementierung

7.3.1 Klassifikation zeitkontinuierlicher Simulationswerkzeuge

Level 3 Multidisziplinäre Modellgenerierung (Modelica, VHDL-A, ...)

Level 2 Graphische Modellierung (SIMULINK, WorkingModel, Aspen, STELLA, ...)
Spezialsimulatoren (ADAMS, SIMPACK, KSIM, ...)

Level 1 Simulationssprachen (ACSL, VHDL, Dare-P, Desire, ...)
Simulationsframeworks (SIMULINK, C++-Klassenbibliothek)

Level 0 Direkte Programmierung (C++, C, MATLAB, FORTRAN)

7.4 Anwendung

- „Laufen lassen“ der Simulation; Simulieren
- Dieser Schritt nimmt, im Vergleich zu den vorherigen Schritten, nur wenig (Menschen-) Zeit ein.
- Die Berechnung selbst kann natürlich lange dauern.

8 Validierung

- Zu dem zu validierenden *Simulationsmodell* gehören das *Modell*, die *Implementierung* und die *Berechnungsverfahren*.
- Die Validierung eines Simulationsmodell beschäftigt sich mit der *systematischen Plausibilitätsprüfung* und damit der Fehlersuche, Konsistenzprüfung und Daten-/Parameterabgleich.
- Eine Lösung kann nur akzeptiert werden, wenn in allen vier Schritten (Modellierung, Diskretisierung, Abbruch- und Rundungsfehler, Visualisierungsfehler) vergleichbar kleine Fehler gemacht wurden.

8.1 Fehlerquellen

- *Modellierungsfehler* (Vereinfachte Modellannahmen, Ungenaue Modellparameter)
- *Approximationsfehler* der Berechnungsverfahren
- *Rundungsfehler*
- *Programmier-, Implementierungsfehler*

8.2 Begriffe und Definitionen

Verifikation

- Normaler (meist mathematischer) Nachweis der Korrektheit, dass ein Programm einer vorgegebenen Spezifikation entspricht.
- Aufgrund der großen Anzahl an Parametern, Zustandsverläufen und Störungseinflüssen ist es in der Regel unmöglich, die vollständige Korrektheit des gesamten Systems zu beweisen.
- Durch geeignete Anzahl und systematischer gesuchter Tests kann die Wahrscheinlichkeit der Korrektheit erhöht werden.
  → Validierung

Validierung

- Plausibilitätsprüfung, dass ein Programm einer vorgegebenen Spezifikation entspricht.
- Ziel der Validierung ist der Nachweis der ausreichenden Glaubwürdigkeit des Simulationsmodells im Hinblick auf den Einsatzbereich.
- Validierung ist wichtig, da dadurch das Risiko von verheerenden Zwischenfällen und Falschaussagen über die Realität stark verringert wird.

8.3 Vorgehensweise

Die Validierung erfordert fachliche Einsicht und Kreativität! Es gibt keine pauschal guten Lösungen.

8.3.1 Validierung der Implementierung (D, L, V)

I1 *Syntaktische Fehlerfreiheit* der Implementierung

I2 *Plausibilitätsprüfung* der Simulationsergebnisse für Spezialfälle („naturgetreues“ Verhalten)

I3 *Numerische Korrektheit* der Implementierung (zum Beispiel durch Vergleich der berechneten Lösung und einer analytischen Lösung)

8.3.2 Validierung des Modells (M)

M1 *Zulässigkeit und logische Konsistenz* der Modellannahmen (und deren Anwendbarkeit zur Problemlösung)

M2 *Ausreichende Detailliertheit* des Modells (und korrekte Modellstruktur)

M3 *Korrektheit (Genauigkeit)* der Modellparameter

8.3.3 Validierung des Berechnungsverfahrens (D, L)

B1 *Eignung* für die numerische Lösung des Modells

B2 *Approximationsfehler* des (iterativen) Berechnungsverfahrens

B3 Einfluss von *Rundungsfehlern*

8.3.4 Tests auf Plausibilität und Konsistenz

T1 *Reproduktion* von beobachtetem bzw. „natürlichen“ Systemverhalten
Wie gut stimmt das berechnete verhalten mit den beobachteten/bekannten Erkenntnissen überein?

