



Universität  
Augsburg  
University

**Vergleich der Integrationsmethoden und der Methoden des  
maschinellen Lernens für gewöhnliche Differentialgleichungen**

**Alexandro Jedaidi**

Universität Augsburg

Mathematisch-Naturwissenschaftlich-Technische Fakultät

19. April 2022

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Theorie gewöhnlicher Differentialgleichungen</b>	<b>3</b>
2.1	Gewöhnliche Differentialgleichungen und Anfangswertprobleme . . . . .	3
2.2	Existenz und Eindeutigkeit . . . . .	5
2.2.1	Existenz von Lösungen . . . . .	5
2.2.2	Eindeutigkeit von Lösungen . . . . .	7
2.3	Abhängigkeit der Lösungen von den Daten . . . . .	10
<b>3</b>	<b>Numerischer Lösungsansatz</b>	<b>13</b>
3.1	Einschrittverfahren . . . . .	13
3.2	Runge-Kutta-Verfahren . . . . .	16
3.2.1	Konsistenz . . . . .	18
3.2.2	Schrittweitensteuerung . . . . .	21
3.3	Mehrschrittverfahren . . . . .	23
3.4	Verfahren für steife Differentialgleichungen . . . . .	28
<b>4</b>	<b>Neuronale Netze</b>	<b>29</b>
4.1	Struktur neuronaler Netze . . . . .	29
4.1.1	Forwardpropagation . . . . .	30
4.1.2	Backwardpropagation . . . . .	30
4.1.3	Gradientenverfahren . . . . .	32
4.1.4	Verschwindender und explodierender Gradient . . . . .	33
4.1.5	Batch Training . . . . .	34
4.2	Prinzip der Lösungspakete . . . . .	34
4.3	Intervall-Lernen . . . . .	37
4.4	Gewichtsinitialisierung . . . . .	37
<b>5</b>	<b>Numerische Auswertung</b>	<b>38</b>
5.1	Rebound-Pendel . . . . .	39
5.2	Steife Differentialgleichung . . . . .	41
5.3	Harmonischer Oszillator . . . . .	44
<b>6</b>	<b>Fazit</b>	<b>54</b>
	<b>Abbildungsverzeichnis</b>	<b>56</b>

# 1 Einleitung

Gewöhnliche Differentialgleichungen lassen sich in der heutigen Zeit in unterschiedlichsten Anwendungen finden. Es ist beispielsweise möglich, die Bewegungen eines Körpers unter Einfluss der Anziehung anderer Planeten durch ein System gewöhnlicher Differentialgleichungen darzustellen. Andererseits können durch sogenannte Lotka-Volterra-Gleichungen, oder auch als Räuber-Beute-Gleichungen bekannt, die Wechselwirkung von Räuber- und Beutetierpopulation beschrieben werden, was wiederum Anwendung in der Biologie findet. Hierbei fällt auf, dass hohes Interesse an einer Lösung dieser Systeme besteht, wobei nicht für jede gewöhnliche Differentialgleichung eine Existenz dieser Lösung garantiert ist.

Falls jedoch die Existenz und gegebenenfalls sogar die Eindeutigkeit einer Lösung des gegebenen Systems vorliegt, ist es meistens mit viel Aufwand verbunden, diese Lösung analytisch zu ermitteln. Deshalb haben sich in den letzten Jahrhunderten die numerische Vorgehensweise zur Lösung gewöhnlicher Differentialgleichungen entwickelt. Diese Verfahren wurden über die Zeit optimiert und durch den exponentiellen Anstieg an Rechenleistung im 21. Jahrhundert können heutzutage sogar Systeme hoher Komplexität mithilfe dieser numerischen Verfahren in angemessener Zeit und brauchbarer Genauigkeit gelöst werden.

Mit der rasanten Entwicklung der Computer fanden aber auch Verfahren des maschinellen Lernens zunehmende Anwendungsbereiche. Allgemein finden die sogenannten neuronalen Netze des maschinellen Lernens ihre Anwendung in beispielsweise der Bild- oder Spracherkennung. Jedoch ist die Eigenschaft neuronaler Netze, Funktionen darstellen zu können, nützlich zur Approximation der Lösung einer gewöhnlichen Differentialgleichung.

In dieser Arbeit werden wir demnach den Vergleich von numerischen Verfahren mit neuronalen Netzen zur Lösung gewöhnlicher Differentialgleichungen diskutieren. Zu Beginn werden wir in Kapitel 2 die Theorie von gewöhnlichen Differentialgleichungen aufarbeiten. Daraufaufgehend behandeln wir in Kapitel 3 die Einschritt- und Mehrschrittverfahren, sowie die BDF-Verfahren, welche als numerische Verfahren für gewöhnliche Differentialgleichungen gelten. Im Anschluss untersuchen wir eine sogenannte Methode der Lösungspakete mit neuronalen Netzen in Kapitel 4 und zuletzt vergleichen wir in Kapitel 5 die behandelten Verfahren anhand drei verschiedenen Anfangswertproblemen mithilfe einiger Fehlermaße.

## 2 Theorie gewöhnlicher Differentialgleichungen

In diesem Abschnitt wird die Theorie zu gewöhnlichen Differentialgleichungen erläutert. Diese orientiert sich hauptsächlich an den Arbeiten [peterdeuflhardfolkmarbornemannNumerischeMathematikGewöhnlicheDifferentialgleichungen, ernsthairergerhardwannerSolvingOrdinaryDifferential, berndaulbachGewöhnlicheDifferentialgleichungen]. Dieses Kapitel ist Grundlage für das Verständnis der Schlussfolgerungen und Ergebnisse dieser Arbeit.

### 2.1 Gewöhnliche Differentialgleichungen und Anfangswertprobleme

Wir beginnen mit der allgemeinen Definition einer gewöhnlichen Differentialgleichung erster Ordnung.

**Definition 2.1.1** *Ein System gewöhnlicher Differentialgleichungen erster Ordnung hat die Form*

$$x'(t) = f(t, x(t)) \quad (2.1)$$

mit der gegebenen Funktion  $f : D \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ , wobei  $D \subseteq \mathbb{R}$  ein Zeitintervall ist. Eine dazugehörige Lösung (falls existent)  $\hat{x} : D \rightarrow \mathbb{R}^n$  ist eine stetig, differenzierbare Funktion welche die Bedingung

$$\hat{x}' = f(t, \hat{x}).$$

erfüllt.

**Definition 2.1.2** *Ein Anfangswertproblem für eine Differentialgleichung (2.1) mit gegebenen Anfangswerten  $x_0 \in \mathbb{R}^n$  und der Anfangszeit  $t_0 \in D \subseteq \mathbb{R}$  hat die Form*

$$x' = f(t, x), \quad x(t_0) = x_0. \quad (2.2)$$

Eine Lösung des Problems  $\hat{x} : D \rightarrow \mathbb{R}^n$  muss also zusätzlich zu (2.1) auch die Anfangswertbedingungen (2.2) erfüllen.

**Bemerkung 2.1.3** *Neben Differentialgleichungen erster Ordnung existieren auch gewöhnliche Differentialgleichungen  $m$ -ter Ordnung. Diese haben die Form*

$$x^{(m)}(t) = f(t, x(t), x'(t), x''(t), \dots, x^{(m-1)}(t)), \quad (2.3)$$

dabei ist  $f : D \times \mathbb{R}^n \times \mathbb{R}^n \times \dots \times \mathbb{R}^n \rightarrow \mathbb{R}^n$  und  $D \subseteq \mathbb{R}$  ein Zeitintervall. Eine dazugehörige Lösung (falls existent)  $\hat{x} : D \rightarrow \mathbb{R}^n$  ist eine  $m$ -mal stetig, differenzierbare Funktion welche die Bedingung

$$\hat{x}^{(m)} = f(t, \hat{x}, \hat{x}', \hat{x}'', \dots, \hat{x}^{(m-1)}).$$

erfüllt. Das zugehörige Anfangswertproblem  $m$ -ter Ordnung mit gegebenen Anfangswerten  $x_{0,1}, \dots, x_{0,m} \in \mathbb{R}^n$  und der Anfangszeit  $t_0 \in D \subseteq \mathbb{R}$  erfüllt zusätzlich die Bedingungen

$$x(t_0) = x_{0,1}, \quad x'(t_0) = x_{0,2}, \dots, \quad x^{(m-1)}(t_0) = x_{0,m}.$$

Außerdem ist es möglich jede gewöhnliche Differentialgleichung  $m$ -ter Ordnung zu einem System gewöhnlicher Differentialgleichungen erster Ordnung umzuwandeln. Die gewöhnliche Differentialgleichung  $m$ -ter Ordnung ist mit Hilfe der Funktionen  $x_j : D \rightarrow \mathbb{R}$  für  $j = 1, \dots, m$  äquivalent zu

einem System erster Ordnung mit  $m$  Gleichungen:

$$\begin{aligned} x_1' &= x_2, \\ x_2' &= x_3, \\ &\vdots \\ x_{m-1}' &= x_m, \\ x_m' &= f(t, x_1, x_2, x_3, \dots, x_m). \end{aligned} \tag{2.4}$$

Für ein Anfangswertproblem  $m$ -ter Ordnung gilt zusätzlich:

$$\begin{aligned} x_1(t_0) &= x_{0,1}, \\ x_2(t_0) &= x_{0,2}, \\ &\vdots \\ x_{m-1}(t_0) &= x_{0,m-1}, \\ x_m(t_0) &= x_{0,m}. \end{aligned} \tag{2.5}$$

Deshalb werden wir uns in dieser Arbeit ohne Beschränkung der Allgemeinheit auf gewöhnliche Differentialgleichungen erster Ordnung beschränken.

**Definition 2.1.4** Eine gewöhnliche Differentialgleichung (2.1) heißt autonom, wenn die rechte Seite  $f$  nicht explizit von der unabhängigen Variable  $t$  abhängt. Das bedeutet, es gilt

$$x' = f(x)$$

auf ganz  $D$ .

Ähnlich wie bei einer Ordnungsreduktion in Bemerkung 2.1.3 können wir auch eine nichtautonome, gewöhnliche Differentialgleichung zu einem autonomen System gewöhnlicher Differentialgleichungen umwandeln. Dazu führen wir eine neue Variable  $z$  ein, für die gilt

$$z' = 1, \quad z(t_0) = t_0.$$

Dann ist das Anfangswertproblem (2.2) äquivalent zu dem autonomen Anfangswertproblem zweiter Ordnung

$$\begin{aligned} z' &= 1, & z(t_0) &= t_0 \\ x' &= f(z, x), & x(t_0) &= x_0. \end{aligned} \tag{2.6}$$

$$\tag{2.7}$$

Die Lösung von dem Anfangswertproblem (2.6) hat also die Form  $\begin{bmatrix} t \\ x(t) \end{bmatrix}$ . Weiterhin definieren wir in diesem Abschnitt noch eine Sonderform einer gewöhnlichen Differentialgleichung.

**Definition 2.1.5** Ein System gewöhnlicher Differentialgleichungen

$$x'(t) = A(t)x(t) + g(t) \quad (2.8)$$

mit  $A : D \rightarrow \mathbb{R}^{n \times n}$  und  $g : D \rightarrow \mathbb{R}^n$  heißt lineares System gewöhnlicher Differentialgleichungen. Falls  $g \equiv 0$ , so nennt man (2.8) homogen, sonst inhomogen.

Wir definieren nun steife lineare Differentialgleichungen unter der Bedingung, dass die Koeffizientenmatrix  $A$  konstant, also unabhängig von  $t$  ist.

**Definition 2.1.6** Eine lineare Differentialgleichung (2.8) mit  $A \in \mathbb{R}^{n \times n}$  und  $g(t) \in \mathbb{R}^n$  heißt steif, wenn

1.  $\operatorname{Re}(\lambda_j(A)) < 0$ , für alle Eigenwerte  $\lambda_j(A)$  von  $A$ ,
2.  $\min_{1 \leq j \leq n} \operatorname{Re}(\lambda_j(A)) \ll \max_{1 \leq j \leq n} \operatorname{Re}(\lambda_j(A))$ .

Wir werden später ein Verfahren betrachten, welches sich speziell dafür eignet, eine steife gewöhnliche Differentialgleichung zu lösen.

## 2.2 Existenz und Eindeutigkeit

In diesem Abschnitt betrachten wir ein Anfangswertproblem erster Ordnung

$$\begin{aligned} x' &= f(t, x) \\ x(t_0) &= x_0 \end{aligned} \quad (2.9)$$

und zeigen, unter welchen Bedingungen der rechten Seite  $f(t, x(t))$  eine Lösung existiert und gegebenenfalls eindeutig ist, welcher aus der Literatur [1, S. 13, 14] stammt.

### 2.2.1 Existenz von Lösungen

Zuerst werden wir einen Satz beweisen, welcher zeigt, dass die Stetigkeit der rechten Seite  $f$  für die Existenz einer Lösung ausreicht. Dazu benötigen wir noch einen Satz aus der Funktionalanalysis.

**Satz 2.2.1 (Fixpunktsatz von Schauder, 2. Version)** Sei  $M$  eine nichtleere, abgeschlossene und konvexe Teilmenge eines Banachraums  $X$  und  $A : M \rightarrow M$  stetig. Dann besitzt  $A$  mindestens einen Fixpunkt  $x$ , sofern die Bildmenge  $A(M)$  relativ kompakt ist.

*Beweis.* [1, S. 13, 14]

In dieser Arbeit wird mit  $\|\cdot\|_2$  die euklidische Norm  $\|x\|_2 = (\sum_{i=1}^n (x_i)^2)^{\frac{1}{2}}$  für  $x \in \mathbb{R}^n$  beschrieben. Der Beweis des folgenden Satzes beruht auf das Buch [2].

**Satz 2.2.2 (Existenzsatz von Peano)** Seien  $t_0 \in \mathbb{R}$ ,  $x_0 \in \mathbb{R}^n$ ,

$$\mathcal{G} = \{(t, x) \in \mathbb{R} \times \mathbb{R}^n : |t - t_0| \leq \alpha, \quad \|x - x_0\|_2 \leq \beta, \quad \alpha, \beta > 0\}$$

und  $f : \mathcal{G} \rightarrow \mathbb{R}^n$  stetig. Dann besitzt das Anfangswertproblem (2.9) mindestens eine Lösung  $\hat{x}$  auf dem Intervall  $D = (t_0 - a, t_0 + a)$ , wobei

$$a = \min\left\{\alpha, \frac{\beta}{M}\right\}, \quad M = \max_{(t, y) \in \mathcal{G}} \|f(t, y)\|_2.$$

*Beweis.* Sei  $J \subset D$  abgeschlossen mit  $t_0 \in J$  und  $\hat{x}$  eine Lösung von (2.9). Da die rechte Seite  $f$  auf  $\mathcal{G}$  stetig ist, gilt nach dem Hauptsatz der Differential- und Integralrechnung:

$$\hat{x}(t) = \hat{x}_0 + \int_{t_0}^t f(s, \hat{x}(s)) \, ds \quad \text{für alle } t \in J. \quad (2.10)$$

Außerdem gilt für jede auf  $J$  stetige Funktion  $\hat{x}(t)$ , die obige Gleichung erfüllt, dass sie auf  $J$  differenzierbar ist. Dies bedeutet, dass eine auf  $J$  stetige Funktion  $\hat{x}(t)$  löst genau dann das Anfangswertproblem (2.9), wenn es die obige Integralgleichung erfüllt. Für eine auf  $J$  stetige Funktion  $\hat{x}(t)$  gilt also:

$\hat{x}(t)$  löst das AWP (2.9), g.d.w.  $\hat{x}(t)$  erfüllt die Gleichung (2.10).

Betrachte nun die Menge  $K := \{x \in C(D) : \|x(t) - x_0\|_2 \leq \beta \text{ für alle } t \in D\}$ , wobei  $C(D) = \{g : D \rightarrow \mathbb{R}^n \mid g \text{ ist stetig auf } D\}$ . Die Menge  $K$  ist offensichtlich nicht leer (betrachte bspw. die konstante Funktion  $x \equiv x_0$ ). Jedem  $x \in K$  wird nun eine Funktion  $Ax \in C(D)$  zugewiesen:

$$(Ax)(t) := x_0 + \int_{t_0}^t f(s, x(s)) \, ds \quad \text{für alle } t \in D.$$

Also gilt für  $(Ax)(t)$  die Gleichung (2.10) und somit löst  $Ax$  nach obiger Äquivalenz das Anfangswertproblem auf  $D$ . Außerdem gilt:

$$\begin{aligned} \|(Ax)(t) - x_0\|_2 &= \left\| \int_{t_0}^t f(s, x(s)) \, ds \right\|_2 \\ &\leq \int_{t_0}^t \|f(s, x(s))\|_2 \, ds \\ &\leq |t - t_0| \max_{(t,y) \in \mathcal{G}} \|f(t, y)\|_2 \\ &\leq |t_0 + a - t_0| M \\ &\leq \frac{\beta}{M} M = \beta \quad \text{für alle } t \in D. \end{aligned}$$

Das bedeutet, dass  $Ax \in K$  für  $x \in K$ , bzw.  $AK \subset K$ , also bildet  $A$  von  $K$  nach  $K$  ab. Nun können wir die hergeleitete Äquivalenz umformulieren: Für eine Funktion  $\hat{x} \in K$  gilt:

$$\hat{x} \text{ löst auf } D \text{ das Anfangswertproblem (2.9)} \Leftrightarrow \hat{x} \text{ ist ein Fixpunkt der Abbildung } A : K \rightarrow K. \quad (2.11)$$

Dazu nutzen wir den Fixpunktsatz von Schauder 2.2.1. Betrachte den Banachraum  $C(D)$  mit der Maximumsnorm

$$\|x\|_\infty := \max_{t \in D} \|x(t)\|_2.$$

Sei  $(x_k)_{k \in \mathbb{N}} \subset K$  eine konvergente Folge mit Grenzwert  $x \in C(D)$ . Dann ist

$$\|x(t) - x_0\|_2 = \left\| \lim_{k \rightarrow \infty} (x_k(t)) - x_0 \right\|_2 = \lim_{k \rightarrow \infty} \|x_k(t) - x_0\|_2 \leq \lim_{k \rightarrow \infty} \beta = \beta.$$

Also ist  $x \in K$  und somit  $K$  abgeschlossen. Sei  $x, y \in K$  beliebig und definiere  $v := \lambda x + (1 - \lambda)y$  für  $0 \leq \lambda \leq 1$ . Dann ist  $v \in K$ , da

$$\begin{aligned}\|v(t) - x_0\|_2 &= \|\lambda x(t) + (1 - \lambda)y(t) - \lambda x_0 - (1 - \lambda)y_0\|_2 \\ &= \lambda \|x(t) - x_0\|_2 + (1 - \lambda) \|y(t) - y_0\|_2 \\ &\leq \beta(\lambda + 1 - \lambda) = \beta,\end{aligned}$$

für alle  $t \in D$ . Das heißt,  $K$  ist außerdem konvex. Die Stetigkeit der Abbildung  $A : K \rightarrow K$  lässt sich wie folgt zeigen. Wir bestimmen ein  $\delta > 0$ , sodass mit  $\|x - y\|_2 < \delta$  gilt:

$$\|f(t, x) - f(t, y)\|_2 < \frac{\epsilon}{a}, \quad t \in D.$$

Somit ist  $f$  gleichmäßig stetig. Betrachte nun  $x, y \in K$  mit  $\|x - y\|_\infty < \delta$  also auch  $\|x(t) - y(t)\|_2 < \delta$  für alle  $t \in D$ , so ist für  $t \in D$  immer  $\|f(t, x(t)) - f(t, y(t))\|_2 < \frac{\epsilon}{a}$  und es folgt

$$\|(Ax)(t) - (Ay)(t)\|_2 \leq |t - t_0| \frac{\epsilon}{a} \leq |t_0 + a - t_0| \frac{\epsilon}{a} = \epsilon \text{ für alle } t \in D,$$

also auch  $\|Ax - Ay\|_\infty \leq \epsilon$ . Es bleibt jetzt nur noch zu zeigen, dass die Bildmenge  $AK$  relativ kompakt ist. Sei hierfür  $x \in K$  beliebig, dann ist

$$\|(Ax)(t)\|_2 \leq \|x_0\|_2 + aM \quad \text{für jedes } t \in D$$

und

$$\|(Ax)(t_1) - (Ax)(t_2)\|_2 \leq |t_1 - t_2|M \quad \text{für alle } t_1, t_2 \in D.$$

Somit ist  $A(K)$  punktweise beschränkt und gleichgradig stetig auf dem kompakten Intervall  $D$ . Nach dem Satz von Arzela-Ascoli [3, S. 49] hat also jede Folge  $(Ax)_n \subset AK$  eine gleichmäßig konvergente Teilfolge. Dies gilt offensichtlich auch für die Maximumsnorm, weshalb  $AK$  relativ kompakt ist.  $\square$

### 2.2.2 Eindeutigkeit von Lösungen

Ähnlich wie im vorherigen Abschnitt existiert ein Satz, welcher zeigt, dass eine im zweiten Argument Lipschitz-stetige rechte Seite  $f$  ausreicht, damit eine eindeutige Lösung für ein Anfangswertproblem (2.2) existiert. Dazu definieren wir zuerst den Begriff der Lipschitz-Stetigkeit einer Funktion.