T2 *Vorhersage* von Verhalten
Wie plausibel ist das simulierte Verhalten für noch nicht beobachtete Szenarien?
Können die Ergebnisse eines Experiments mit ausreichender Genauigkeit vorhergesagt werden?

T3 *Verhaltensanomalien*
Stark unterschiedliches/widersprüchliches Verhalten des Modells gegenüber dem realen System weißt auf große Fehler im Modell oder im Berechnungsverfahren oder in der Implementierung hin.

T4 Systemverhalten in *Extremsituation*
Entspricht das simulierte Verhalten in extremen Szenarien den Erwartungen, auch wenn diese in der Realität selten/nie auftreten?

T5 *Parametervariationen und Parametersensitivität*
Ist das Verhalten sensitiv zu plausiblen Variationen der Modellparameter?

9 Identifikation von Modellen

9.1 Systemidentifikation

Das Ziel der Systemidentifikation ist zu bestimmen, wie die Ausgangsgrößen von den Eingangsgrößen quantitativ abhängen, also der Bestimmung eines Systemmodells f mit geeigneten Parameterwerten p (der rechten Seite von $\dot{x} = f(x(t), p)$).

Dies hat folgenden Nutzen:

- Bessere (d.h. genauere) Vorhersagen durch Simulation mit den identifizierten Modellen.
- Validierung von bestehenden Simulationsmodellen.

9.1.1 Arten der und Modellbildung

Arten von Modellbildung:

1. Spekulation
   Beispiel: Soziale Systeme
2. Vorhersage
   Beispiel: Ökonomische Systeme
3. Analyse
   Beispiel: Biologische Systeme
4. Steuerung
   Beispiel: Chemische Systeme, Mechanische Systeme
5. Design
   Beispiel: Mechanische Systeme, Elektrotechnische Schaltungen

Des weiteren können die obigen Modellbildungsarten in die drei Oberkategorien „Black Box“ und „White Box“ eingeteilt werden:

- Black Box
  - Modellbildung auf Basis des beobachteten Verhaltens am Ausgang in Abhängigkeit vom Eingang.
  - Induktives Vorgehen bei der Modellierung.
  - Generische Ansätze, die allgemeine Funktionen repräsentieren, z.B. $\dot{x} = \hat{f}(x(t), p)$.
    Die Parameter p beschreiben dann die Struktur und die Ausgestaltung der Ansatzfunktion(en) von $\hat{f}$.
- White Box

- Modellbildung auf Basis grundlegender Gesetzmäßigkeiten und der Systemstruktur.
- Deduktives Vorgehen bei der Modellierung.
- Aus grundlegenden Gesetzmäßigkeiten (Mechanik, Elektrotechnik, ...) abgeleitete Modelle, z.B. Differentialgleichungen $\dot{x} = f(x(t), p)$.
  Die Parameter p können mit experimenteller Daten identifiziert, bzw. kalibriert werden.

Außerdem gibt es noch den Ansatz einer „Grey Box“, welche Ansätze von Black und White Box vereint.

9.1.2 Hauptschritte der Systemidentifikation

- *Strukturidentifikation*
  Stimmt das Verhalten von f mit den vorhandenen Daten und generellen Erkenntnissen überein?
- *Parameteridentifikation*
  Die Struktur ist bereits gegeben, es müssen die Modellparameter p gefunden werden, sodass die Abweichungen zwischen experimentellen und simuliert Daten möglichst klein werden.

> **Warning:** Die beste Lösung ist nicht unbedingt die mit dem kleinsten Fehler. Es müssen auch andere Faktoren, zum Beispiel Differenzierbarkeit betrachtet werden.

9.2 Parameteridentifikation

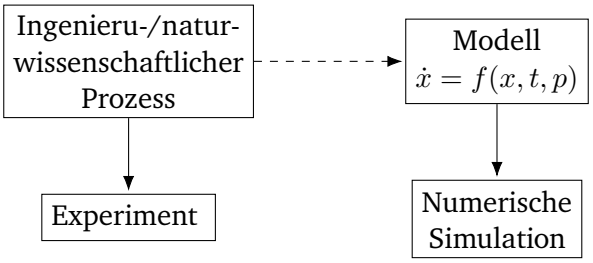

Abbildung 9.1: Parametereinschätzung bei DGL-Systemen

- Kalibrierung der Modellparameter p: $\eta_{ij} = x_i(t_j) + \varepsilon_{ij}$
  - n_{ij} beschreiben die experimentell gemessenen Werte von $x_i(t_j)$.
  - $x(t_j, p)$ beschreibt das Ergebnis der numerischen Simulation mit Parameter p.
- Es sollen nun die Quadrate der Abweichungen minimiert werden ($\omega_{ij} = const > 0$ sind die Gewichtungsfaktoren):