**Definition 2.2.3** Sei  $\mathcal{M} \subset \mathbb{R} \times \mathbb{R}^n$ . Eine Funktion  $f : \mathcal{M} \rightarrow \mathbb{R}^n$  heißt genau dann Lipschitz-stetig auf  $\mathcal{M}$  bezüglich des zweiten Arguments, falls ein  $L > 0$  existiert, sodass gilt:

$$\|f(t, x) - f(t, y)\|_2 \leq L \|x - y\|_2$$

für alle  $(t, x), (t, y) \in \mathcal{M}$ .

Eine ähnliche Definition findet sich in [ernsthairergerhardwannerSolvingOrdinaryDifferential]. Außerdem benötigen wir für den Beweis noch den Fixpunktsatz von Weissinger aus [2].

**Satz 2.2.4 (Fixpunktsatz von Weissinger)** Sei  $(B, \|\cdot\|)$  ein Banachraum und  $U \subset B$  abgeschlossen und nichtleer. Ferner ist  $\sum_{j=1}^{\infty} \alpha_j$  eine konvergente Reihe positiver Zahlen, also



$\sum_{j=1}^{\infty} \alpha_j < \infty$ ,  $\alpha_j \geq 0$ , sowie  $A : U \rightarrow U$  eine Selbstabbildung, für die

$$\|A^j u - A^j v\| \leq \alpha_j \|u - v\| \quad \text{für alle } u, v \in U \quad \text{und } j \in \mathbb{N}$$

gilt. Dann besitzt  $A$  genau einen Fixpunkt in  $U$ , nämlich

$$u = \lim_{n \rightarrow \infty} A^n u_0$$

mit beliebigem  $u_0 \in U$ . Außerdem ist  $u$  der Grenzwert der rekursiven Folge  $u_j := Au_{j-1}$  für  $j \geq 1$  mit  $u_0 \in U$  beliebig und es gilt die Fehlerabschätzung

$$\|u - u_n\| \leq \sum_{j=n}^{\infty} \alpha_j \|u_1 - u_0\|.$$

*Beweis.* [2, S. 139]

Nun können wir einen grundlegenden Satz in der Theorie gewöhnlicher Differentialgleichungen beweisen, welcher sich ähnlich wie der Beweis des Satzes von Peano an auf Überlegungen von H. Heuser in [2] beruht. Wir werden in dieser Arbeit die gegebene Beweisidee etwas genauer ausführen.

**Satz 2.2.5 (Existenz- und Eindeutigkeitssatz von Picard-Lindelöf)** Seien  $t_0 \in \mathbb{R}$  sowie  $x_0 \in \mathbb{R}^n$ ,

$$\mathcal{G} = \{(t, x) \in \mathbb{R} \times \mathbb{R}^n : |t - t_0| \leq \alpha, \quad \|x - x_0\|_2 \leq \beta, \quad \alpha, \beta > 0\},$$

und  $f : \mathcal{G} \rightarrow \mathbb{R}^n$  stetig und Lipschitz-stetig im zweiten Argument. Dann besitzt das Anfangswertproblem (2.9) genau eine Lösung  $\hat{x}$  auf dem Intervall  $D = (t_0 - a, t_0 + a)$ , wobei

$$a = \min\left\{\alpha, \frac{\beta}{M}\right\}, \quad M = \max_{(t,y) \in \mathcal{G}} \|f(t, x)\|_2.$$

*Beweis.* Nun ist  $f$  Lipschitz-stetig, also setzen wir  $B = C(D)$  mit der Maximumsnorm,  $U = K$  und  $A : K \rightarrow K$  aus dem Beweis von Satz 2.2.2. Wir zeigen mit Induktion über  $j \in \mathbb{N}$ , dass

$$\|(A^j u)(t) - (A^j v)(t)\|_2 \leq \frac{|t - t_0|^j}{j!} L^j \|u - v\|_{\infty}, \quad u, v \in K$$

für alle  $j \in \mathbb{N}$  und  $t \in D$  gilt. Die Maximumsnorm  $\|\cdot\|_\infty$  wurde bereits in dem Beweis von Satz 2.2.2 definiert.

$$\begin{aligned}
 (j=1): \quad \|(Au)(t) - (Av)(t)\|_2 &\leq \underbrace{\|x_0 - x_0\|_2}_{=0} + \left\| \int_{t_0}^t f(s, u(s)) - f(s, v(s)) \, ds \right\|_2 \\
 &\leq |t - t_0| L \|u - v\|_\infty \\
 \text{(IV):} \quad \|(A^j u)(t) - (A^j v)(t)\|_2 &\leq \frac{|t - t_0|^j}{j!} L^j \|u - v\|_\infty \\
 (j \rightarrow j+1): \quad \|(A^{j+1} u)(t) - (A^{j+1} v)(t)\|_2 &\leq \underbrace{\|x_0 - x_0\|_2}_{=0} + \left\| \int_{t_0}^t f(s, (A^j u)(s)) - f(s, (A^j v)(s)) \, ds \right\|_2 \\
 &\leq \int_{t_0}^t \|f(s, (A^j u)(s)) - f(s, (A^j v)(s))\|_2 \, ds \\
 &\leq L \int_{t_0}^t \|(A^j u)(s) - (A^j v)(s)\|_2 \, ds \\
 &\stackrel{\text{IV}}{\leq} L \frac{1}{j!} L^j \|u - v\|_\infty \int_{t_0}^t |s - t_0|^j \, ds \\
 &= \frac{1}{j!} L^{j+1} \|u - v\|_\infty \frac{(s - t_0)^{j+1}}{j+1} \Big|_{t_0}^t \\
 &= \frac{|t - t_0|^{j+1}}{(j+1)!} L^{j+1} \|u - v\|_\infty.
 \end{aligned}$$

Nun folgt direkt

$$\|A^j u - A^j v\|_\infty \leq \frac{(aL)^j}{j!} \|u - v\|_\infty.$$

Die Reihe  $\sum_{j=1}^{\infty} \frac{(aL)^j}{j!}$  ist nach dem Quotientenkriterium offensichtlich konvergent. Also können wir den Fixpunktsatz von Weissinger 2.2.4 anwenden. Der daraus gewonnene Fixpunkt ist eindeutig und mit Anwendung der Äquivalenz (2.11) aus dem Beweis von Peano 2.2.2 folgt, dass der Fixpunkt die gesuchte eindeutige Lösung ist.  $\square$

Für lineare Differentialgleichungen erster Ordnung (2.8) mit einer konstanter Matrix  $A$  gilt wegen

$$\|Ax + g(t) - Ay - g(t)\|_2 \leq \|A\|_2 \|x - y\|_2,$$

dass die rechte Seite Lipschitz-stetig in dem zweiten Argument ist, weshalb mit Satz 2.2.5 die Existenz und Eindeutigkeit einer Lösung folgt. Die Lösung  $x$  kann mit Hilfe der *Matrixexponentialfunktion* [4] angegeben werden:

$$x(t) = e^{A(t-t_0)} x_0 + \int_{t_0}^t e^{A(t-s)} g(s) \, ds. \quad (2.12)$$

Einfaches Ableiten und Einsetzen von  $t_0$  zeigt, dass diese Funktion die Differentialgleichung (2.8) tatsächlich löst.

**Bemerkung 2.2.6** Man kann zeigen, dass für  $X \subseteq \mathbb{R} \times \mathbb{R}^n$  offen und jede Funktion  $f : X \rightarrow \mathbb{R}^n$ , die stetig und stetig differenzierbar im zweiten Argument ist,  $f$  Lipschitz-stetig ist.

### 2.3 Abhängigkeit der Lösungen von den Daten

In Kapitel 5 werden wir gewöhnliche Differentialgleichungen betrachten, die zur Simulation bzw. Vorhersage abgeschlossener Systeme genutzt werden. Darin ist es üblich, dass Anfangsdaten durch Messfehler von tatsächlichen Daten abweichen. Deshalb ist es sinnvoll, Aussagen zu betrachten, die zeigen, welche Auswirkungen kleine Störungen der Anfangsdaten auf die Lösung der Differentialgleichungen haben. In diesem Abschnitt ist die rechte Seite  $f$  stetig und Lipschitz-stetig im zweiten Argument, sodass die Eindeutigkeit der Lösung durch den Satz von Picard-Lindelöf 2.2.5 garantiert ist. Um eine Aussage über die stetige Abhängigkeit von den Daten treffen zu können, beweisen wir zuerst einen wichtigen Hilfssatz. Die Überlegungen und Ergebnisse in diesem Abschnitt stammen aus [3].

**Satz 2.3.1 (Gronwallsche Ungleichung)** *Seien  $D = [t_0, t_f]$  ein Intervall und die stetige, nicht-negative Funktion  $u : D \rightarrow \mathbb{R}$  sowie  $a \geq 0, b > 0$  gegeben. Des Weiteren gilt die Ungleichung*

$$u(t) \leq \alpha \int_{t_0}^t u(s) \, ds + \beta$$

für alle  $t \in D$ . Dann gilt:

$$u(t) \leq e^{\alpha(t-t_0)} \beta$$

für alle  $t \in D$ .

*Beweis.* Definiere zuerst eine Hilfsfunktion

$$v(t) := \alpha \int_{t_0}^t u(s) \, ds + \beta.$$

Für diese gilt

$$v'(t) = \alpha u(t) \leq \alpha v(t)$$

für alle  $t \in D$ . Daraus folgt

$$(e^{-\alpha t} v(t))' = e^{-\alpha t} (v'(t) - \alpha v(t)) \leq 0, \quad t \in D.$$

Die Funktion  $e^{-\alpha t} v(t)$  ist also monoton fallend, das bedeutet

$$e^{-\alpha t} v(t) \leq e^{-\alpha t} v(t) \stackrel{t \geq t_0}{\leq} e^{-\alpha t_0} v(t_0) = e^{-\alpha t_0} \beta.$$

Daraus folgt die Behauptung. □

Außerdem benötigen wir noch folgendes Lemma.

**Lemma 2.3.2** *Sei  $\mathcal{M} \subset \mathbb{R} \times \mathbb{R}^n$  offen und  $f : \mathcal{M} \rightarrow \mathbb{R}$  eine stetige Funktion, die zusätzlich Lipschitz-stetig im zweiten Argument ist mit*

$$\|f(t, x) - f(t, y)\|_2 \leq L \|x - y\|_2$$

für alle  $(t, x), (t, y) \in \mathcal{M}$  und  $L > 0$ . Ist  $\hat{x}$  eine stetig differenzierbare Funktion auf dem Intervall  $D \subset \mathbb{R}$  und eine Lösung des Anfangswertproblems (2.9) und ist  $\hat{x}_a$  eine stetig differenzierbare

Funktion und eine Näherungslösung mit  $(t, \hat{x}_a(t)) \in \mathcal{M}$  für alle  $t \in D$  und es gilt

$$\begin{aligned}\|\hat{x}'_a(t) - f(t, \hat{x}_a(t))\|_2 &\leq d_f \quad t \in D, \\ |t_0 - \tilde{t}_0| &\leq d_t, \\ \|x_0 - \hat{x}_a(\tilde{t}_0)\|_2 &\leq d_a\end{aligned}$$

( $d_f$  repräsentiert die Störung der rechten Seite,  $d_t$  die Störung der Anfangszeit und  $d_a$  die Störung des Anfangswerts). Dann gilt die Abschätzung

$$\|\hat{x}(t) - \hat{x}_a(t)\|_2 \leq e^{L|t-t_0|} (d_a + d_t(d_f + \sup_{s \in D} \|f(s, \hat{x}_a(s))\|_2) + \frac{d_f}{L}) - \frac{d_f}{L}.$$

*Beweis.* Betrachte zuerst die Differenz der Lösung  $\hat{x}$  und  $\hat{x}_a$  bei  $t = t_0$ :

$$\begin{aligned}\|\hat{x}(t_0) - \hat{x}_a(t_0)\|_2 &= \left\| \hat{x}_0 - \int_{\tilde{t}_0}^{t_0} \hat{x}'_a(s) ds - \hat{x}_a(\tilde{t}_0) \right\|_2 \\ &\leq \left\| \hat{x}_0 - \hat{x}_a(\tilde{t}_0) \right\|_2 + \left\| \int_{\tilde{t}_0}^{t_0} [\hat{x}'_a(s) - f(s, \hat{x}_a(s))] ds \right\|_2 + \left\| \int_{\tilde{t}_0}^{t_0} f(s, \hat{x}_a(s)) ds \right\|_2 \\ &\leq d_a + d_t(d_f + \sup_{s \in D} \|f(s, \hat{x}_a(s))\|_2).\end{aligned}$$

Nun können wir mit Hilfe der Lipschitz-Eigenschaft der rechten Seite  $f$  die Differenz für allgemeines  $t \in D, t > t_0$  abschätzen:

$$\begin{aligned}\|\hat{x}(t) - \hat{x}_a(t)\|_2 &= \left\| \hat{x}_0 + \int_{t_0}^t f(s, \hat{x}) ds - \hat{x}_a(t_0) - \int_{t_0}^t \hat{x}'_a(s) ds \right\|_2 \\ &\leq \|\hat{x}_0 - \hat{x}_a(t_0)\|_2 + \int_{t_0}^t [\|f(s, \hat{x}(s)) - f(s, \hat{x}_a(s))\|_2 + \|\hat{x}'_a(s) - f(s, \hat{x}_a(s))\|_2] ds \\ &\leq d_a + d_t(d_f + \sup_{s \in D} \|f(s, \hat{x}_a(s))\|_2) + \int_{t_0}^t [L \|\hat{x}(s) - \hat{x}_a(s)\|_2 + d_f] ds.\end{aligned}$$

Um das Gronwallsche Lemma verwenden zu können, setzen wir  $u(t) := \|\hat{x}(t) - \hat{x}_a(t)\|_2 + \frac{d_f}{L}$ ,  $\beta := d_a + d_t(d_g + \sup_{s \in D} \|f(s, \hat{x}_a(s))\|_2) + \frac{d_f}{L}$  und  $\alpha := L$ . Offensichtlich gilt dann

$$u(t) \leq \alpha \int_{t_0}^t u(s) ds + \beta.$$

Also können wir das Gronwallsche Lemma anwenden und somit folgt

$$\|\hat{x}(t) - \hat{x}_a(t)\|_2 + \frac{d_g}{L} \leq e^{L(t-t_0)} \left[ d_a + d_t(d_f + \sup_{s \in D} \|f(s, \hat{x}_a(s))\|_2 + \frac{d_g}{L}) \right].$$

Der Beweis für  $t \in D$  mit  $t < t_0$  funktioniert analog.  $\square$

Mit diesem Lemma können wir nun Aussagen über die stetige Abhängigkeit der Lösung von den Daten treffen.

**Satz 2.3.3** Sei  $\mathcal{M} \subset \mathbb{R} \times \mathbb{R}^{1+n}$  offen und  $f : \mathcal{M} \rightarrow \mathbb{R}^n$  eine stetige Funktion, die Lipschitz-stetig im zweiten Argument mit Konstante  $L > 0$  gegeben. Dann hängt die Lösung  $x$  des Anfangswertproblems (2.9) stetig von den Anfangsdaten  $(t_0, x_0) \in \mathcal{M}$  und der rechten Seite  $f$  ab. Darunter versteht man: sind eine Lösung  $x$  auf einem kompakten Intervall  $D \subset \mathbb{R}$ , eine Umgebung  $U$  des Graphen  $\{(t, x(t)) : t \in D\}$  in  $\mathcal{M}$  und ein  $\epsilon > 0$  gegeben, dann existiert ein  $\delta = \delta(\epsilon, U, f, D) > 0$ ,

sodass die Lösung  $\hat{x}$  des gestörten Anfangswertproblems

$$\begin{aligned}\hat{x}' &= \hat{f}(t, \hat{x}) \\ \hat{x}(\hat{t}_0) &= \hat{x}_0\end{aligned}$$

auf ganz  $D$  existiert und die Abschätzung

$$\|x(t) - \hat{x}(t)\|_2 \leq \epsilon, \quad t \in D$$

erfüllt. Voraussetzungen hierfür sind, dass  $\hat{t}_0 \in D$ ,  $(\hat{t}_0, \hat{x}_0) \in \mathcal{M}$ ,  $\hat{f}$  stetig auf  $U$ , Lipschitz-stetig im zweiten Argument und

$$|t_0 - \hat{t}_0| \leq \delta, \quad \|x_0 - \hat{x}_0\|_2 \leq \delta, \quad \|f(t, x) - \hat{f}(t, x)\|_2 \leq \delta, \quad (t, x) \in U$$

gilt.