$$\min_{p \in \mathbb{R}^n} \varphi(p) = \frac{1}{2} \sum_{j=1}^{n_t} \sum_{i \in I_j} \omega_{ij} \cdot \big(\underbrace{\eta_{ij} - x_i(t_j, p)}_{:= r_{ij}(p)} \big)^2$$

- Nebenbedingung. x ist eine numerische Lösung des AWP

$$\dot{x}_i = f_i(x, t, p), \quad x_i(0) = x_{i,0}, \quad i = 1, \cdots, n_x$$

9.2.1 Schema der Parameteridentifikation

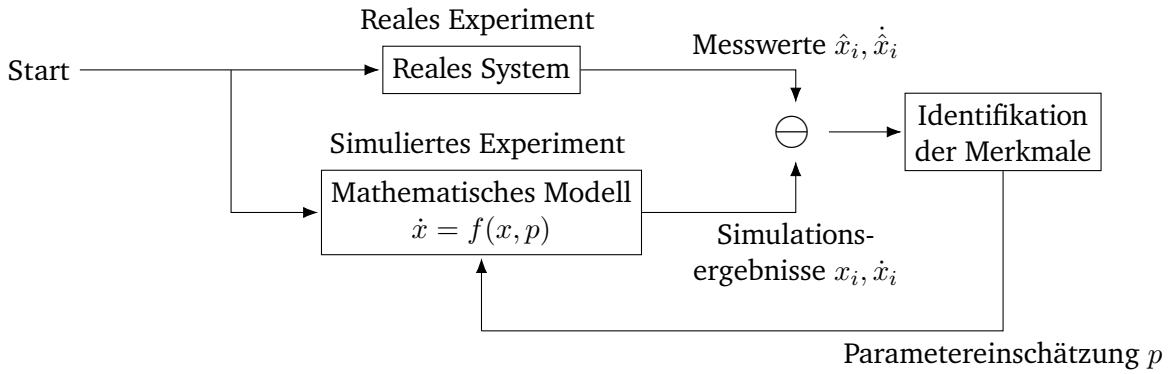

Abbildung 9.2: Schema der Parameteridentifikation

9.2.2 Gütekriterien

Zur Bewertung (und Minimierung) des Abweichungsfehlers sind unterschiedliche Gütekriterien möglich, beispielsweise:

- Definition: $r_i(p) \coloneqq \eta_{ij} - x_i(t_j, p)$
- Summe der Absolutbeträge der Abweichungen

$$\varphi_1(p) = \|r(p)\|_1 = \sum_{i=1}^{n_r} |r_i(p)|$$

- Summe der „kleinsten Quadrate“

$$\varphi_2(p) = \frac{1}{2}\|r(p)\|_2^2 = \frac{1}{2}\sum_{i=1}^{n_r} \big(r_i(p)\big)^2$$

- Maximale Abweichung

$$\varphi_\infty(p) = \max\{|r_i(p)| : i = 1, \cdots, n_r\}$$

Bei der Wahl der Gütefunktion ist jedoch einiges zu beachten:

- Bei der Auswahl ist zu beachten, wie das numerische Optimierungsverfahren zur Berechnung funktioniert und wie gut es mit der ausgewählten Gütefunktion zusammenarbeiten kann.
- Auch wenn r differenzierbar ist, sind φ_1 und φ_∞ im Allgemeinen *nicht differenzierbar*. Differenzierbar erleichtern die Suche nach einem Minimum immens (durch Nullstellenfindung auf $\frac{\partial}{\partial p}\varphi_i(p)$).
- φ_∞ ist jedoch sehr sensitiv gegenüber einzelnen Ausreißern.
- φ_2 ist stetig differenzierbar, aber wenig sensitiv gegenüber einzelnen Ausreißern.
- Generell liegen bei durch Minimierung von Abweichung produzierten Optimierungsproblemen viele lokale Minima vor, die globale Optimierung ist somit nicht gegeben.

Nichtlineare, Kleinste Quadrate

- Meistens wird das verfahren der kleinsten Quadrate (φ_2) verwendet (u.a. wegen der Differenzierbarkeit).
- Auch ist das Verfahren aus statistischen Gründen interessant:
  Wenn
  - die Messfehler ε_{ij} unabhängig und
  - normalverteilt sind mit Median 0 und konstanter Varianz ω^2,

  dann ist die Lösung des Optimierungsproblems eine Lösung mit größtmöglicher Wahrscheinlichkeit.
- Durch Ausnutzung der Struktur von φ_2 kann die Effizienz der numerischen Optimierung stark erhöht werden.

9.2.3 Kalibrierung

Erfolg

- Können die Parameter p so berechnet werden, dass die Abweichungen des Modells von dem Experiment „klein“ sind, dann „stimmen“ Modell und Parameter.

Misserfolg

- Können keine Parameter p gefunden werden, dann ist möglicherweise
  1. das Optimierungsverfahren ungeeignet,
  2. die experimentellen Daten nicht ausreichend relevant für die Bestimmung der Parameter,
  3. die Messfehler in den Messdaten zu groß oder Ausreißer verfälschen die Messwerte oder
  4. das verwendete Modell ist nicht detailliert genug oder enthält nicht alle/nicht die richtigen Effekte.

10 Physikalisch basierte Spiele

10.1 Definition

Was ist ein Computerspiel? Ein Spiel ist ein interaktives Erlebnis, welches den Spieler mit immer schwereren Aufgaben fordert, in denen er oder sie neue Techniken lernt und die Aufgaben eventuell meistert.

- In den meisten Fällen wird ein mathematisches Modell zur Approximation und Simulation der Welt verwendet.
- In agenten-basierten Simulation interagieren mehrere eigenständige Einheiten („Agenten“), zum Beispiel Charaktere, Fahrzeuge, ...
- Alle interaktiven Spiele sind zeitliche Simulationen, das heißt die Spielwelt ist dynamisch und der Zustand ändert sich mit der Zeit.
- Ein Computerspiel muss nur auf begrenzt viele und vorhersehbare Eingaben des Spielers reagieren.
- Die meisten Spiele präsentieren die Geschichten und reagieren auf den Spieler in Echtzeit. Das heißt, sie sind interaktive Echtzeit-Simulationen, die sich dementsprechend an gewisse Beschränkungen halten müssen (ein nicht-einhalten hat in der Regel aber keine dramatischen Folgen).

10.2 Game Loop

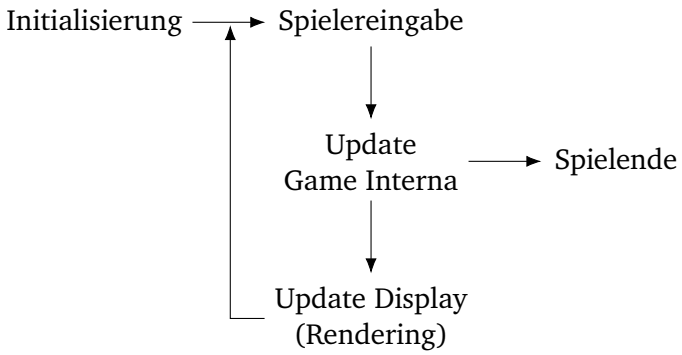

Abbildung 10.1: Game Loop der Hauptereignisse

10.3 Game Engine

Die Game Engine enthält unter anderem:

- Eine API, mit der das Spiel selbst kommunizieren kann.
- Scripting Engine
- AI Engine
- Physics Engine
- Sound Engine
- Graphics Engine
- Netzwerke
- I/O.Geräte
- Schnittstellen und Kernsystem (mathematische Funktionen, Datenstrukturen, Speicher-/Dateiverwaltung/...)

Auf der untersten Ebene Kommuniziert die Game Engine mit den Hardware-Abstraktions-Schichten (z.B. DirectX und OpenGL).

10.4 Physics & Collision Engine

Physikalisch basierte Spiele...