*Beweis.* Nach Anpassung von  $\epsilon$  können wir annehmen, dass  $\overline{B_\epsilon((t, u(t)))} \subset U$  für jedes  $t \in D$  gilt. Wir definieren  $\hat{D}_{\max}$  als das maximale Existenzintervall der Lösung  $\hat{x}$  des gestörten Anfangswertproblems. Nun gilt

$$\|\hat{x}'(t) - f(t, \hat{x}(t))\|_2 = \|f(t, \hat{x}(t)) - \hat{f}(t, \hat{x}(t))\|_2 \leq \epsilon$$

für alle  $t \in \hat{D}_{\max}$ . Setzen wir  $d_f = d_t = d_a = \delta$ , so können wir Lemma 2.3.2 anwenden und erhalten

$$\|x(t) - \hat{x}(t)\|_2 \leq e^{L|t-t_0|}(\delta + \delta(\delta + \sup_{s \in D} \|f(s, \hat{x}(s))\|_2) + \frac{\delta}{L})$$

für alle  $t \in D \cap \hat{D}_{\max}$ . Hierbei ist  $\delta > 0$  frei wählbar, weshalb wir  $\delta$  in Abhängigkeit von  $\epsilon$ ,  $\sup_{s \in D} \|f(s, \hat{x}(s))\|_2$ ,  $L$  und  $|t - t_0|$  so bestimmen, dass

$$\|x(t) - \hat{x}(t)\|_2 \leq \epsilon, \quad \text{für } t \in D \cap \hat{D}_{\max}$$

gilt. Die Lösung  $\hat{x}$  verlässt die kompakte Menge

$$\{(s, y) \in \mathbb{R} \times \mathbb{R}^n : \|(s, y) - (t, x(t))\|_2 \leq \epsilon \text{ für ein } t \in D\} \subset \mathcal{M}$$

nicht. Also können wir mit einem Satz über das Verhalten der maximalen Lösung aus [3, Satz 3.21] zeigen, dass  $D \subset \hat{D}_{\max}$  gilt. Also haben wir die Fehlerabschätzung  $\|x(t) - \hat{x}(t)\|_2 \leq \epsilon$  für  $t \in D$  und die Existenz der Lösung  $\hat{x}$  des gestörten Anfangswertproblems auf  $D$  gezeigt.  $\square$

### 3 Numerischer Lösungsansatz

Dieses Kapitel beruht im Wesentlichen auf [ernsthairergerhardwannerSolvingOrdinaryDifferential, prof.dr.josefstoeerNumerischeMathematik, 5]. Wir werden uns im Folgenden das Anfangswertproblem

$$\begin{aligned} x' &= f(t, x) \\ x(t_0) &= x_0 \end{aligned} \tag{3.1}$$

betrachten, wobei  $f$  stetig ist.

Unser Ziel ist es, eine effektive Methode zur Lösung eines Anfangswertproblem zu verwenden. Da es nicht immer möglich ist, eine gewöhnliche Differentialgleichung analytisch zu lösen, gibt es sogenannte Ein- und Mehrschrittverfahren, welche eine Lösung der gewöhnlichen Differentialgleichung approximiert. Der Grundgedanke dieser Verfahren ist es das Zeitintervall  $D = [t_0, t_0 + a]$  zu diskretisieren,  $t_0 < t_1 < \dots < t_N = t_0 + a$  und beginnend mit dem Anfangswert  $t_0$  eine Näherung  $u_i \approx x(t_i)$  für  $i = 0, \dots, N$  zu berechnen. Solche Verfahren werden mit Hilfe von Quadraturformeln der numerischen Integration [6, Numerische Integration] hergeleitet. Dies basiert darauf, dass die Differentialgleichung auf einem Teilintervall  $[t_i, t_{i+1}] \subset D$  integriert wird und nach dem Hauptsatz der Differential- und Integralrechnung auf folgende Gleichung gebracht werden kann:

$$x(t_{i+1}) - x(t_i) = \int_{t_i}^{t_{i+1}} x'(t) dt = \int_{t_i}^{t_{i+1}} f(t, x(t)) dt.$$

Das Integral kann nun mit einer bereits genannten Quadraturformel approximiert werden, welches je nach Wahl der Quadraturformel ein Einschrittverfahren bildet.

#### 3.1 Einschrittverfahren

In diesem Abschnitt betrachten wir die Einschrittverfahren.

**Definition 3.1.1** Sei  $D = [t_0, t_0 + \alpha]$  ein Zeitintervall und  $D_h = \{t_i = t_0 + ih \text{ für } i = 0, \dots, N\}$  eine Zerlegung von  $D$  mit der Schrittweite  $h = \frac{\alpha}{N}$ . Ein explizites Einschrittverfahren für das Anfangswertproblem (3.1) hat die Form

$$\begin{aligned} u_0 &= x_0 \\ u_{i+1} &= u_i + h\phi(t_i, u_i, h, f), \quad i = 0, \dots, N-1. \end{aligned} \tag{3.2}$$

Dabei nennt man  $\phi : D_h \times \mathbb{R}^n \times \mathbb{R} \times C(D \times K, \mathbb{R}^n) \rightarrow \mathbb{R}^n$  die Inkrementfunktion, wobei  $K \subseteq \mathbb{R}^n$ . Implizite Einschrittverfahren haben die Form

$$\begin{aligned} u_0 &= x_0 \\ u_{i+1} &= u_i + h\phi(t_i, u_i, u_{i+1}, h, f), \quad i = 0, \dots, N-1, \end{aligned}$$

mit  $\phi : D_h \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \times C(D \times K, \mathbb{R}^n) \rightarrow \mathbb{R}^n$ .

Zur Analyse der Approximationsqualität der Einschrittverfahren werden neue Fehlermaße eingeführt.

**Definition 3.1.2 (Lokaler Diskretisierungsfehler)** Sei das Anfangswertproblem (3.1) mit im zweiten Argument Lipschitz-stetiger rechten Seite  $f$  und ein explizites Einschrittverfahren (3.2)

gegeben. Für  $\hat{t} \in [t_0, t_0 + a]$  und  $0 < h < t_0 + a - \hat{t}$  ist der lokale Diskretisierungsfehler definiert als

$$\tau(\hat{t}, h) := \frac{x(\hat{t} + h) - u_1(\hat{t})}{h},$$

wobei  $u_1(\hat{t}) = x(\hat{t}) + h\phi(\hat{t}, x(\hat{t}), h, f)$  die Approximation der exakten Lösung nach einem Schritt mit  $x(\hat{t})$  als Startpunkt ist.

Falls zusätzlich für alle  $f$  mit stetiger und beschränkter Ableitung (bis zur Ordnung  $p$ ) in der  $x$ -Variable gilt, dass für alle  $\hat{t} \in (t_0, t_0 + a]$

$$\lim_{h \rightarrow 0} \tau(\hat{t}, h) = 0 \quad (\tau(\hat{t}, h) = \mathcal{O}(h^p) \text{ für } h \rightarrow 0),$$

dann nennt man das Einschrittverfahren konsistent (von der Ordnung  $p$ ).

**Definition 3.1.3 (Globaler Diskretisierungsfehler)** Mit  $\hat{t} = t_i = t_0 + ih$ ,  $i = 1, \dots, N$  ist der globale Diskretisierungsfehler definiert als

$$e(\hat{t}, h) := x(\hat{t}) - u_i,$$

wobei  $u_i$  die vom Einschrittverfahren im  $i$ -ten Schritt approximierte Lösung ist.

Falls zusätzlich für alle  $f$  mit stetiger und beschränkter Ableitung (bis zur Ordnung  $p$ ) in der  $x$ -Variable gilt, dass für alle  $\hat{t} \in (t_0, t_0 + a]$

$$\lim_{h \rightarrow 0} e(\hat{t}, h) = 0 \quad (e(\hat{t}, h) = \mathcal{O}(h^p) \text{ für } h \rightarrow 0),$$

dann nennt man das Einschrittverfahren konvergent (von der Ordnung  $p$ ).

Es ist uns möglich, eine obere Schranke für den globalen Fehler zu finden. Dazu müssen wir zuerst die diskrete Version der Gronwallsche Ungleichung in Satz 2.3.1 beweisen.

**Satz 3.1.4 (Diskrete Gronwallsche Ungleichung)** Gegeben seien die Folgen

$(\alpha)_i, (\beta)_i, (\gamma)_i \geq 0$  mit  $i = 0, \dots, N$  und es gilt

$$\gamma_{i+1} \leq (1 + \alpha_i)\gamma_i + \beta_i, \quad i = 0, \dots, N-1.$$

Dann gilt

$$\gamma_{i+1} \leq \left( \gamma_0 + \sum_{j=0}^i \beta_j \right) e^{\alpha_0 + \dots + \alpha_i}, \quad i = 0, \dots, N-1.$$

*Beweis.* Wir beweisen diesen Satz mittels Induktion nach  $i$ :

$$\begin{aligned}
 (i = 0) : \quad & \gamma_1 \leq \underbrace{(1 + \alpha_0)}_{\leq e^{\alpha_0}} \gamma_0 + \underbrace{\beta_0}_{=\beta_0 e^0 \leq \beta_0 e^{\alpha_0}} \leq (\gamma_0 + \beta_0) e^{\alpha_0}, \\
 \text{(IV) :} \quad & \gamma_i \leq \left( \gamma_0 + \sum_{j=0}^{i-1} \beta_j \right) e^{\alpha_0 + \dots + \alpha_{i-1}} \\
 (i \rightarrow i+1) : \quad & \gamma_{i+1} \leq (1 + \alpha_i) \gamma_i + \beta_i \\
 & \stackrel{\text{IV}}{\leq} (1 + \alpha_i) \left( \gamma_0 + \sum_{j=0}^{i-1} \beta_j \right) e^{\alpha_0 + \dots + \alpha_{i-1}} + \beta_i \\
 & \leq e^{\alpha_i} \left( \gamma_0 + \sum_{j=0}^{i-1} \beta_j \right) e^{\alpha_0 + \dots + \alpha_{i-1}} + \beta_i e^{\alpha_0 + \dots + \alpha_i} \\
 & \leq \left( \gamma_0 + \sum_{j=0}^i \beta_j \right) e^{\alpha_0 + \dots + \alpha_i}.
 \end{aligned}$$

□

**Satz 3.1.5** Sei das Einschrittverfahren (3.2) und das Anfangswertproblem (3.1) gegeben. Ist die Inkrementfunktion  $\phi$  in der  $x$ -Variable Lipschitz-stetig, also

$$\|\phi(t, x_1, h, f) - \phi(t, x_2, h, f)\|_2 \leq L \|x_1 - x_2\|_2$$

für alle  $(t, x_1, h), (t, x_2, h) \in D \times \mathbb{R}^n \times [0, h_0]$  mit  $h_0, L > 0$ , dann gilt für den globalen Fehler in  $t_i = t_0 + ih$  die obere Schranke

$$\|e(t_i, h)\|_2 \leq (\|e_0\|_2 + (t_i - t_0)\tau_h) e^{L(t_i - t_0)}, \quad i = 1, \dots, N.$$

Dabei ist  $e_0 = x(t_0) - u_0$  und  $\tau_h = \max_{k=1, \dots, N} \|\tau(t_k, h)\|_2$ .

*Beweis.* Wir lösen zuerst die Gleichung des lokalen Diskretisierungsfehlers nach  $x(t_{j+1})$  auf:

$$x(t_{j+1}) = x(t_j) + h\phi(t_j, x(t_j), h, f) + h\tau(t_j, h), \quad j = 0, \dots, i-1.$$

Dann setzen wir das Ergebnis in den globalen Fehler ein

$$e(t_{j+1}, h) = x(t_{j+1}) - u_{j+1} = x(t_j) - u_j + h\tau(t_j, h) + h(\phi(t_j, x(t_j), h, f) - \phi(t_j, u_j, h, f))$$

und schätzen ab

$$\|e(t_{j+1}, h)\|_2 \leq \|e(t_j, h)\|_2 + h\tau_h + hL \|e(t_j, h)\|_2 = (1 + hL) \|e(t_j, h)\|_2 + h\tau_h.$$



Nun können wir mit  $\alpha_j := hL$ ,  $\gamma_j := \|e(t_j, h)\|_2$  und  $\beta_j := h\tau_h$  die diskrete Gronwallsche Ungleichung anwenden und erhalten

$$\begin{aligned}\|e(t_{j+1}, h)\|_2 &\leq (\|e_0\|_2 + \sum_{i=0}^j h\tau_h) e^{\sum_{i=0}^j hL} \\ &= (\|e_0\|_2 + (j+1)h\tau_h) e^{(j+1)hL} \\ &= (\|e_0\|_2 + (t_{j+1} - t_0)\tau_h) e^{L(t_{j+1} - t_0)}\end{aligned}$$

für alle  $j = 0, \dots, i-1$ , also insbesondere

$$\|e(t_i, h)\|_2 = (\|e_0\|_2 + (t_i - t_0)\tau_h) e^{L(t_i - t_0)}$$

für  $j = i-1$ . □

Der Satz besagt also, dass die Konsistenz des Einschrittverfahrens bereits die Konvergenz impliziert. Allgemein ist dies nicht der Fall, wie wir in Abschnitt 3.3 sehen werden.

### 3.2 Runge-Kutta-Verfahren

Ein *Runge-Kutta-Verfahren* ist ein Einschrittverfahren höherer Ordnung. Um ein Runge-Kutta-Verfahren herleiten zu können, müssen wir zuerst definieren, was eine Quadraturformel ist. Diese Definition orientiert sich hierbei an [7, Kapitel 9].

**Definition 3.2.1** Eine Quadratur ist eine Abbildung  $Q_m : C([a, b]) \rightarrow \mathbb{R}$ , für die gilt

$$\int_a^b g(x) dx \approx Q_m(g) = (b-a) \sum_{l=1}^m \alpha_l g(s_l) \quad \text{für alle } g \in C([a, b]),$$

wobei  $\Delta = \{a = s_1, \dots, s_m = b\}$  eine Intervallunterteilung ist. Die Zahlen  $s_1, \dots, s_m$  heißen Stützstellen und  $\alpha_1, \dots, \alpha_m$  heißen Gewichte der Quadratur.

Für eine interpolatorische Quadratur gilt zusätzlich

$$\alpha_l = \int_0^1 \prod_{\substack{j=1 \\ j \neq l}}^m \frac{t - t_j}{t_i - t_j} dt, \quad t_j = \frac{s_j - a}{b - a}.$$

In unserem Fall ist  $(b-a) = t_{i+1} - t_i = h$ , da wir das Integral  $\int_{t_i}^{t_{i+1}} f(s, x(s)) ds$  mit einer Quadratur approximieren:

$$\int_{t_i}^{t_{i+1}} f(s, x(s)) ds \approx h \sum_{l=1}^m \alpha_l f(s_l, x(s_l)).$$

Die Stützstellen sind also  $s_1 = t_i$ ,  $s_l = t_i + c_l h$  für  $l = 2, \dots, m$ . Dazu approximieren wir  $f(s_l, x(s_l)) \approx k_{l,i}$  mit

$$\begin{aligned}k_{1,i} &= f(t_i, u_i) \\ k_{l,i} &= f(t_i + c_l h, u_i + h \sum_{j=1}^{l-1} a_{lj} k_{j,i}), \quad l = 2, \dots, m.\end{aligned} \tag{3.3}$$

Dies ergibt das  $m$ -stufige explizite Runge-Kutta-Verfahren

$$u_{i+1} = u_i + h \sum_{l=1}^m b_l k_{l,i}, \quad (3.4)$$

mit  $b_1, \dots, b_m \in \mathbb{R}$ . Wir werden später ein klassisches 4-stufiges Runge-Kutta-Verfahren verwenden, welches wir jetzt herleiten. Dazu benötigen wir die Simpsonregel

$$Q_3(g) = \frac{b-a}{6} \left( g(a) + 4g\left(\frac{a+b}{2}\right) + g(b) \right).$$

Es gilt also die Approximation

$$\begin{aligned} \int_{t_i}^{t_{i+1}} f(s, x(s)) \, ds &\approx \frac{h}{6} \left( f(t_i, x(t_i)) + 4f\left(t_i + \frac{h}{2}, x\left(t_i + \frac{h}{2}\right)\right) + f(t_i + h, x(t_i + h)) \right) \\ &= h \left( \frac{1}{6} f(t_i, x(t_i)) + \frac{2}{3} f\left(t_i + \frac{h}{2}, x\left(t_i + \frac{h}{2}\right)\right) + \frac{1}{6} f(t_{i+1}, x(t_{i+1})) \right). \end{aligned}$$

Damit ergibt sich durch mehrfacher Anwendung der Taylor-Entwicklung das 4-stufige klassische Runge-Kutta-Verfahren

$$u_{i+1} = u_i + h \left( \frac{1}{6} k_{1,i} + \frac{1}{3} k_{2,i} + \frac{1}{3} k_{3,i} + \frac{1}{6} k_{4,i} \right)$$

mit den Stufen

$$\begin{aligned} k_{1,i} &= f(t_i, u_i), \\ k_{2,i} &= f\left(t_i + \frac{h}{2}, u_i + \frac{h}{2} k_{1,i}\right), \\ k_{3,i} &= f\left(t_i + \frac{h}{2}, u_i + \frac{h}{2} k_{2,i}\right), \\ k_{4,i} &= f\left(t_i + h, u_i + h k_{3,i}\right). \end{aligned}$$

Wir haben also die Koeffizienten

$$b := (b_1, \dots, b_m)^\top = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{6} \end{pmatrix}, \quad c := (c_1, \dots, c_m)^\top = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 1 \end{pmatrix}, \quad A := [a_{lj}]_{l,j=1}^{m,m} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Im Allgemeinen gilt für explizite Runge-Kutta-Verfahren, dass  $a_{lj} = 0$  für  $l \leq j$ .

**Bemerkung 3.2.2** Die Koeffizienten charakterisieren ein Runge-Kutta-Verfahren, weshalb sie für üblich in sogenannten Butcher-Tabellen zusammengefasst werden:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

Da Runge-Kutta-Verfahren Einschrittverfahren sind, gibt es auch implizite Runge-Kutta-Verfahren mit der Form

$$\begin{aligned} k_{1,i} &= f(t_i + c_1 h, u_i + h(a_{11}k_{1,i} + \cdots + a_{1m}k_{m,i})) \\ &\vdots \\ k_{m,i} &= f(t_i + c_m h, u_i + h(a_{m1}k_{1,i} + \cdots + a_{mm}k_{m,i})) \\ u_{i+1} &= u_i + h(b_1k_{1,i} + \cdots + b_mk_{m,i}) \end{aligned}$$

Hier kann  $A$  im Gegensatz zu expliziten Runge-Kutta-Verfahren eine voll besetzte Matrix sein.

### 3.2.1 Konsistenz

Es wurde bereits gezeigt, dass die Konvergenz eines Einschrittverfahrens aus der Konsistenz folgt. Deshalb genügt es sich die Konsistenz(ordnung) der Runge-Kutta-Verfahren zu betrachten. Dabei spielt die Wahl der Koeffizienten eine große Rolle. Ziel ist es, die Koeffizienten so zu wählen, dass die Konsistenzordnung des Runge-Kutta-Verfahrens möglichst hoch ist. Dazu zeigen wir zuerst, dass unter bestimmten Voraussetzungen für die Koeffizienten ein explizites Runge-Kutta-Verfahren invariant gegenüber Autonomisierung sind. Das bedeutet, dass ein Runge-Kutta-Verfahren genau dann ein Anfangswertproblem (3.1) mit der Näherungslösung  $u_i$  approximiert, wenn es das autonome Anfangswertproblem

$$\begin{aligned} z' &= 1, & z(t_0) &= t_0, \\ x' &= f(z, x), & x(t_0) &= x_0 \end{aligned} \tag{3.5}$$

mit der Näherungslösung  $v_i := \begin{bmatrix} t_i \\ u_i \end{bmatrix}$  approximiert wird.