- enthalten die Simulation grundlegender, physikalischer Eigenschaften als Hauptbestandteil.
- Bei Computerspielen ist dies vor allem die Newtonsche Mechanik.
- Oftmals werden neben starren Körpern auch deformierbare Körper (dynamische Körper) verwendet, zum Beispiel bei der Simulation von Kleidung.

Physik Engine

- Die Physik-Engine ist eine Sammlung von Algorithmen, welche physikalische Eigenschaften simulieren bzw. Tools zur Verfügung stellen, um mit diesen zu arbeiten.
- Grundlegend wird zwischen zwei arten von Physik Engines unterschieden:

  Echtzeit Einige Details werden vernachlässigt, um die Rechenzeit zu minimieren.
  Der Fokus der Anwendung liegt hier besonders auf Computerspielen oder kleinen Simulationen.

  Hohe Genauigkeit Nahezu alle physikalischen Größen der realen Welt werden berücksichtigt, die Rechenzeit wird missachtet.
  Der Fokus der Anwendung liegt hier auf der Forschung und Entwicklung neuer Systeme, bei denen die exakten Werte relevant sind.

- Allerdings ist die Implementierung von vollwertigen Physik Engines in Spielen nicht immer vorteilhaft, da sie einen hohen Rechenaufwand produzieren. Allerdings steigt durch die Genauigkeit der Simulation die Qualität des Spiels.
- Folgende Kernelemente gehören in jede Physik Engine:

- Typen von Objekten und deren Bewegungsverhalten
  Objekte: Starrkörper, Partikelsysteme, deformierbare Körper, ...
  Bewegungsverhalten: Integration von Differentialgleichungssystemen
- Kontaktmodellierung, -detektion und -behandlung; Erkennen von Ort, Kraft und/oder Impuls von Kollision zwischen ruhenden, verbundenden und kollidierenden Objekten.

10.5 Modellierung eines Objektes

- Punktmasse
- Masse mit Trägheitstensor (Beschreibung der räumlichen Masseverteilung des Objektes)
- Verbundene Partikel (starr oder elastisch)

10.6 Kontake, Kollisionen, Kräfte und Impulse

- Beispiel: Buch liegt auf einem Tisch
  - Keine Geschwindigkeit, keine Bewegung
  - Das Buch drückt mit immer der selben Kraft auf den Tisch.
- Beispiel: Aufprallender Ball
  - Dynamischer Rückstoß bei Aufprall auf dem Boden.
  - Die Änderung der Ballgeschwindigkeit wird von der Aufprallenergie beeinflusst.
  - Kollisionsbehandlung (Abprall) erfolgt instantan und nicht für eine sehr kurze Zeitdauer.
- Die meisten Physik Engines behandeln entweder alles über *Kräfte* oder alles über *Impulse*.
  - Kräfte-basierte Physik Engines sind hierbei näher an der realen Welt, der Berechnungsaufwand ist aber extrem groß.
  - Impuls-basierte Physik Engines haben meist einen geringeren Berechnungsaufwand und ähnliches Objektverhalten wie Kräfte-basierte Engines.

11 Simulation autonomer Roboter

- Roboter sind komplexe technische Systeme bestehend aus Aktuatoren, Sensoren, Recheneinheiten, sehr vielen Algorithmen, ...
- Sie treten in vielen Anwendungen und Erscheinungsformen auf (Roboterarme, mobile Roboter, selbstfahrende Autos, ...).
- Die Simulation eines Roboters umfasst
  1. Roboterbewegung,
  2. Robotersensoren und
  3. Interaktion des Roboters mit der Umgebung.

11.1 Problemspezifikation

Aufgaben und Anforderungen an die Simulation:

- Auslegung und Abschätzung von Hardware (Antriebe, Sensoren, ...)
- Berechnung der optimalen Bewegungsabläufe
- Auslegung und Tests von Reglern
- Test der Steuerungssoftware von autonomen Robotern
- u.v.m.
- Je nach Aufgabe unterscheiden sich die Anforderungen an die Simulation drastisch.
- Da die Steuerungssoftware sehr komplex sein kann, ist es nötig, diese sehr gut zu testen.
- Durch die Zeit- und Kostenintensive Nutzung realer Roboter ist eine geeignete Robotersimulation notwendig.

12 Beispiele aus der Forschung

Dieser Abschnitt wird nicht in der Zusammenfassung behandelt und ist im Foliensatz nachzulesen.