**Satz 3.2.3** *Gegeben sei ein explizites Runge-Kutta-Verfahren (3.4). Dies ist genau dann invariant gegenüber Autonomisierung, wenn für die Koeffizienten gilt:*

$$\sum_{j=1}^m b_j = 1 \quad \text{und} \quad c_l = \sum_{j=1}^{l-1} a_{lj}, \quad l = 1, \dots, m.$$

*Beweis.* Für ein autonomes Anfangswertproblem (3.5) hat das Runge-Kutta-Verfahren die Form

$$\hat{k}_{l,i} = \begin{bmatrix} \hat{k}_{l,i}^z \\ \hat{k}_{l,i}^x \end{bmatrix} = \begin{bmatrix} 1 \\ f\left(t_i + h \sum_{j=1}^{l-1} a_{lj} \cdot \hat{k}_{j,i}^z, u_i + h \sum_{j=1}^{l-1} a_{lj} \hat{k}_{j,i}^x\right) \end{bmatrix}, \quad l = 1, \dots, m,$$

$$v_{i+1} = v_i + h \sum_{l=1}^m b_l \hat{k}_{l,i} = \begin{bmatrix} t_i + h \sum_{l=1}^m b_l \cdot 1 \\ u_i + h \sum_{l=1}^m b_l \hat{k}_{l,i}^x \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} t_{i+1} \\ u_{i+1} \end{bmatrix}.$$

Es gilt also genau dann  $t_i + h \sum_{l=1}^m b_l \cdot 1 = t_{i+1}$ , wenn  $\sum_{l=1}^m b_l = 1$ . Für die zweite Komponente gilt genau dann  $u_i + h \sum_{l=1}^m b_l \hat{k}_{l,i}^x = u_{i+1}$ , wenn  $\hat{k}_{l,i}^x = k_{l,i}$ ,  $l = 1, \dots, m$ . Äquivalent dazu muss gelten

$$f\left(t_i + h \sum_{j=1}^{l-1} a_{lj} \cdot 1, u_i + h \sum_{j=1}^{l-1} a_{lj} \hat{k}_{j,i}^x\right) = f\left(t_i + h c_l, u_i + h \sum_{j=1}^{l-1} a_{lj} k_{j,i}\right), \quad l = 1, \dots, m,$$

was genau dann erfüllt ist, wenn  $c_l = \sum_{o=1}^{l-1} a_{lo}$ ,  $l = 1, \dots, m$ . □

Im Folgenden werden wir uns also auf autonome Anfangswertprobleme der Form

$$x' = f(x), \quad x(t_0) = x_0 \tag{3.6}$$

beschränken. Unser Ziel ist es, Voraussetzungen an die Koeffizienten des Runge-Kutta-Verfahrens zu setzen, so dass wir eine größtmögliche Konsistenzordnung erhalten. Dazu betrachten wir den lokalen Diskretisierungsfehler

$$\begin{aligned} \tau(t_i, h) &= \frac{x(t_i + h) - x(t_i)}{h} - \phi(t_i, x(t_i), h, f) \\ &= \frac{x(t_i + h) - x(t_i)}{h} - \sum_{l=1}^m b_l k_{l,i}. \end{aligned} \tag{3.7}$$

Wenden wir nun die Taylorentwicklung auf die exakte Lösung  $x$  in  $t_{i+1}$  für das Anfangswertproblem (3.6) an, erhalten wir

$$\begin{aligned} x(t_{i+1}) &= x(t_i) + hx'(t_i) + \frac{1}{2}h^2x''(t_i) + \mathcal{O}(h^3) \\ &= x(t_i) + hf(x(t_i)) + \frac{1}{2}h^2Df(x(t_i))f(x(t_i)) + \mathcal{O}(h^3), \end{aligned}$$

wobei für die Ableitungen der Lösung  $x$  die Kettenregel benutzt wird

$$\begin{aligned} x'(t_i) &= f(x(t_i)), \\ x''(t_i) &= Df(x(t_i))x'(t_i) = Df(x(t_i))f(x(t_i)) \end{aligned}$$

und  $Df$  die Jacobi-Matrix von  $f$  ist. Da wir uns nur auf das autonome Anfangswertproblem (3.6) beschränken, hat das explizite Runge-Kutta-Verfahren die Stufen

$$\begin{aligned} k_{i,1} &= f(x(t_i)), \\ k_{i,2} &= f(x(t_i) + ha_{21}k_{1,i}), \\ &\vdots \\ k_{i,m} &= f(x(t_i) + ha_{m1}k_{1,i} + \cdots + ha_{m,m-1}k_{m-1,i}). \end{aligned}$$

Nun wenden wir ähnlich wie oben die Taylorentwicklung im Punkt  $x(t_i)$  an und erhalten

$$\begin{aligned} k_{i,1} &= f(x(t_i)), \\ k_{i,2} &= f(x(t_i)) + ha_{21}Df(x(t_i))k_{1,i} + \mathcal{O}(h^2), \\ &\vdots \\ k_{i,m} &= f(x(t_i)) + ha_{m1}Df(x(t_i))k_{1,i} + \cdots + ha_{m,m-1}Df(x(t_i))k_{m-1,i} + \mathcal{O}(h^2). \end{aligned}$$

Im nächsten Schritt setzen wir die Stufen sukzessive ein und verwenden

$$hk_{j,i} = hf(x(t_i)) + h^2a_{j-1,l}Df(x(t_i))f(x(t_i)) + \mathcal{O}(h^2) = hf(x(t_i)) + \mathcal{O}(h^2).$$

Die resultierenden Stufen haben die Form

$$\begin{aligned} k_{i,1} &= f(x(t_i)), \\ k_{i,2} &= f(x(t_i)) + ha_{21}Df(x(t_i))f(x(t_i)) + \mathcal{O}(h^2), \\ &\vdots \\ k_{i,m} &= f(x(t_i)) + ha_{m1}Df(x(t_i))f(x(t_i)) + \cdots + ha_{m,m-1}Df(x(t_i))f(x(t_i)) + \mathcal{O}(h^2). \end{aligned}$$

Einsetzen in (3.7) ergibt

$$\begin{aligned} \tau(t_i, h) &= f(x(t_i)) + \frac{1}{2}hD(f(x(t_i)))f(x(t_i)) + \mathcal{O}(h^2) \\ &\quad - \sum_{l=1}^m b_l \left( f(x(t_i)) + h \sum_{j=1}^{l-1} a_{lj}D(f(x(t_i)))f(x(t_i)) + \mathcal{O}(h^2) \right) \\ &= \left( 1 - \sum_{l=1}^m b_l \right) f(x(t_i)) + h \left( \frac{1}{2} - \sum_{l=1}^m \sum_{j=1}^{l-1} b_l a_{lj} \right) Df(x(t_i))f(x(t_i)) + \mathcal{O}(h^2). \end{aligned}$$

Da  $\sum_{l=1}^m b_l = 1$  gilt, kann schnell festgestellt werden, dass für gegenüber Autonomisierung invariante Runge-Kutta-Verfahren die Konsistenzordnung  $p = 1$  gilt. Außerdem gelten weitere Voraussetzungen für höhere Konsistenzordnungen, wie beispielsweise für  $p = 2$ :

$$\sum_{l=1}^m \sum_{j=1}^{l-1} b_l a_{lj} = \sum_{l=1}^m b_l c_l = \frac{1}{2}.$$

Allgemein lassen sich für höhere Konsistenzordnungen weitere Voraussetzung durch Taylorentwicklungen mit höheren  $h$ -Potenzen finden.

**Satz 3.2.4 (Butchersranken)** Für die maximale erreichbare Konsistenzordnung  $p$  eines ex-

pliziten Runge-Kutta-Verfahrens mit  $m$  Stufen gilt

$$p \leq m.$$

*Beweis.* [ernsthairergerhardwannerSolvingOrdinaryDifferential]

Außerdem kann man in [8] folgende Zusammenhänge zwischen der Stufe  $m$  des Runge-Kutta-Verfahrens (3.4) und dessen Konsistenzordnung finden:

Stufe $m$	1	2	3	4	5	6	7	8	9	$\geq 9$
Ordnung $p$	1	2	3	4	4	5	6	6	7	$\leq m - 2$

### 3.2.2 Schrittweitensteuerung

Wir haben im letzten Abschnitt gesehen, unter welchen Bedingungen die Konvergenz der Runge-Kutta-Verfahren gewährleistet ist. Unser Ziel ist es nun, die bisher konstante Schrittweite  $h$  so zu wählen, dass der Rechenaufwand zur Approximation der Lösung so gering wie möglich gehalten wird. Dazu treten es im Grunde zwei Fälle auf.

Ist der lokale Diskretisierungsfehler größer als eine gegebene Toleranz, so sollten wir die Schrittweite verkleinern, um den Fehler zu verringern. Ein Beispiel hierfür wäre eine oszillierende Lösung, bei der eine zu große Schrittweite zu hohem lokalen Fehler führen würde, da das Einschrittverfahren Wendungspunkte überspringen würde. Ist der lokale Diskretisierungsfehler aber klein in Relation zur Toleranz, dann können wir die Schrittweite vergrößern, um den Rechenaufwand zu minimieren. Dies tritt beispielsweise bei nahezu linearen Lösungsfunktionen auf.

Also müssen wir uns zuerst überlegen, wie wir den lokalen Diskretisierungsfehler sinnvoll mit minimalen Rechenaufwand abschätzen, ohne die tatsächliche Lösung der Differentialgleichung zu kennen. Dazu verwenden wir sogenannte *eingebettete Runge-Kutta-Verfahren*, welche aus zwei Einschrittverfahren

$$u_{i+1}^{(1)} = u_i + h_i \phi_p(t_i, u_i, h_i, f)$$

und

$$u_{i+1}^{(2)} = u_i + h_i \phi_{p+1}(t_i, u_i, h_i, f)$$

mit den Konsistenzordnungen  $p$  und  $p + 1$  bestehen. Die Butcherschranken implizieren, dass die Anzahl der Stufen schneller steigt, als die Konsistenzordnung, weshalb es nur bis zu einem gewissen Grad sinnvoll ist, ein Runge-Kutta-Verfahren höherer Ordnung für  $u_i^{(2)}$  zu verwenden. Deshalb beschränken wir uns auf Verfahren mit Konsistenzordnung  $p + 1$ . Das Verfahren nennt man *eingebettet*, wenn die für  $u_i^{(1)}$  berechneten Stufen bei der Berechnung von  $u_i^{(2)}$  wieder verwendet werden. Dabei werden Funktionsauswertungen gespart, was den Rechenaufwand und die Laufzeit reduziert. Die dazugehörige Butcher-Tabelle wird dementsprechend zu einer *kombinierten* Butcher-Tabelle erweitert

$c$	$A$
	$(b^{(1)})^\top$
	$(b^{(2)})^\top$

Dabei ist  $b^{(1)}$  der Koeffizientenvektor für von  $u_{i+1}^{(1)}$  und  $b^{(2)}$  Koeffizientenvektor für  $u_{i+1}^{(2)}$ .

Wir definieren

$$\begin{aligned}\Delta_p &:= x(t_i + h_i) - u_{i+1}^{(1)} = x(t_i + h_i) - u_i - h_i \Phi_p(t_i, u_i, h_i, f) = h_i \tau^{(1)}(t_i, h_i) = \mathcal{O}(h_i^{p+1}), \\ \Delta_{p+1} &:= x(t_i + h_i) - u_{i+1}^{(2)} = x(t_i + h_i) - u_i - h_i \Phi_{p+1}(t_i, u_i, h_i, f) = h_i \tau^{(2)}(t_i, h_i) = \mathcal{O}(h_i^{p+2}).\end{aligned}$$

Daraus erhalten wir eine Abschätzung des lokalen Diskretisierungsfehlers mit

$$\begin{aligned}h_i \tau_p &= \Delta_p = \Delta_{p+1} + h_i (\Phi_{p+1}(t_i, u_i, h_i, f) - \Phi_p(t_i, u_i, h_i, f)) \\ &\approx h_i (\Phi_{p+1}(t_i, u_i, h_i, f) - \Phi_p(t_i, u_i, h_i, f)),\end{aligned}$$

da  $\Delta_{p+1} \approx 0$  für kleine  $h_i$  gilt. Also erhalten wird mit der alten Schrittweite  $h_{\text{alt}}$

$$\Delta_{p,\text{alt}} \approx h_i (\Phi_{p+1}(t_i, u_i, h_i, f) - \Phi_p(t_i, u_i, h_i, f)) = \|u_i^{(2)} - u_i^{(1)}\|_2.$$

Da wir nun den lokalen Fehler abgeschätzt haben, können wir uns darum kümmern, die Schrittweite anzupassen. Sei die Toleranz mit  $\epsilon > 0$  gegeben. Bei zu großem Fehler, also  $\Delta_{p,\text{alt}} > \epsilon$ , wählen wir eine neue Schrittweite  $h_{\text{neu}} < h_{\text{alt}}$  und wiederholen den aktuellen Schritt. Unser Ziel dabei ist es, dass der neue Fehler  $\Delta_{p,\text{neu}}$  etwas kleiner ist als  $\epsilon$ , was mit der Approximation  $\Delta_{p,\text{neu}} \approx \rho \epsilon$  mit  $0 < \rho < 1$  gesichert ist. Da  $\Delta_p = \mathcal{O}(h_i^{p+1}) \approx c h_i^{p+1}$  für jede Schrittweite gilt, kann man folgern

$$\frac{\rho \epsilon}{h_{\text{neu}}^{p+1}} \approx \frac{\Delta_{p,\text{neu}}}{h_{\text{neu}}^{p+1}} \approx c \approx \frac{\Delta_{p,\text{alt}}}{h_{\text{alt}}^{p+1}} \Rightarrow h_{\text{neu}} \approx h_{\text{alt}} \sqrt[p+1]{\frac{\rho \epsilon}{\Delta_{p,\text{alt}}}}.$$

Im anderen Fall ist der Fehler zu klein, also  $\Delta_{p,\text{alt}} \approx 0$ . Dann können wir die Schrittweite mit Hilfe einer Hochschaltungsbeschränkung anpassen. Dabei setzen wir  $h_{\text{neu}} = \min(q h_{\text{alt}}, h_{\text{max}})$ , wobei  $h_{\text{max}}$  die maximale Schrittweite und  $q > 1$  der Hochschaltfaktor ist. Wird der Schritt akzeptiert, so setzen wir  $u_{i+1} = u_{i+1}^{(2)}$  und führen die Approximation mit der neuen Schrittweite fort.

Wir werden in dieser Arbeit hauptsächlich das *4-5-Runge-Kutta-Verfahren* (oder *Dormand-Prince-Verfahren*) mit der kombinierten Butcher-Tabelle

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{3}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

verwenden.

### 3.3 Mehrrschrittverfahren

Durch die Butchersranken wissen wir, dass ab einer Konsistenzordnung  $p$  eines Runge-Kutta-Verfahrens immer mehr Stufen berechnet werden müssen, was wiederum eine erhöhte Anzahl an Funktionsauswertungen bedeutet. Dies wollen wir, wenn möglich, verhindern. Deshalb führen wir sogenannte *Mehrschrittverfahren* ein, welche im Gegensatz zu Einschrittverfahren die Information aus bereits berechneten Näherungen nutzen. Mehrrschrittverfahren haben also den Vorteil, dass (in expliziten Verfahren) pro Schritt nur eine zusätzliche Funktionsauswertung benötigen.

**Definition 3.3.1** Sei  $D = [t_0, t_0 + a]$  ein Zeitintervall und  $D_h = \{t_i = t_0 + ih \text{ für } i = 0, \dots, N\}$  eine Zerlegung von  $D$

$$t_0 < t_1 < \dots < t_N = t_0 + a$$

mit der Schrittweite  $h = \frac{a}{N}$ . Ein  $k$ -Schrittverfahren für das Anfangswertproblem (3.1) hat die Form

$$\begin{aligned} &\text{Startwerte } u_0, \dots, u_{k-1} \in \mathbb{R}^n, \\ &\sum_{j=0}^k \alpha_j u_{i+j} = h \phi(t_i, u_i, \dots, u_{i+k}, h, f), \quad i = 0, \dots, N-k, \end{aligned} \quad (3.8)$$

wobei  $\alpha_k \neq 0$ .

Falls  $\phi$  unabhängig von  $u_{i+k}$  ist, kann man (3.8) nach  $u_{i+k}$  auflösen und erhält ein explizites  $k$ -Schrittverfahren. Im impliziten Fall muss man ein (im Allgemeinen nichtlineares) Gleichungssystem lösen, um  $u_{i+k}$  bestimmen zu können.

Die Startwerte  $u_0, \dots, u_{k-1}$  können beispielsweise durch einmalige Verwendung von Einschrittverfahren ermittelt werden.

**Definition 3.3.2** Das  $k$ -Schrittverfahren (3.8) heißt linear, falls

$$\sum_{j=0}^k \alpha_j u_{i+j} = h \sum_{j=0}^k \beta_j f(t_{i+j}, u_{i+j}), \quad i = 0, \dots, N-k, \quad (3.9)$$

gilt. Zudem heißt für  $\beta_k = 0$  das Verfahren explizit, sonst implizit.

Zur Herleitung eines linearen Mehrrschrittverfahrens integrieren wir die Differentialgleichung (3.1) und erhalten

$$x(t_{i+k}) - x(t_{i+r-l}) = \int_{t_{i+r-l}}^{t_{i+k}} f(s, x(s)) \, ds, \quad 0 < r-l < k, \quad r, l \in \mathbb{N}.$$

Nun approximieren wir den Integranden  $f(s, x(s))$  mit dem Interpolationspolynom  $p \in \mathbb{R}_r[s]$ , wobei  $p(t_{i+j}) = f(t_{i+j}, x(t_{i+j}))$  für  $j = 0, \dots, r$  gilt. Durch die Lagrange-Interpolationsformel [7, Kapitel 7] und  $t_{i+k} = t_i + kh$  hat  $p$  die Form

$$p(s) = \sum_{j=0}^r f(t_{i+j}, x(t_{i+j})) \prod_{\substack{q=0 \\ q \neq j}}^r \frac{s - t_{i+q}}{t_{i+j} - t_{i+q}} = \sum_{j=0}^r f(t_{i+j}, x(t_{i+j})) \prod_{\substack{q=0 \\ q \neq j}}^r \frac{s - (t_i + qh)}{(j+q)h}.$$



Mit dieser Interpolation erhalten wir

$$\begin{aligned}
 x(t_{i+k}) - x(t_{i+r-l}) &\approx \int_{t_{i+r-l}}^{t_{i+k}} p(s) \, ds \\
 &= \sum_{j=0}^r f(t_{i+j}, x(t_{i+j})) \int_{t_{i+r-l}}^{t_{i+k}} \prod_{\substack{q=0 \\ q \neq j}}^r \frac{s - (t_i + qh)}{(j+q)h} \, ds \\
 &= h \sum_{j=0}^r f(t_{i+j}, x(t_{i+j})) \int_{-l}^{k-r} \prod_{\substack{q=0 \\ q \neq j}}^r \frac{r-q+x}{j+q} \, dx \\
 &= h \sum_{j=0}^r \beta_j^{(r,l)} f(t_{i+j}, u_{i+j}),
 \end{aligned}$$

wobei

$$\beta_j^{(r,l)} := \int_{-l}^{k-r} \prod_{\substack{q=0 \\ q \neq j}}^r \frac{(r-q+x)}{(j-q)} \, dx.$$

Im vorletzten Schritt wurde die Substitution  $s - t_i - rh = xh$  benutzt. Nun verwenden wir die Näherungen  $u_{i+j}$  und erhalten somit das *lineare  $k$ -Schrittverfahren vom Typ  $(r, l)$* .

$$u_{i+k} - u_{i+r-l} = h \sum_{j=0}^r \beta_j^{(r,l)} f(t_{i+j}, u_{i+j}), \quad (3.10)$$

Ähnlich wie für die Einschrittverfahren definieren wir nun zuerst den *lokalen Diskretisierungsfehler* der Mehrschrittverfahren.

**Definition 3.3.3** Sei ein Mehrschrittverfahren (3.8) und das dazugehörige Anfangswertproblem (3.1) gegeben. Dann ist der lokale Diskretisierungsfehler gegeben durch

$$\tau(t_i, h) = \frac{1}{h} \sum_{j=0}^k \alpha_j x(t_i + jh) - \phi(t_i, x(t_i), x(t_i + h), \dots, x(t_i + kh), h, f).$$

Das Mehrschrittverfahren (3.8) heißt konsistent von der Ordnung  $p \geq 1$ , wenn für alle  $f$  mit stetiger und beschränkter Ableitung in der  $x$ -Variable bis zur Ordnung  $p$  gilt, dass  $\tau(t_i, h) = \mathcal{O}(h^p)$ .

Wir können unter bestimmten Bedingungen die Konsistenzordnungen der linearen Mehrschrittverfahren vom Typ  $(r, l)$  (3.10) explizit angeben.

**Satz 3.3.4** Unter den Bedingungen, dass  $f$   $(r+1)$ -mal stetig differenzierbar auf  $D \times K$ , wobei  $K \subseteq \mathbb{R}^n$  ist, hat das lineare Mehrschrittverfahren vom Typ  $(r, l)$  (3.10) die Konsistenzordnung  $p = r + 1$ .

*Beweis.* Da  $f$   $(r+1)$ -mal stetig differenzierbar auf  $D \times K$  ist, existiert nach Bemerkung 2.2.6 und Picard-Lindelöf 2.2.5 eine eindeutige Lösung (3.1), welche  $(r+2)$ -mal stetig differenzierbar ist.

Für den lokalen Diskretisierungsfehler der linearen Mehrschrittverfahren vom Typ  $(r, l)$  (3.10) gilt

$$\begin{aligned}
\tau(t_i, h) &= \frac{1}{h} (x(t_i + kh) - x(t_i + (r-l)h)) - \sum_{j=0}^r \beta_j^{(r,l)} f(t_i + jh, x(t_i + jh)) \\
&= \frac{1}{h} \int_{t_i+r-l}^{t_i+k} x'(s) ds - \frac{1}{h} \int_{t_i+r-l}^{t_i+l} p(s) ds \\
&= \frac{1}{h} \int_{t_i+r-l}^{t_i+l} (f(s, x(s)) - p(s)) ds \\
&= \frac{1}{h} \int_{t_i+r-l}^{t_i+l} \prod_{j=0}^r (s - (t_i + jh)) \frac{f^{(r+1)}(\xi(s), x(\xi(s)))}{(r+1)!} ds \\
&= \frac{1}{h} \int_{t_i+r-l}^{t_i+l} \prod_{j=0}^r (s - (t_i + jh)) \frac{x^{(r+2)}(\xi(s))}{(r+1)!} ds,
\end{aligned}$$

wobei  $\xi(s) \in [t_{i+r-l}, t_{i+k}]$  und die vorletzte Gleichung resultiert aus der Fehlerdarstellung der Polynominterpolation, vgl [7, S. 213]. Substituieren wir nun  $s = t_i + rh + xh$ , so folgt daraus  $s - (t_i + jh) = xh - (j-r)h$  und

$$\begin{aligned}
\prod_{j=0}^r (s - (t_i + jh)) &= h^{r+1} \prod_{j=0}^r (x - j + r) = h^{r+1} (x+r)(x+r-1) \dots (x+r-r) \\
&= h^{r+1} (-1)^{r+1} (-x) \dots (-x-r) = h^{r+1} (-1)^{r+1} \frac{(-x)!}{(-x-r-1)!} \\
&= h^{r+1} (-1)^{r+1} (r+1)! \frac{(-x)!}{(-x-r-1)! (r+1)!} \\
&= h^{r+1} (-1)^{r+1} (r+1)! \binom{-x}{r+1},
\end{aligned}$$

wobei im letzten Schritt die Definition des Binomialkoeffizienten

$$\binom{n}{k} := \frac{n!}{(n-k)! k!}$$

mit  $n, k \in \mathbb{Z}$ ,  $n \geq k \geq 0$  verwendet wurde. Einsetzen in den lokalen Diskretisierungsfehler ergibt

$$\tau(t_i, h) = (-1)^{r+1} h^{r+1} \int_{-l}^{k-r} \binom{-x}{r+1} x^{(r+2)}(\xi(t_i + rh + xh)) dx$$

und dessen Abschätzung

$$\|\tau(t_i, h)\|_2 \leq h^{r+1} \underbrace{\left\| x^{(r+2)} \right\|_\infty \int_{-l}^{k-r} \binom{-x}{r+1} dx}_{=:c} = h^{r+1} c,$$

wobei  $c$  eine von  $h$  unabhängige Konstante ist. Daraus folgt, dass  $\tau(t_i, h) = \mathcal{O}(h^{r+1})$  ist.  $\square$

Um die Konsistenz allgemeiner linearer  $k$ -Schrittverfahren (3.9) angeben zu können, benötigen wir noch die Definitionen für die ersten und zweiten charakteristische Polynome.

**Definition 3.3.5** Für ein lineares  $k$ -Schrittverfahren (3.9) sind die Polynome

$$\rho(z) := \sum_{j=0}^k \alpha_j z^j \quad \text{und} \quad \sigma(z) := \sum_{j=0}^k \beta_j z^j$$

definiert. Das Polynom  $\rho$  nennt man das erste charakteristische Polynom und  $\sigma$  das zweite charakteristische Polynom des gegebenen Verfahrens.

Damit können wir nun folgende Aussage über die Konsistenzordnung der linearen Mehrschrittverfahren beweisen.

**Satz 3.3.6** Für das lineare  $k$ -Schrittverfahren (3.9) sind folgende Aussagen äquivalent:

1. (3.9) ist konsistent von der Ordnung  $p$ .
2. Für das Anfangswertproblem  $x' = x$ ,  $x(0) = 1$  gilt  $\tau(t_i, h) = \mathcal{O}(h^p)$ .
3.  $\frac{\rho(z)}{\ln(z)} - \sigma(z)$  hat eine  $p$ -fache Nullstelle in  $z = 1$ , d.h.  $\frac{\rho(z)}{\ln(z)} - \sigma(z) = \mathcal{O}((z-1)^p)$  für  $z \rightarrow 0$ .
4.  $\sum_{j=0}^k \alpha_j = 0$  zusammen mit  $\sum_{j=0}^k j^m \alpha_j = \sum_{j=0}^k m j^{m-1} \beta_j$ , für  $m = 0, 1, \dots, p$ .

Insbesondere ist für eine stetig differenzierbare rechte Seite  $f$  das lineare  $k$ -Schrittverfahren (3.9) konsistent, falls

$$\rho(1) = 0 \quad \text{und} \quad \rho'(1) = \sigma(1).$$

*Beweis.* Wir beginnen mit 1.  $\Rightarrow$  2.: Wegen 1. ist das Verfahren schon für allgemeine gewöhnliche Differentialgleichungen erster Ordnung konsistent, also auch für den speziellen Fall in 2.. Somit gilt 2..

2.  $\Rightarrow$  3.: Wir können die Lösung des Anfangsproblems in 2. explizit berechnen:  $x(t) = e^t$ . Jetzt können wir den lokalen Diskretisierungsfehler angeben

$$\tau(t_i, h) = \frac{1}{h} \sum_{j=0}^k \alpha_j e^{t_i+jh} - \sum_{j=0}^k \beta_j e^{t_i+jh} = e^{t_i} \left( \frac{1}{h} \rho(e^h) - \sigma(e^h) \right).$$

Es gilt  $\tau(t_i, h) = \mathcal{O}(h^p)$  genau dann, wenn  $\frac{1}{h} \rho(e^h) - \sigma(e^h) = \mathcal{O}(h^p)$ . Betrachte die Taylor-Entwicklung der Funktion  $\ln(z)$  mit der Entwicklungsstelle  $z_0 = 1$ . Dann ergibt sich

$$\ln z = \ln(1 + (z-1)) = \ln(1) + \mathcal{O}(z-1) = \mathcal{O}(z-1).$$

Setzen wir nun  $z = e^h$  oder  $h = \ln(z)$  und es folgt

$$\frac{\rho(z)}{\ln(z)} - \sigma(z) = \mathcal{O}(h^p) = \mathcal{O}((\ln(z))^p) = \mathcal{O}((z-1)^p).$$

3.  $\Rightarrow$  4.: Setzen wir  $z = e^h$  folgt

$$\frac{1}{h}\rho(e^h) - \sigma(e^h) = \mathcal{O}(h^p).$$

Durch Multiplikation mit der Schrittweite  $h$  erhalten wir

$$\mathcal{O}(h^{p+1}) = \rho(e^h) - h\sigma(e^h) = \sum_{j=0}^k \alpha_j e^{jh} - h \sum_{j=0}^k \beta_j e^{jh}.$$

Für die Taylor-Entwicklung von der Funktion  $e^{jh}$  um 0 bis zur p-ten Ordnung gilt

$$e^{jh} = \sum_{m=0}^p \frac{j^m}{m!} h^m + \mathcal{O}(h^{p+1}) = 1 + \sum_{m=1}^p \frac{j^m}{m!} h^m + \mathcal{O}(h^{p+1}).$$

Einsetzen in die obige Gleichung ergibt

$$\begin{aligned} \mathcal{O}(h^{p+1}) &= \sum_{j=0}^k \alpha_j \left( 1 + \sum_{m=1}^p \frac{j^m}{m!} h^m \right) - h \sum_{j=0}^k \beta_j \left( 1 + \sum_{m=1}^p \frac{j^m}{m!} h^m \right) \\ &= \sum_{j=0}^k \alpha_j + \sum_{m=1}^p h^m \sum_{j=0}^k \left( \frac{j^m}{m!} \alpha_j - \frac{j^{m-1}}{(m-1)!} \beta_j \right) + \mathcal{O}(h^{p+1}). \end{aligned}$$

Wir können die Summationsreihenfolge tauschen, da beide Summen endlich sind. Damit die Gleichung erfüllt ist, muss gelten:

$$\sum_{j=0}^k \alpha_j = 0, \quad \sum_{j=0}^k \left( \frac{j^m}{m!} \alpha_j - \frac{j^{m-1}}{(m-1)!} \beta_j \right) = 0 \quad \text{für } m = 1, \dots, p.$$

Somit gilt 4..

4.  $\Rightarrow$  1.: Wir betrachten die Taylor-Entwicklungen

$$\begin{aligned} x(t_i + jh) &= \sum_{m=0}^p \frac{(jh)^m}{m!} x^{(m)}(t_i) + x^{(p+1)}(\xi) \frac{(jh)^{p+1}}{(p+1)!}, \quad \xi \in [t_i, t_i + jh], \\ x'(t_i + jh) &= \sum_{m=0}^{p-1} \frac{(jh)^m}{m!} x^{(m+1)}(t_i) + x^{(p+1)}(\eta) \frac{(jh)^{p+1}}{(p+1)!}, \quad \eta \in [t_i, t_i + jh]. \end{aligned}$$

Für den lokalen Diskretisierungsfehler gilt dann

$$\begin{aligned} \tau(t_i, h) &= \frac{1}{h} \sum_{j=0}^k \alpha_j x(t_i + jh) - \sum_{j=0}^k \beta_j f(t_i + jh, x(t_i + jh)) \\ &= \sum_{j=0}^k \left( \frac{1}{h} \alpha_j \left( \sum_{m=0}^p \frac{(jh)^m}{m!} x^{(m)}(t_i) \right) - \beta_j \left( \sum_{m=1}^p \frac{(jh)^{m-1}}{(m-1)!} x^{(m)}(t_i) \right) \right) + \mathcal{O}(h^p) \\ &= x(t_i) \frac{1}{h} \sum_{j=0}^k \alpha_j + \sum_{m=1}^p h^{m-1} x^{(m)}(t_i) \sum_{j=0}^k \left( \frac{j^m}{m!} \alpha_j - \frac{j^{m-1}}{(m-1)!} \beta_j \right) + \mathcal{O}(h^p) \\ &= \mathcal{O}(h^p), \end{aligned}$$

wobei  $f(t_i + jh, x(t_i + jh)) = x'(t_i + jh)$  und im letzten Schritt wurden die Eigenschaften von 4. genutzt.

Mit  $\rho(1) = 0$  und  $\rho'(1) = \sigma(1)$  gilt

$$\sum_{j=0}^k \alpha_j = 0 \quad \text{und} \quad \sum_{j=0}^k j\alpha_j = \sum_{j=0}^k \beta_j$$

Also ist 4. für  $m = 1 = p$  erfüllt und daraus folgt mit 1. die Konsistenz (von Ordnung  $p = 1$ ).  $\square$

Mit folgendem Satz können wir die *Konvergenz* eines Mehrschrittverfahrens erzielen, wobei hierfür der die Theorie der *Stabilität* eines Verfahrens benötigt wird, welche in [ernsthairergerhardwannerSolvingOrdinaryDifferentialEquations] nachgelesen werden kann.

**Satz 3.3.7** *Das Mehrschrittverfahren (3.8) ist konvergent (von der Ordnung  $p$ ), also*

$$\|x(t_i) - u_i\|_2 = \mathcal{O}(h^p), \quad \text{für } t_i = t_0 + ih \in [t_0, t_0 + a],$$

wenn es stabil und konsistent ist.

*Beweis.* [ernsthairergerhardwannerSolvingOrdinaryDifferentialEquations]

### 3.4 Verfahren für steife Differentialgleichungen

Durch Betrachtung steifer Differentialgleichungen, siehe Definition 2.1.6, erkennen wir, dass die Verwendung allgemeiner Ein- oder Mehrschrittverfahren nicht sinnvoll ist. Wir werden später dazu ein Beispiel in Abschnitt 5.2 betrachten. Deshalb leiten wir nun geeignete Verfahren her, welche *Rückwärtsdifferenzenverfahren* (engl: backward differentiation formula oder BDF) genannt werden. Zur Herleitung eines BDF-Verfahrens interpolieren wir zuerst die Lösung  $x(t)$  von (3.1) durch ein Polynom  $p \in \mathbb{R}_k[t]$  mit  $p(t_{i+j}) = x(t_{i+j})$  für  $j = 0, \dots, k$ . Unter Verwendung der Lagrange-Interpolationsformel

$$p(t) = \sum_{l=0}^k x(t_{i+l}) \mathcal{L}_{i+l}(t) \quad \text{mit} \quad \mathcal{L}_{i+l}(t) = \prod_{\substack{l=0 \\ l \neq j}}^k \frac{t - t_{i+l}}{t_{i+j} - t_{i+l}}$$

erhalten wir die Approximation

$$f(t_{i+k}, x(t_{i+k})) = x'(t_{i+k}) \approx p'(t_{i+k}) = \sum_{l=0}^k x(t_{i+l}) \mathcal{L}'_{i+l}(t_{i+k}).$$

Durch Ersetzen der unbekannten Werte  $x(t_{i+j})$  durch Näherungen  $u_{i+j}$  resultiert das BDF-Verfahren

$$\sum_{j=0}^k \alpha_j u_{i+j} = hf(t_{i+k}, u_{i+k})$$

mit

$$\alpha_j = h \mathcal{L}'_{i+j}(t_{i+k}), \quad j = 0, \dots, k,$$

welches ein implizites, lineares  $k$ -Schrittverfahren ist.

## 4 Neuronale Netze

In diesem Kapitel werden wir eine weitere Möglichkeit betrachten, ein Anfangswertproblem zu approximieren. Dabei gehen wir genauer auf neuronale Netze ein. Dieses Kapitel bezieht sich hauptsächlich auf [9]. Obwohl die bisher besprochenen numerischen Verfahren wie Runge-Kutta- und Mehrschrittverfahren heutzutage effizient in Bezug auf Fehlertoleranz und Rechenaufwand sind, ist es dennoch interessant, neuronale Netze zu betrachten. Falls beispielsweise das Interesse besteht, die Lösung einer gewöhnlichen Differentialgleichung nur in einem bestimmten Zeitpunkt  $\hat{t}$  auszuwerten, so müssen in numerischen Verfahren von der Anfangszeit  $t_0$  bis zur gewünschten Zeit  $\hat{t}$  iteriert werden. Ein *trainiertes* neuronales Netz repräsentiert jedoch die gesuchte Lösung und ist deshalb in der Lage eine Funktionsauswertung in  $\hat{t}$  auszuführen, was im Gegensatz zu den numerischen Verfahren viel Rechenarbeit spart. Neuronale Netze haben in Bezug auf das Approximieren mehrerer Differentialgleichungen noch einen ausschlaggebenden Vorteil für die Recheneffizienz, welchen wir in Abschnitt 4.2 betrachten werden.

### 4.1 Struktur neuronaler Netze

Zur Übersicht beschränken wir uns zuerst auf ein neuronales Netz mit einer Eingangsschicht mit drei Neuronen, zwei versteckte Schichten mit jeweils drei Neuronen und eine Ausgangsschicht mit zwei Neuronen, wie in Abbildung 1 dargestellt. Die gelben Neuronen repräsentieren hier jeweils ein *on-Neuron*, oder *Bias*, welches konstant  $x_0^{(l)} = -1$  ist. Allgemein haben wir  $L$  Schichten und  $n^{(l)}$ ,  $l = 1, \dots, L$ , Neuronen pro Schicht, wobei der Bias ausgeschlossen ist. Die Gewichte  $\omega_{ij}^{(l)}$  geben an, in welcher Relation die Neuronen der vorherigen Schicht  $l - 1$  zu den Neuronen der nächsten Schicht  $l$  stehen. Dabei werden die Gewichte der Bias-Neuronen mit  $b_j^{(l)} := \omega_{0j}^{(l)}$  von den anderen Gewichten unterschieden. Die Signale der Neuronen ab der  $l$ -ten Schicht bilden sich aus der Summe

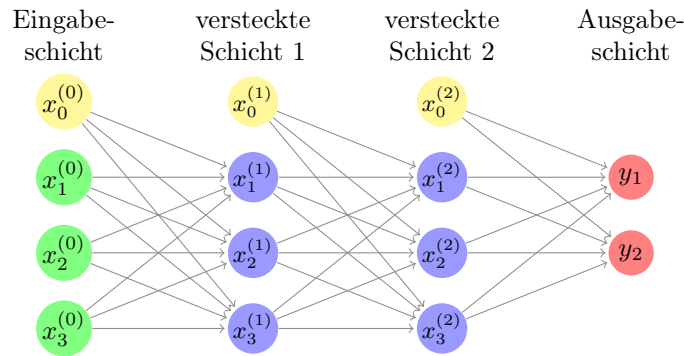


Abbildung 1: Neuronales Netzwerk mit 3 Eingabeneuronen (grün), Bias (gelb), zwei versteckte Schichten (blau) und 2 Ausgabeneuronen (rot).

aller vorherigen Signale mit entsprechender Gewichtung:

$$s_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} \omega_{ij}^{(l)} x_i^{(l-1)} - b_j^{(l)}. \quad (4.1)$$

Darauffolgend ergeben sich die Ausgaben der nächsten Schicht  $l$

$$x_j^{(l)} = \Psi(s_j^{(l)}). \quad (4.2)$$

Dabei ist  $\Psi : \mathbb{R} \rightarrow \mathbb{R}$  die sogenannte *Aktivierungsfunktion*. In unserer Anwendung werden wir hauptsächlich den *Tangens hyperbolicus*

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

als Aktivierungsfunktion nutzen. Das Trainieren eines neuronalen Netzes lässt sich im Grunde auf ein Minimierungsproblem einer *Kostenfunktion*  $\mathcal{C}$  reduzieren. Dazu werden die Konzepte der Forward- und Backwardpropagation, sowie ein *Gradientenverfahren* benötigt, was wir in den folgenden Teilabschnitten besprechen werden.

#### 4.1.1 Forwardpropagation

*Forwardpropagation* wird der Vorgang genannt, in dem man das neuronale Netz Schritt für Schritt durchgeht um die Ausgaben der versteckten Schichten zu berechnen. Dabei wird beginnend mit der ersten versteckten Schicht  $l = 1$  zuerst (4.1), um die jeweiligen Signale zu berechnen und im Anschluss (4.2) angewendet. Im letzten Schritt der Forwardpropagation erhalten wir die Ausgangsschicht  $y_j, j = 1, \dots, n^{(L)}$ . Die Forwardpropagation ist eine Vorbereitung auf den zweiten Schritt des Lernprozesses, der *Backpropagation*.

#### 4.1.2 Backwardpropagation

Sei  $\mathcal{C}(X)$  die oben genannte Kostenfunktion des neuronalen Netzwerks mit der Ausgabe  $Y := (y_j)_j$ , wobei  $X$  die Gewichte aller Schichten inklusive Bias sind. Dabei ist  $Y$  offensichtlich abhängig von den Gewichten und den Bias. Die Kostenfunktion  $\mathcal{C}$  nutzt die Ausgaben  $Y$ , weshalb sie deshalb auch abhängig von den Gewichten und den Bias ist. Unser Ziel ist es  $\mathcal{C}(X)$  zu minimieren, weshalb wir an dessen Steigung, also dem *Gradienten* der Kostenfunktion, interessiert sind. Der Gradient der Kostenfunktion hat für jede Schicht  $l = 1, \dots, L$  die Form

$$\nabla \mathcal{C} = \begin{bmatrix} \frac{\partial \mathcal{C}}{\partial \omega_{01}^{(l)}} & \cdots & \frac{\partial \mathcal{C}}{\partial \omega_{n^{(l)}1}^{(l)}} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{C}}{\partial \omega_{0n^{(l-1)}}^{(l)}} & \cdots & \frac{\partial \mathcal{C}}{\partial \omega_{n^{(l)}n^{(l-1)}}^{(l)}} \end{bmatrix}.$$

Der Gradient der Kostenfunktion  $\mathcal{C}$  kann also in  $\nabla_{\omega} \mathcal{C}$  und  $\nabla_b \mathcal{C}$  aufgeteilt werden. Dabei ist  $\nabla_b \mathcal{C}$  die erste Spalte der obigen Matrix. Das Gewicht  $\omega_{ij}^{(l)}$  fügt das Signal des Neurons in der  $(l-1)$ -ten Schicht zu dem Signal des Neuron der  $l$ -ten Schicht unabhängig von jeglichen anderen Gewichten hinzu. Also ergibt sich folgendes für den Gradienten der Kostenfunktion.

**Satz 4.1.1** Die Komponenten des Gradienten  $\nabla_{\omega} \mathcal{C}$  und  $\nabla_b \mathcal{C}$  der Kostenfunktion sind gegeben durch

$$\begin{aligned} \frac{\partial \mathcal{C}}{\partial \omega_{ij}^{(l)}} &= \frac{\partial \mathcal{C}}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial \omega_{ij}^{(l)}}, \\ \frac{\partial \mathcal{C}}{\partial b_j^{(l)}} &= \frac{\partial \mathcal{C}}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial b_j^{(l)}} \end{aligned}$$

mit  $i = 0, \dots, n^{(l-1)}, j = 1, \dots, n^{(l)}, l = 1, \dots, L$ .

*Beweis* Es ist  $x_j^{(l)} = \psi(s_j^{(l)})$  wie in (4.2). Wir zeigen die Gleichung für  $\nabla_\omega C(w, b)$ , der Beweis für  $\nabla_b C(w, b)$  funktioniert analog mit  $i = 0$ :

$$\begin{aligned} \frac{\partial \mathcal{C}(\omega, b)}{\partial \omega_{ij}^{(l)}} &= \frac{\partial \mathcal{C}(\omega, b)}{\partial x_j^{(l)}} \frac{\partial x_j^{(l)}}{\partial \omega_{ij}^{(l)}} = \frac{\partial \mathcal{C}(\omega, b)}{\partial x_j^{(l)}} \frac{\partial \Psi(s_j^{(l)})}{\partial \omega_{ij}^{(l)}} \\ &= \frac{\partial \mathcal{C}(\omega, b)}{\partial x_j^{(l)}} \frac{\partial \Psi(s_j^{(l)})}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial \omega_{ij}^{(l)}} = \frac{\partial \mathcal{C}(\omega, b)}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial \omega_{ij}^{(l)}}, \end{aligned}$$

wobei in der letzten Gleichheit die Kettenregel benutzt wurde:

$$\frac{\partial \mathcal{C}(\omega, b)}{\partial x_j^{(l)}} \frac{\partial \Psi(s_j^{(l)})}{\partial s_j^{(l)}} = \frac{\partial C(\omega, b)}{\partial x_j^{(l)}} \frac{\partial x_j^{(l)}}{\partial s_j^{(l)}} = \frac{\partial C(\omega, b)}{\partial s_j^{(l)}}.$$

□

Nun definieren wir  $\delta_j^{(l)} := \frac{\partial C}{\partial s_j^{(l)}}$ , welches die Änderung des Kostenfunktion im Bezug zum Signal  $s_j^{(l)}$  angibt. Der zweite Faktor des Gradienten können wir mit Hilfe von (4.1) explizit ausgerechnet und somit folgt

$$\begin{aligned} \frac{\partial \mathcal{C}}{\partial \omega_{ij}^{(l)}} &= \delta_j^{(l)} x_i^{(l-1)}, \\ \frac{\partial \mathcal{C}}{\partial b_j^{(l)}} &= \delta_j^{(l)} x_0^{(l-1)} = -\delta_j^{(l)}. \end{aligned}$$

Hiermit kann der Gradient der Kostenfunktion vereinfacht werden zu

$$\begin{aligned} \nabla \mathcal{C}_\omega &= \delta_j^{(l)} x_i^{(l-1)} \\ \nabla \mathcal{C}_b &= -\delta_j^{(l)}. \end{aligned}$$

Es fällt auf, dass für die Berechnung des Gradienten der Kostenfunktion nur die verschiedenen  $\delta_j^{(l)}$  benötigt werden, da wir die  $x_i^{(l-1)}$  schon in der Forwardpropagation berechnet haben. Diese  $\delta_j^{(l)}$  werden mit dem sogenannten *Backpropagation*-Algorithmus berechnet, welchen wir im folgenden herleiten werden. Im Beweis von Satz 4.1.1 haben wir gesehen, dass für  $\delta_j^{(L)}$  gilt

$$\delta_j^{(L)} = \frac{\partial C}{\partial x_j^{(L)}} \frac{\partial x_j^{(L)}}{\partial s_j^{(L)}} = \frac{\partial C}{\partial x_j^{(L)}} \Psi'(s_j^{(L)}) = \frac{\partial C}{\partial y_j} \Psi'(s_j^{(L)}).$$

Betrachtet man nun die vorletzte Schicht bzw. die letzte versteckte Schicht, so fällt auf, dass das Signal  $s_j^{(L-1)}$  jedes Signal in der Ausgabeschicht beeinflusst und damit folgt mit Hilfe der mehrdimensionalen Kettenregel, dass

$$\delta_i^{(L-1)} = \sum_{j=1}^{n^{(L)}} \frac{\partial C}{\partial s_j^{(L)}} \frac{\partial s_j^{(L)}}{\partial s_i^{(L-1)}} = \sum_{j=1}^{n^{(L)}} \delta_j^{(L)} \frac{\partial s_j^{(L)}}{\partial s_i^{(L-1)}}.$$



Allgemein betrachtet beeinflusst das Signal  $s_j^{(L-1)}$  ausschließlich alle Signale in der nachfolgenden Schicht, also folgt aus der oben hergeleiteten Formel

$$\delta_i^{(l-1)} = \sum_{j=1}^{n^{(l)}} \delta_j^{(l)} \frac{\partial s_j^{(l)}}{\partial s_i^{(l-1)}}, \quad i = 1, \dots, n^{(l-1)}. \quad (4.3)$$

Den rechten Ausdruck in der Summe können wir explizit ausrechnen:

$$\begin{aligned} \frac{\partial s_i^{(l)}}{\partial s_i^{(l-1)}} &= \frac{\partial}{\partial s_i^{(l-1)}} \left( \sum_{k=0}^{n^{(l-1)}} \omega_{kj}^{(l)} x_k^{(l-1)} \right) \\ &= \frac{\partial}{\partial s_i^{(l-1)}} \left( \sum_{k=1}^{n^{(l-1)}} \omega_{kj}^{(l)} \Psi(s_k^{(l-1)}) - b_j^{(l)} \right) \\ &= \omega_{ij}^{(l)} \Psi'(s_i^{(l-1)}). \end{aligned}$$

Setzen wir jetzt das Ergebnis in (4.3), so erhält man die zentrale Formel des *Backpropagation*-Algorithmus:

$$\delta_i^{(l-1)} = \Psi'(s_i^{(l-1)}) \sum_{j=1}^{n^{(l)}} \delta_j^{(l)} \omega_{ij}^{(l)}, \quad i = 1, \dots, n^{(l-1)}. \quad (4.4)$$

#### 4.1.3 Gradientenverfahren

Ein allgemeines Gradientenverfahren mit der *Lernrate*  $\eta > 0$  hat die Form

$$\begin{aligned} \omega_{ij}^{(l)[k+1]} &= \omega_{ij}^{(l)[k]} - \eta \delta_j^{(l)[k]} x_i^{(l-1)[k]} \\ b_{ij}^{(l)[k+1]} &= b_{ij}^{(l)[k]} + \eta \delta_j^{(l)[k]}. \end{aligned}$$

Unser Fokus in Bezug auf Gradientenverfahren wird auf der *ADAM* (engl. adaptive moment estimation) liegen. Sei  $g^{[k]} := \nabla C(\omega^{[k]})$  der Gradient der Kostenfunktion nach den Gewichten (inklusive Bias) in der  $k$ -ten Iteration. Dann ist das ADAM-Verfahren mit den Variablen  $\beta_1, \beta_2 \in [0, 1]$  und den Momenten  $m^{[0]} = 0, v^{[0]} = 0$

$$\begin{aligned} m^{[k]} &= \beta_1 m^{[k-1]} + (1 - \beta_1) g^{[k]} \\ v^{[k]} &= \beta_2 v^{[k-1]} + (1 - \beta_2) (g^{[k]})^2 \end{aligned}$$

und deren Korrekturen

$$\begin{aligned} \hat{m}^{[k]} &= \frac{m^{[k]}}{1 - \beta_1^k}, \\ \hat{v}^{[k]} &= \frac{v^{[k]}}{1 - \beta_2^k} \end{aligned}$$

gegeben durch

$$\omega_{ij}^{[k+1]} = \omega_{ij}^{[k]} - \eta \frac{\hat{m}^{[k]}}{\sqrt{|\hat{v}^{[k]}|} + \epsilon}.$$

Eine detaillierte Herleitung lässt sich in **[ovidiucalinDeepLearningArchitectures]** finden. Der Parameter  $\epsilon > 0$  verhindert eine Division durch 0, falls  $\hat{v}^{[k]} = 0$ . Eine Voreinstellung der Parameter wäre beispielsweise  $\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-8}$  und  $\eta = 0.001$ . Zu Beginn ( $k = 0$ ) müssen jedoch schon Gewichte gegeben sein, um  $g^{[k]}$  berechnen zu können. Dazu werden wir uns in Abschnitt 4.4

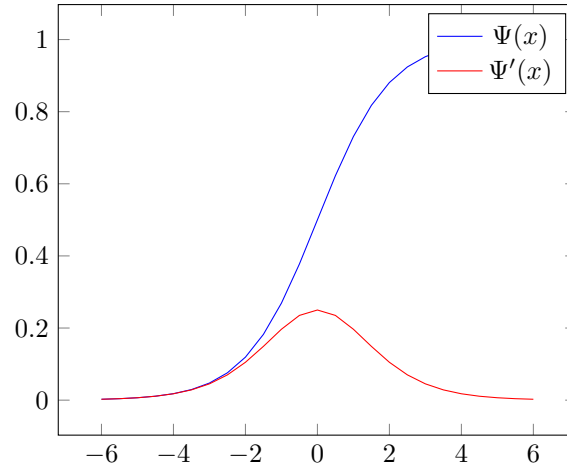


Abbildung 2: Graphen der Sigmoid-Aktivierungsfunktion  $\Psi(x) = \frac{1}{1+e^{-x}}$  und dessen Ableitung.

genauere Gedanken machen.

#### 4.1.4 Verschwindender und explodierender Gradient

Die Berechnung des Gradienten  $\nabla C$  ist fehleranfällig, wobei zwei folgende Probleme auftreten können:

a) **verschwindender Gradient** : Das Problem des *verschwindenden* Gradienten charakterisiert sich durch immer kleiner werdende  $\delta_j^{(l)}$ , was wiederum direkt zu einem verschwindenden Gradienten, also  $\nabla C \approx 0$ , führt.

Angenommen: Die Aktivierungsfunktion  $\Psi$  ist gegeben durch die Sigmoidfunktion, d.h.  $\Psi(x) = \frac{1}{1+e^{-x}}$  und die Gewichte sind zufällig initialisiert, wobei für alle  $\omega_{ij}^{(l)} < 1$  gilt.

Sobald  $|x|$  groß wird, ist die Ableitung  $\Psi'(x) \approx 0$  (siehe Abbildung 2). Dies hat die Auswirkung, dass  $\delta_j^{(L)}$  bei betragsmäßig großen Signalen um den Faktor  $\Psi'(s_j^{(L)})$  kleiner wird. Dadurch werden die nachfolgenden Werte

$$\delta_i^{(l-1)} = \Psi'(s_i^{(l-1)}) \sum_{j=1}^{n^{(l)}} \delta_j^{(l)} \omega_{ij}^{(l)}$$

auch immer kleiner, da sowohl  $\delta_j^{(l)}$  als auch  $\omega_{ij}^{(l)}$  klein sind. Die Werte von  $\delta_i^{(l)}$  werden so klein, dass sie gewichtet mit hinreichend kleiner Lernrate ( $\eta < 1$ ) im Gradientenverfahren keine Auswirkung auf das neue Gewicht haben. Für ein allgemeines Gradientenverfahren gilt dann

$$\omega_{ij}^{(l)[k+1]} = \omega_{ij}^{(l)[k]} - \underbrace{\eta \delta_j^{(l)[k]} x_i^{(l-1)[k]}}_{\approx 0} \approx \omega_{ij}^{(l)[k]}.$$

Um diesem Problem entgegenzuwirken, gibt es mehrere Verbesserungsmöglichkeiten.

**Möglichkeit 1** : Man wählt eine andere Aktivierungsfunktion. Die Effektivität des allgemeinen Gradientenverfahren sinkt rapide bei Funktionen mit Plateaus. Da die Sigmoidfunktion bei betragsmäßig großen Signalen für einen verschwindenden Gradienten  $\nabla C$  sorgt (was zu Plateaus führt), ist sie für neuronale Netze mit mehreren versteckten Schichten nicht geeignet. Aktivierungsfunktionen,

wie zum Beispiel die ReLU-Funktion  $\Psi(x) = \max(0, x)$  mit der Ableitung  $\Psi'(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$  sind besser geeignet für tiefe neuronale Netze, da dessen Ableitungen keine verkleinernde Wirkung

auf  $\delta_i$  haben, sodass der Gradient  $\nabla C$  nicht verschwindet.

**Möglichkeit 2** : Eine geschicktere Initialisierung der Gewichte  $\omega_{ij}^{(l)}$  wie in Abschnitt 4.4.

b) **explodierender Gradient** : Analog zum verschwindenden Gradienten existiert auch das Problem des *explodierenden* Gradienten, in dem  $\delta_j^{(l)} > 1$  ist.

Angenommen: Die Gewichte sind zufällig initialisiert, wobei für alle  $\omega_{ij}^{(l)} > 1$  und  $b_j^{(l)} = 0$  gilt. Bei linearer Regression werden die Signale  $s_j^{(l)}$  beispielsweise für wachsendes  $l$  immer größer, da

$$x_j^{(l)} = \Psi(s_j^{(l)}) = \Psi \left( \sum_{i=1}^{n^{(l)}} \underbrace{\omega_{ij}^{(l)} x_i^{(l-1)}}_{>1} \right)$$

gilt. Solange  $\frac{\partial C}{\partial y_i} > 1$  ist, gilt bei einer Aktivierungsfunktion mit  $\Psi'(x) > 1$  auch  $\delta_j^{(L)} > 1$ , was zu einem explodierenden Gradienten  $\nabla C$  führt. Falls  $\Psi'(x) = 1$  (wie beispielsweise für die ReLU-Funktion bei positiver Eingabe oder linearer Aktivierungsfunktion), so werden die Werte der  $\delta_i^{(l)}$  zwar nicht direkt von der Ableitung der Aktivierungsfunktion beeinflusst, aber bei einem sehr tiefgehendem Netz ( $L > 50$ ) werden die Werte der  $\delta_i$  trotzdem groß, da  $\omega_{ij}^{(l)} > 1$  gilt. Ähnlich wie beim verschwindenden Gradienten werden wir in 4.4 eine geschickte Initialisierung der Gewichte herleiten, um diesem Problem entgegenzuwirken.

#### 4.1.5 Batch Training

Während des Lernprozesses eines neuronalen Netzes wird über die gesamten Eingangsdaten  $X^{(0)}$  iteriert. Dabei nennt man ein einzelnes Element von  $X^{(0)}$  *Probe* (engl. sample). Eine Ansammlung an Proben heißt wiederum *Batch*. Außerdem werden die Lernprozesse auch *Epochen* genannt, dessen Anzahl angibt, wie oft das neuronale Netz die gegebenen Eingangsdaten zum Training nutzt. Es macht Sinn, die Eingangsdaten in Batches einzuteilen, denn so wird Speicherplatz gespart und an Rechengeschwindigkeit gewonnen, da die Gewichte erst nach dem Durchgang eines Batches aktualisiert werden, anstatt sie nach jeder Probe zu berechnen. Dies wird realisiert, indem wir die Gradienten der Proben in einem Batch mit Hilfe der Backpropagation berechnen und im Anschluss den Durchschnitt

$$\widetilde{\nabla C} = \frac{1}{|B|} \sum_{i=1}^{|B|} \nabla C(X^{(0,i)})$$

verwendet, wobei  $|B|$  die Größe der Batches angibt und  $X^{(0,i)}$ ,  $1 \leq i \leq |B|$ , eine Probe des Batches  $B$  ist. Dieser Mittelwert wird dann durch das gegebene Gradientenverfahren verwendet, um die Gewichte  $\omega_{ij}^{(l)}$  zu aktualisieren. Dabei spielt die Größe der Batches  $|B|$  eine wichtige Rolle, denn wenn  $|B|$  zu klein ist, so werden die Abschätzungen des Durchschnitts zu ungenau.

## 4.2 Prinzip der Lösungspakete

Wir werden uns in diesem Abschnitt an der wissenschaftlichen Arbeit [10] orientieren. Unsere Problemstellung liefert im Allgemeinen keine beschriebene Datenmenge (engl. labeled data), mit der wir unser neuronales Netz trainieren können. Wir haben lediglich die gewöhnliche Differentialgleichung erster Ordnung

$$x' = f(t, x) \tag{4.5}$$

mit den Anfangsbedingungen  $x(t_0) = x_0$  gegeben. Dabei definieren wir die Funktion

$$\begin{aligned} G : \mathbb{R}^n \times \mathbb{R}^n \times [t_0, t_f] \times \mathbb{R}^p &\rightarrow \mathbb{R}^n \\ G(x, x', t, \theta) &= x' - f(t, x, \theta), \end{aligned} \quad (4.6)$$

wobei  $p \in \mathbb{N}$  die Anzahl der physikalischen Parametern ist. Hierfür werden wir ein sogenanntes *Lösungspaket* (engl. solution bundle) bilden, welches aus den Lösungen der Gleichung (4.6) besteht.

**Definition 4.2.1** *Seien die Teilmengen  $X_0 \subset \mathbb{R}^n$ ,  $\Theta \subset \mathbb{R}^p$  und  $[t_0, t_f] \subset \mathbb{R}$  gegeben. Dann heißt*

$$x(t; x_0, \theta)$$

Lösungspaket der gewöhnlichen Differentialgleichung (4.6), wobei  $t \in [t_0, t_f]$ ,  $x_0 \in X_0$  und  $\theta \in \Theta$ .

Unser Ziel ist es diese Lösungspakete mit Hilfe neuronaler Netze zu approximieren. Wir definieren die Approximation der Lösung über  $(X_0, \Theta)$  mit

$$\hat{x}(t; x_0, \theta) = x_0 + a(t)N(t; x_0, \theta; \omega), \quad (4.7)$$

wobei  $N : \mathbb{R}^{1+n+p} \rightarrow \mathbb{R}^n$  das neuronale Netz mit den Gewichten  $\omega$  repräsentiert und  $a : [t_0, t_f] \rightarrow \mathbb{R}$  die Eigenschaft, dass  $a(t_0) = 0$  ist. Hierdurch wird das Erfüllen der Anfangswertbedingungen in der Approximation (4.7) sichergestellt. Durch numerische Experimente in [10] wurde herausgefunden, dass  $a$  der Form

$$a(t) = 1 - e^{t_0 - t}$$

besonders gut geeignet ist. Zusammengefasst wollen wir für jedes Lösungspaket  $(t_i, x_{0i}, \theta_i)$  eine Approximation  $\hat{x}$  finden, so dass

$$G(\hat{x}(t_i; x_{0i}, \theta_i), \hat{x}'(t_i; x_{0i}, \theta_i), t_i, \theta_i) \approx 0.$$

Die Kostenfunktion ergibt sich dann durch

$$\mathcal{C} = \frac{1}{|B|} \sum_{(t_i, x_{0i}, \theta_i) \in B} \mu(t_i) \|G(\hat{x}(t_i; x_{0i}, \theta_i), \hat{x}'(t_i; x_{0i}, \theta_i), t_i, \theta_i)\|_2^2, \quad (4.8)$$

wobei  $\mu : [t_0, t_f] \rightarrow \mathbb{R}$  durch  $\mu(t) = e^{\lambda(t_0 - t)}$  gegeben ist, wobei  $\lambda > 0$ . Die Batches  $B$  bestehen aus Ansammlungen von 3-Tupel  $(t_i, x_{0i}, \theta_i)$ , also  $B = \{(t_i, x_{0i}, \theta_i) \in [t_0, t_f] \times X_0 \times \Theta\}$  mit der Kardinalität  $|B|$ , welche *Batch Size* genannt wird. Mit vorgegebenen Gebieten für  $X_0$  und  $\Theta$  durch die Problemstellung (bspw. physikalische Zusammenhänge der gegebenen Differentialgleichung) können die Eingabewerte bestimmt werden. Dafür genügt in den meisten Fällen eine zufällige Wahl aus einer Gleichverteilung der Mengen  $X_0$  und  $\Theta$ . Da wir in unserer Anwendung theoretisch unendlich Eingangsdaten zur Verfügung haben, kann der Begriff einer *Epoche* nicht definiert werden. Stattdessen repräsentiert jeder Batch eine einzigartige Probe die für den Trainingsablauf genutzt wird, wobei eine steigende Anzahl der Proben eine bessere Approximation liefert. Der Korrekturfaktor  $\mu(t_i)$  wird zur Beeinflussung des lokalen Fehlers

$$\tau(t_i; x_{0i}, \theta_i) := G(\hat{x}(t_i; x_{0i}, \theta_i), \hat{x}'(t_i; x_{0i}, \theta_i), t_i, \theta_i)$$

eingeführt. Ähnlich wie in Satz 3.1.5 für Einschrittverfahren können wir auch hier einen Zusammenhang des lokalen und des globalen Fehlers des neuronalen Netzes zeigen, wobei hier die rechte Seite  $f$  der gewöhnlichen Differentialgleichung (4.5) Lipschitz-stetig im zweiten Argument ist. Der

globale Fehler des neuronalen Netzes ist gegeben mit  $e(t) = \hat{x}(t) - x(t)$ , wobei  $x$  die Lösung von (4.5) ist.

**Satz 4.2.2** Für das Anfangswertproblem (4.5) mit im zweiten Argument Lipschitz-stetiger rechten Seite  $f$  ist der lokale Fehler  $\tau$  gegeben durch  $\tau(t) = \hat{x}'(t) - f(t, \hat{x})$ . Es gilt

$$\|e(t)\|_2 = \frac{\tau_{t_f}}{L} \left( e^{L(t-t_0)} - 1 \right),$$

wobei  $\tau_{t_f} = \max_{t_0 \leq t \leq t_f} \|\tau(t)\|_2$  und  $L$  die Lipschitz-Konstante der rechten Seite  $f$ .

*Beweis.* Der lokale Fehler ergibt sich durch einfaches Konstruieren von  $G$ .

Für den globalen Fehler  $e(t) = \hat{x}(t) - x(t)$  gilt

$$\hat{x}(t) = e(t) + x(t).$$

Einsetzen in den lokalen Fehler ergibt

$$\begin{aligned} \tau(t) &= e'(t) + x'(t) - f(t, e(t) + x(t)) \\ \text{oder } e'(t) &= \tau(t) - x'(t) + f(t, e(t) + x(t)), \end{aligned}$$

wobei  $x$  die Lösung von (4.5) ist, also  $x' = f(t, x(t))$ . Des Weiteren ist  $f$  Lipschitz-stetig im zweiten Argument, das heißt es gilt

$$\|f(t, x(t) + e(t)) - f(t, x(t))\|_2 \leq L \|e(t)\|_2.$$

Also kann man die Norm der rechten Seite obiger Gleichung schreiben als

$$\begin{aligned} \|\tau(t) + f(t, e(t) + x(t)) - x'(t)\|_2 &= \|\tau(t) + f(t, e(t) + x(t)) - f(t, x(t))\|_2 \\ &\leq \|\tau(t)\|_2 + L \|e(t)\|_2 \\ &\leq \tau_{t_f} + L \|e(t)\|_2. \end{aligned}$$

Dabei wurde die Dreiecksungleichung benutzt und der lokale Fehler wurde durch das Maximum aller lokalen Fehler abgeschätzt. Jetzt suchen wir eine Funktion  $E(t)$  die folgende Gleichung erfüllt

$$E'(t) = \tau_{t_f} + LE(t),$$

wobei  $0 \leq \|e(t)\|_2 \leq E(t)$  und  $E(t_0) = 0$  gilt. Also haben wir eine lineare gewöhnliche Differentialgleichung erster Ordnung, dessen Lösung sich mit (2.12) berechnen lässt

$$\begin{aligned} E(t) &= e^{L(t-t_0)} E(t_0) + \int_{t_0}^t e^{L(t-s)} \tau_{t_f} ds \\ &= \tau_{t_f} e^{Lt} \int_{t_0}^t e^{-Ls} ds \\ &= \frac{\tau_{t_f}}{L} \left( e^{L(t-t_0)} - 1 \right). \end{aligned}$$

Daraus folgt

$$\|e(t)\|_2 \leq \frac{\tau_{t_f}}{L} \left( e^{L(t-t_0)} - 1 \right),$$

was sich mit der Abschätzung in Satz 3.1.5 vergleichen lässt.  $\square$

Hierdurch erkennt man, dass der globale Fehler durch einen frühen lokalen Fehler exponentiell mit

der Zeit wachsen kann. Deshalb ist es sinnvoll, eine exponentiell fallende Korrekturfunktion  $\mu$  zu nutzen, um den lokalen Fehler in frühen Zeitpunkten zu skalieren.

### 4.3 Intervall-Lernen

Dieser Abschnitt beruht auf [cedricflamantPersonlicheKommunikation2022]. Beim Approximieren von gewöhnlichen Differentialgleichungen mithilfe von Lösungspaketen ist es unser Ziel, die Kostenfunktion zu minimieren. Dabei kann ein Fehler am Anfang des Trainingsprozesses schon dazu führen, dass das neuronale Netz das gegebene Anfangswertproblem nicht approximiert, wie in Satz 4.2.2 zu sehen ist. Genauer wird die Trajektorie (Definition siehe Kapitel ??) der approximierten Funktion durch den Fehler in andere Bereiche des Phasenraums geleitet. Dies wiederum sorgt für einen größeren globalen Fehler, da nun ein anderes Anfangswertproblem approximiert wird. Eine mögliche Behebung dieses Problems ist das Prinzip des sogenannten *Intervall-Lernens* (eng. curriculum learning). Eine allgemeine Diskussion dieses Prinzips lässt sich in [11] nachlesen. Für das Approximieren von gewöhnlichen Differentialgleichungen nutzen wir das Intervall-Lernen so, dass wir das Intervall der Zeiten  $[t_0, t_f]$  begrenzen. Hierzu werden wir die Batches  $B$  anpassen, also erhalten wir neue Batches  $B'$  mit

$$B' := \{(t_i, x_{0i}, \theta_i) \in [t_0, t_m] \times X_0 \times \Theta\},$$

wobei  $t_m < t_f$  und  $m$  die Anzahl der bereits trainierten Batches ist. Wir lassen also das neuronale Netz zuerst mit einer kleinen Menge an Daten trainieren, um problematische Stellen, die zu einer Verschiebung der Trajektorien sorgen, zu vermeiden. Dabei wird die Menge der Daten in Abhängigkeit der bereits trainierten Batches erhöht. In Kapitel 5 werden wir

$$t_m = \frac{t_f}{\ln(11)} \ln \left( 10 \frac{m}{|B|} + 1 \right), \quad m = 1, \dots, |B|,$$

und

$$\lambda = \frac{4}{t_m + 5}$$

zur Beschränkung des Zeitintervalls verwenden.

### 4.4 Gewichtsinitialisierung

Die Gewichtsinitialisierung ist ein entscheidender Faktor um Probleme des explodierenden und verschwindenden Gradienten entgegenzuwirken. In diesem Abschnitt werden wir ein Intervall bestimmen, welches dann zur Initialisierung der Gewichte  $\omega_{ij}$  genutzt wird. Dazu werden wir die  $(l-1)$ -te und  $l$ -te Schicht eines neuronalen Netzes und die zugehörigen Gewichte  $\omega_{ij} := \omega_{ij}^{(l)}$  sowie  $x_i^{(l-1)}$  als unabhängige, identisch verteilte Zufallszahlen betrachten. Außerdem gilt für den Erwartungswert der Gewichte  $\mathbb{E}(\omega_{ij}) = 0$  und die Bias  $b$  werden mit 0 initialisiert. Genauer zur Berechnung folgender Varianzen lässt sich in [ovidiuocalinDeepLearningArchitectures] nachlesen. Für die Varianz der Ausgabe der  $l$ -ten Schicht  $x_i^{(l)}$  ergibt sich

$$\text{Var}(x_j^{(l)}) = n^{(l-1)} \Psi'(0)^2 \text{Var}(\omega_{ij}) \text{Var}(x_i^{(l-1)}).$$

Unter der Voraussetzung, dass die Varianz der beiden Schichten gleich bleibt, also  $\text{Var}(x_j^{(l)}) = \text{Var}(x_j^{(l-1)})$ , ergibt sich für die Varianz der Gewichte

$$\text{Var}(\omega_{ij}) = \frac{1}{n^{(l-1)}\Psi'(0)^2}.$$

Die gegebene Voraussetzung macht Sinn um die oben erwähnten Probleme des explodierenden bzw. verschwindenden Gradienten zu vermeiden. Genau wie bei der Varianz der Ausgaben  $x_j^{(l-1)}$  und  $x_j^{(l)}$  ist es auch vorteilhaft die Varianz der Kostenfunktion ausgehend von der Backpropagation gleichzustellen. Also folgt aus

$$\text{Var}\left(\frac{\partial C}{\partial \omega_{ij}^{(l-1)}}\right) = \text{Var}\left(\frac{\partial C}{\partial \omega_{ij}^{(l)}}\right)$$

mit Hilfe einiger Umformungen (vgl. [ovidiucalinDeepLearningArchitectures]) und der Annahme, dass die Aktivierungsfunktion  $\Psi(x) = x$  ist, die Gleichung

$$\text{Var}(\delta_j^{(l-1)}) = \text{Var}(\delta_j^{(l)}),$$

wobei  $\delta_j^{(l)}$  die aus der Backpropagation resultierende Variable ist. Hiermit folgt für Varianz der Gewichte

$$\text{Var}(\omega_{ij}) = \frac{1}{n^{(l)}}.$$

Das Erfüllen beider Voraussetzungen führt zu einer restriktiven Bedingung  $n^{(l-1)} = n^{(l)}$ , weshalb man sich für

$$\text{Var}(\omega_{ij}) = \frac{2}{n^{(l-1)} + n^{(l)}}$$

einigt. Werden die Gewichte nun gleichverteilt auf  $[-a, a]$ , dessen Varianz  $\frac{a^2}{3}$  gleichgestellt mit der Varianz der Gewichte  $a = \frac{\sqrt{6}}{\sqrt{n^{(l-1)} + n^{(l)}}}$  ergibt, also

$$\omega_{ij} \sim \left[ -\frac{\sqrt{6}}{\sqrt{n^{(l-1)} + n^{(l)}}}, \frac{\sqrt{6}}{\sqrt{n^{(l-1)} + n^{(l)}}} \right]$$

für alle  $i = 1, \dots, n^{(l-1)}$ ,  $j = 1, \dots, n^{(l)}$ ,  $l = 1, \dots, L$ . Diese Verteilung heißt *Xavier- oder Glorot-Initialisierung*. In Betracht der Aktivierungsfunktion  $\Psi(x) = \tanh(x)$  kann eine Initialisierung mit der Gleichverteilung

$$\omega_{ij} \sim \left[ -\frac{\sqrt{6}}{\sqrt{n^{(l-1)} \tanh(2)}}, \frac{\sqrt{6}}{\sqrt{n^{(l-1)} \tanh(2)}} \right]$$

gefunden werden, vgl. hierbei [12, S. 27].

## 5 Numerische Auswertung

Wir werden nun die numerischen Verfahren zur Lösung von Anfangswertproblemen mit dem Verfahren der Lösungspakete miteinander vergleichen. Es werden hierfür die Python Bibliotheken *numpy*, *scipy* für numerische Verfahren und *tensorflow* für die Architektur der neuronalen Netze verwendet. Das gegebene Anfangswertproblem wird zuerst mithilfe des *4-5-Runge-Kutta-Verfahren* beziehungsweise einem BDF-Verfahren gelöst und danach mit einem neuronalen Netz approximiert. Die folgenden Ergebnisse werden darauffolgend mit verschiedenen Maßen verglichen. Dabei sind unter anderem die *Trajektorien* der berechneten Lösungen gemeint. Eine Trajektorie ist hierbei

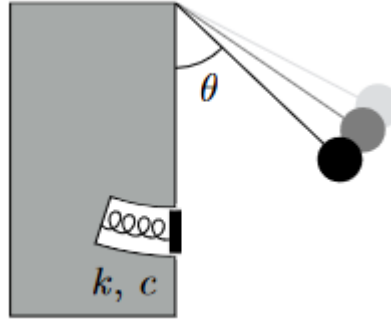


Abbildung 3: Diagramm eines Rebound Pendels[10, S. 6]

das Bild  $T(x_0) := \{x(t, x_0) : t \in I_{\max}\}$  der Lösung des Anfangswertproblems (3.1), wobei  $I_{\max}$  das maximale Existenzintervall ist. Des Weiteren werden wir den globalen Fehler des numerischen Verfahrens  $\|x(t_i) - u_i\|_2$  mit dem globalen Fehler der neuronalen Netze  $\|\hat{x}(t) - x(t)\|_2$  in Abhängigkeit der Zeit  $t_i = t_0 + ih$  und  $t \in D$  vergleichen. Dabei ist zu beachten, dass eine exakte Lösung  $x$  vorliegen muss. In dem Abschnitt 5.1 ist das analytische Berechnen der Lösung aufwendig, weshalb es nicht möglich ist, die globalen Fehler zu vergleichen. Deshalb werden wir ein Runge-Kutta-Verfahren mit sehr kleiner Schrittweite zum Vergleich verwenden, da Einschrittverfahren für  $h \rightarrow 0$  konvergieren. Hierbei betrachten wir sowohl die Trajektorien, als auch den Graph der ersten Komponente der berechneten Lösung in Abhängigkeit der Zeit. Zuletzt geben wir die Kostenfunktion der neuronalen Netze in Abhängigkeit der Epochen und, wenn möglich, den globalen Fehler für das gesamte Zeitintervall in Abhängigkeit der Epochen an. Dabei wenden wir den sogenannten *gleitenden Mittelwert* an, um die Oszillation der beiden Graphen durch die Varianzen der Batches, zu minimieren. Dabei sind sprunghafte Änderungen der Kostenfunktion und des Fehlers gemeint, die entstehen, sobald neue Zufallszahlen gewählt werden und das Netzwerk noch nicht angepasst wurde. Hierzu verwenden wir die Funktion `pandas.DataFrame.mean` der Bibliothek `pandas`. Die Dokumentation dieser Funktion findet sich in [13]. Für das 4-5-Runge-Kutta-Verfahren verwenden wir die Bibliothek `scipy.integrate`, genauer die Funktion `scipy.integrate.RK45` (Dokumentation siehe [14]). Die Berechnungen wurden mit einem AMD Ryzen 7 5800X CPU, 32GB RAM und einer NVIDIA RTX 3060 Ti GPU durchgeführt. Der Quellcode zur Lösung der folgenden Anfangswertproblemen mit numerischen Verfahren und die Implementierung der neuronalen Netze befindet sich auf dem beigelegtem Datenträger.

### 5.1 Rebound-Pendel

Das Rebound-Pendel ist ein System, in dem ein Pendel an einer Seite auf eine gedämpfte Feder treffen kann. Ein Beispiel des beschriebenen Systems wird in Abbildung 3 gezeigt. Das zugehörige Anfangswertproblem zweiter Ordnung hat die Form

$$\begin{aligned}\theta'' &= -\frac{g}{l} \sin(\theta) + H(-\theta) \text{ReLU}\left(-\frac{kl}{m} \theta - c\theta'\right) \\ \theta(0) &= 1, \quad \theta'(0) = 0.2\end{aligned}$$



Netzwerkstruktur	
Anzahl der Schichten	$L = 10$
Eingabeschicht	$n^{(0)} = 5$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 128, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Hyperparameter und Initialisierungsintervalle	
Anfangswertbereiche	$(\theta_0, \varphi_0) \in [0.0, 1.0] \times [-0.2, 0.2]$
Paramaterbereiche	$(k, c) \in [2.0, 5.0] \times [0.0, 2.0]$
Zeitraum	$t \in [0, 10]$
Optimierung	
Gradientenverfahren	Adam
Gewichtsfunktion	$b(t) = e^{-0.5t}$
Batch Size	$ B  = 10000$
Epochen	100000
Lernrate	$\eta = 0.0001$
Gewichtsinitialisierung	Xavier-Initialisierung
Statistiken	
Trainingrate	9 Batches/Sekunde
Trainingszeit	3.1 Stunden

Tabelle 1: Netzwerkstruktur, Hyperparameter, Initialisierungsintervalle, Optimierungsparameter und Statistiken für das Rebound-Pendel-Anfangswertproblem.

mit dem Auslenkungswinkel  $\theta$ , beziehungsweise mit der Einführung der Winkelgeschwindigkeit  $\varphi = \theta'$  zu einem System erster Ordnung

$$\begin{aligned}
 \theta' &= \varphi \\
 \varphi' &= -\frac{g}{l} \sin(\theta) + H(-\theta) \text{ReLU}\left(-\frac{kl}{m} \theta - c\varphi\right) \\
 \theta(0) &= 1, \quad \varphi(0) = 0.2.
 \end{aligned} \tag{5.1}$$

Hierbei ist  $\text{ReLU}(x) = \max(x, 0)$ ,

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

die *Heavside*-Funktion,  $g$  die Erdbeschleunigung,  $l$  die Länge des Pendels,  $m$  die Masse des Pendels,  $k$  die Federkonstante und  $c$  der Dämpfungskoeffizient. Wir werden dieses System nun in dem Zeitraum  $t \in [0, 10]$  lösen bzw. approximieren und die Resultate vergleichen. Für beide Verfahren werden zur Vereinfachung  $g = 1$  und  $l = 1$  gesetzt. Außerdem wird für das Runge-Kutta-Verfahren  $k = 3$  und  $c = 1$  gewählt. Für die Funktionen zur Berechnung der Lösungen mithilfe der Runge-Kutta-Verfahren werden die relative Toleranz  $rtol = 10^{-5}$  und die globale Toleranz  $atol = 10^{-7}$  gesetzt. Für den Vergleichswert, der durch eine weitere Berechnung der *scipy.integrate.RK45*-Funktion entsteht, setzen wir  $rtol = atol = 10^{-14}$ , um kleinere Schrittweiten zuzulassen. In Tabelle 1 sind alle relevanten Daten für das Rebound-Pendel-Anfangswertproblem gegeben, wobei  $\Phi$  die Aktivierungsfunktion für die jeweiligen Schichten ist. Wie bereits erwähnt, haben wir keine exakte Lösung vorliegen, weshalb wir die berechneten Lösungen mit dem Runge-Kutta-Verfahren mit sehr kleinen Schrittweite vergleichen. In Abbildung 4 sehen wir, dass sich die beiden Trajektorien einen ähnlichen Verlauf haben. Jedoch sind die Trajektorien des Runge-Kutta-Verfahrens viel näher an der Vergleichstrajektorie. Ähnliches lässt sich in Abbildung 5 beobachten, wo beide Verfahren einen ähnlichen Verlauf haben, aber das Runge-Kutta-Verfahren näher an dem Vergleichsgraphen

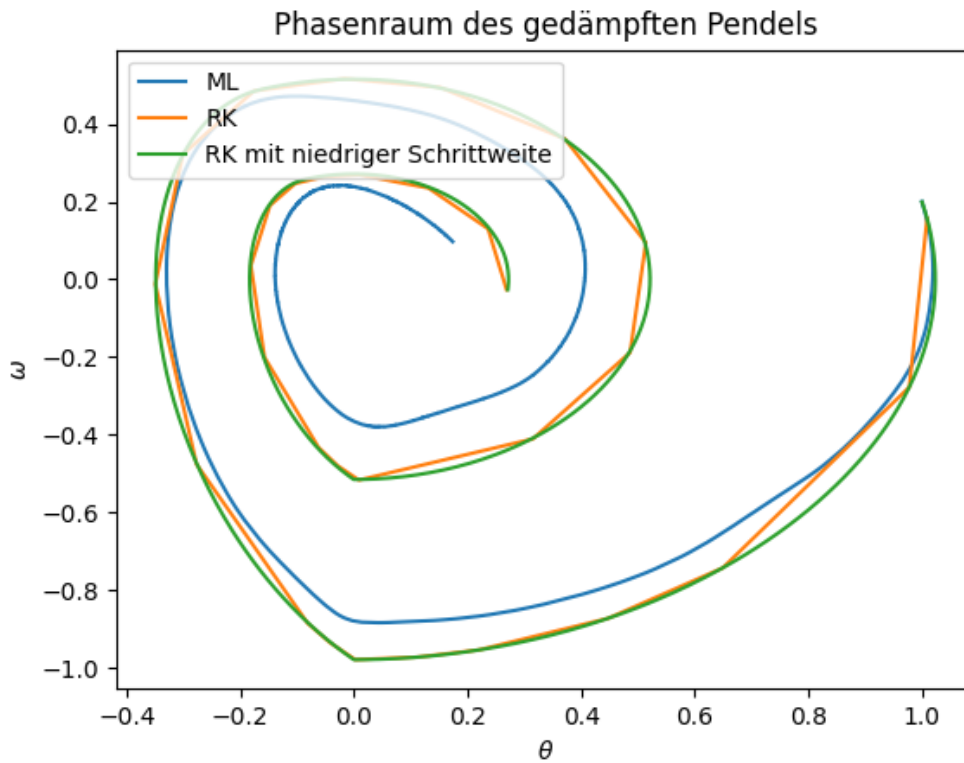


Abbildung 4: Trajektorie der verschiedenen Lösungen.

als die Approximation des neuronalen Netzes ist. In Abbildung 6 befindet sich die Entwicklung der Kostenfunktion bei steigenden Epochen. Man kann beobachten, dass die Werte der Kostenfunktion bei hohen Epochen wieder steigt. Dies zeigt, dass eine Erhöhung der Epochen ohne Anpassung anderer Hyperparameter, wie beispielsweise der Lernrate, nicht zielführend für eine Verbesserung der Approximation sein kann. Zusammenfassend erkennen wir, dass das Runge-Kutta-Verfahren ein besseres Ergebnis liefert und sobald wir die relative und absolute Toleranz verringern, kann dieses Ergebnis noch verbessert werden.

## 5.2 Steife Differentialgleichung

Das Anfangswertproblem

$$\begin{aligned} x_1' &= \frac{1}{2}((\lambda_1 + \lambda_2)x_1 + (\lambda_1 - \lambda_2)x_2), \\ x_2' &= \frac{1}{2}((\lambda_1 - \lambda_2)x_1 + (\lambda_1 + \lambda_2)x_2), \\ x_1(0) &= c_1 + c_2, \quad x_2(0) = c_1 - c_2 \end{aligned} \tag{5.2}$$

lässt sich mit der Matrix

$$A = \frac{1}{2} \begin{bmatrix} \lambda_1 + \lambda_2 & \lambda_1 - \lambda_2 \\ \lambda_1 - \lambda_2 & \lambda_1 + \lambda_2 \end{bmatrix}$$

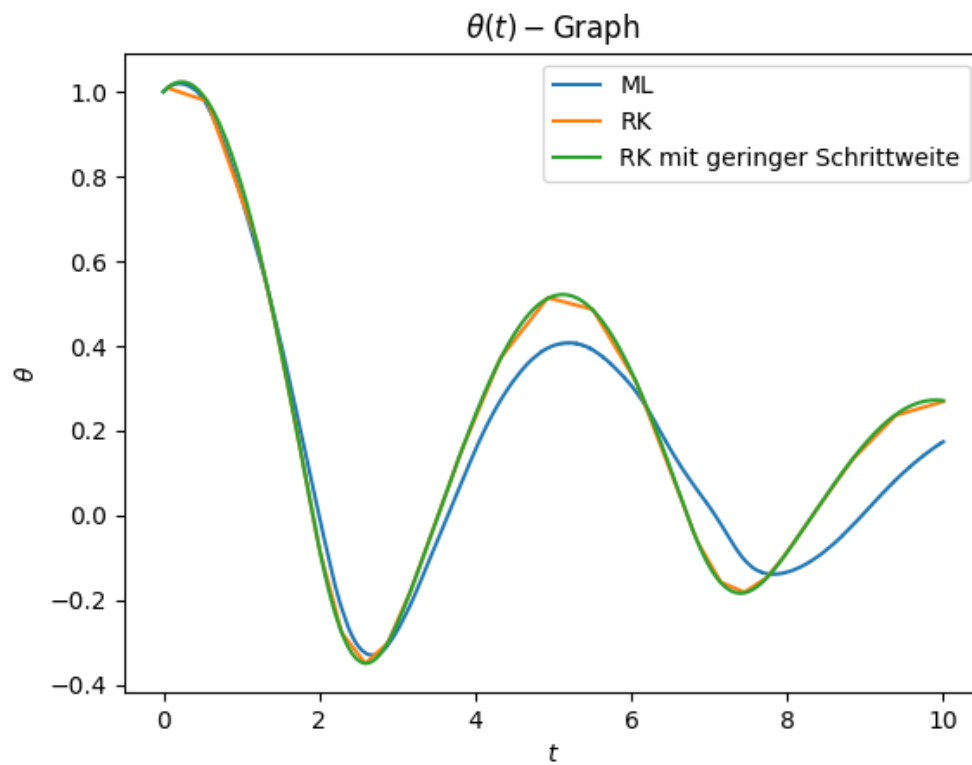
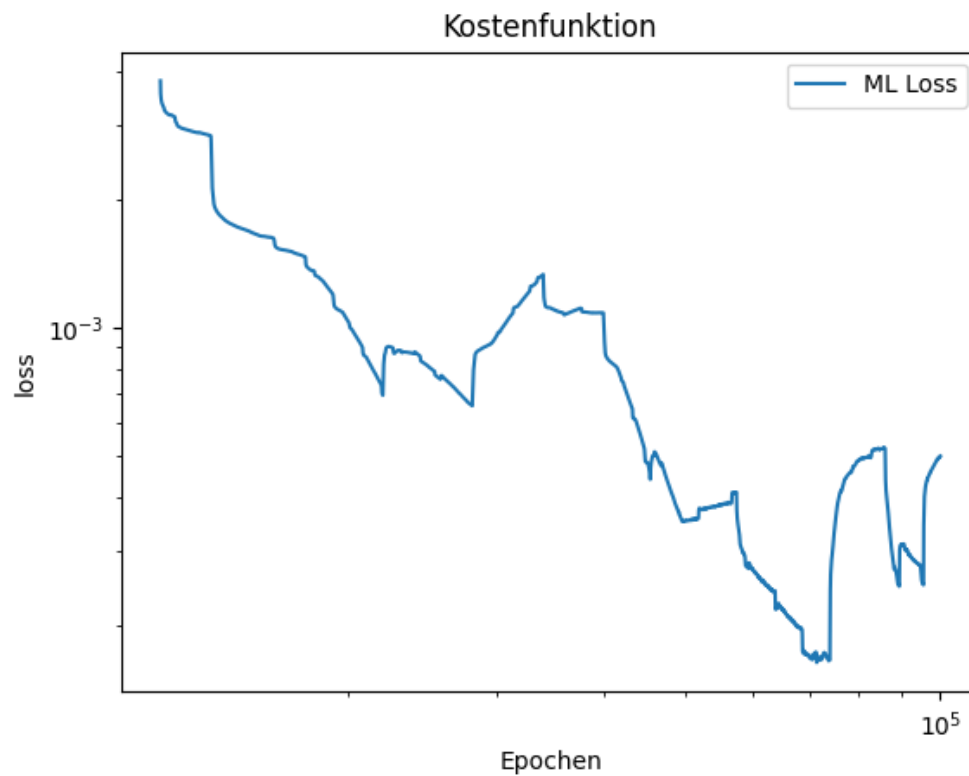
Abbildung 5:  $(t, \theta(t))$  Graph der berechneten Lösungen.

Abbildung 6: Kostenfunktion des neuronalen Netzes bei steigender Epochenanzahl.

zu

$$x' = Ax,$$

$$x(0) = \begin{bmatrix} c_1 + c_2 \\ c_1 - c_2 \end{bmatrix}$$

umformulieren. Dabei betrachten wir  $\lambda_1, \lambda_2 < 0$  und  $c_1, c_2 \in \mathbb{R}$ . Die Matrix  $A$  hat die Eigenwerte  $\lambda_1$  und  $\lambda_2$  und für  $\lambda_1 = -100$  und  $\lambda_2 = -1$  ist das Anfangswertproblem (5.2) nach Definition (2.1.6) *steif*. Da (5.2) eine lineare Differentialgleichung erster Ordnung ist, können wir die Lösung mithilfe (2.12) angeben:

$$x(t) = \begin{bmatrix} c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t} \\ c_1 e^{\lambda_1 t} - c_2 e^{\lambda_2 t} \end{bmatrix}.$$

Der erste Eigenwert  $\lambda_1$  lässt den ersten Summand der Lösung viel schneller gegen 0 konvergieren für  $t \rightarrow \infty$ , als der zweite Eigenwert  $\lambda_2$ . Verwenden wir nun das explizite Euler-Verfahren mit der Form

$$u_0 = x_0$$

$$u_i = u_{i-1} + hAu_{i-1} = \dots = (I + hA)^i u_0, \quad i = 1, \dots, N$$

um das Anfangswertproblem zu lösen. Die numerische Lösung ist also gegeben mit

$$u_i = \begin{bmatrix} c_1(1 + h\lambda_1)^i + c_2(1 + h\lambda_2)^i \\ c_1(1 + h\lambda_1)^i - c_2(1 + h\lambda_2)^i \end{bmatrix}.$$

Damit die exakte Lösung und die numerische Lösung das gleiche Grenzverhalten besitzen, muss  $|1 + h\lambda_1| < 1$  und  $|1 + h\lambda_2| < 1$  gelten. Hieraus resultieren Bedingungen für  $h$ , also eine Schrittweitenbeschränkung der Form  $h < \frac{2}{|\lambda_1|}$  und  $h < \frac{2}{|\lambda_2|}$ . Da  $|\lambda_2| \ll |\lambda_1|$ , bestimmt der erste Eigenwert über die Beschränkung der Schrittweite, obwohl dieser in der exakten Lösung für größere Zeiten  $t$  fast keinen Einfluss hat. Wir werden sehen, dass dieses Problem auch für Runge-Kutta-Verfahren auftreten, weshalb wir außerdem ein *BDF-Verfahren*, also ein Mehrschrittverfahren, verwenden werden, um die Ergebnisse mit dem Verfahren der Lösungspakete vergleichen zu können. Für alle Verfahren setzen wir  $c_1 = 3$ ,  $c_2 = 4$ ,  $\lambda_1 = -100$  und  $\lambda_2 = -1$ . Für die Implementierung des BDF-Verfahrens wird die Funktion `scipy.integrate.BDF` (Dokumentation siehe [15]) verwendet. Für beide Verfahren setzen wir dazu die relative Toleranz  $rtol = 10^{-2}$  und die absolute Toleranz  $atol = 10^{-7}$  um die steife Eigenschaft des gegebenen Anfangswertproblems (5.2) beobachten zu können. Ähnlich wie in Abschnitt 5.1 sind in Tabelle 2 alle relevanten Daten zu dem verwendeten neuronalen Netz (5.2) gegeben. In Abbildung 7 können wir den Verlauf der Trajektorie mit dem Anfangspunkt  $x_0 = x(0) = (7, -1)^\top$  für die erwähnten Verfahren betrachten. Wie bereits besprochen beschränkt der betragsmäßig große Eigenwert  $\lambda_1 = -100$  die Schrittweite  $h$ , was zu einer ungenauen Lösung des Runge-Kutta-Verfahren führt. Die Approximation des neuronalen Netzes jedoch lässt sich mit der Lösung des BDF-Verfahrens vergleichen, da es für ein gewissen Zeitintervall sogar einen niedrigeren globalen Fehler hat als die Lösung des BDF-Verfahrens. Dieses Verhalten lässt sich auch in der Abbildung des Fehlers in Abhängigkeit der Zeit 8 wiederfinden.

Netzwerkstruktur	
Anzahl der Schichten	$L = 10$
Eingabeschicht	$n^{(0)} = 5$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 32, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Hyperparameter und Initialisierungsintervalle	
Anfangswertbereiche	$x_{1,0} \in [c_1 + c_2 - 0.05, c_1 + c_2 + 0.05]$ $x_{2,0} \in [c_1 - c_2 - 0.05, c_1 - c_2 + 0.05]$
Paramaterbereiche	$\lambda_1 \in [\lambda_1 - 0.05, \lambda_1 + 0.05]$ $\lambda_2 \in [\lambda_2 - 0.05, \lambda_2 + 0.05]$
Zeitraum	$t \in [0, 10]$
Optimierung	
Gradientenverfahren	Adam
Gewichtsfunktion	$b(t) = 1$
Batch Size	$ B  = 10000$
Epochen	1000000
Lernrate	$\eta = 0.0001$
Gewichtsinitialisierung	Xavier-Initialisierung
Statistiken	
Trainingrate	125 Batches/Sekunde
Trainingszeit	2.22 Stunden

Tabelle 2: Netzwerkstruktur, Hyperparameter, Initialisierungsintervalle, Optimierungsparameter und Statistiken für das steife Anfangswertproblem (5.2).

Der oszillierende Fehler des Runge-Kutta-Verfahrens stimmt mit dem Verhalten der Trajektorie überein. Des Weiteren fällt auf, dass der Fehler des neuronalen Netzes über die Zeit nahezu konstant bleibt, während der Fehler der anderen Verfahren im Laufe der Zeit sinkt. Zudem sehen wir in 9 und 10 den Verlauf des globalen Fehlers und der Kostenfunktion für steigende Epochen. Sowohl der globale Fehler als auch die Kostenfunktion sinken wie erwartet bei steigender Epchenzahl, jedoch wird die Steigung beider Graphen betragsmäßig kleiner. Um eine genauere Approximation zu erhalten, müssten wir die Anzahl der Epochen erhöhen, wobei dies zu einer Laufzeit von über  $\approx 2.2$  Stunden führen würde. Das BDF-Verfahren hat trotz einer Beschränkung der globalen und relativen Toleranz das Anfangswertproblem in 0.039 Sekunden mit akzeptablen Fehler gelöst. Deshalb können wir zusammenfassend sagen, dass das BDF-Verfahren allgemein qualifizierter ist zur Lösung von steifen Anfangswertproblemen.

### 5.3 Harmonischer Oszillator

Zuletzt werden wir die Lösung für den harmonischen Oszillator mit verschiedenen Netzwerkstrukturen approximieren und die Ergebnisse untereinander und mit dem 4-5-Runge-Kutta-Verfahren vergleichen. Die Bewegung eines harmonischen Oszillators ist gegeben mit

$$\begin{aligned} x' &= v, & v' &= -\frac{k}{m}x, \\ x(0) &= 0, & v(0) &= 0, \end{aligned} \tag{5.3}$$

wobei zur Vereinfachung  $m = 1$  gesetzt wird. Außerdem gilt für das Runge-Kutta-Verfahren  $k = 1$ , für relative Toleranz  $rtol = 10^{-10}$  und die globale Toleranz  $atol = 10^{-14}$ . Die Tabelle 3 enthält alle Daten, welche für alle folgenden neuronale Netze verwendet werden. Wir werden zuerst eine konstante Anzahl Schichten  $L = 6$  verwenden und die Anzahl der Neuronen pro Schicht variieren. Dazu betrachten wir versteckte Schichten mit den Längen  $n^{(l)} = 4, 8, 16, 32, l = 1, \dots, L - 1$ .

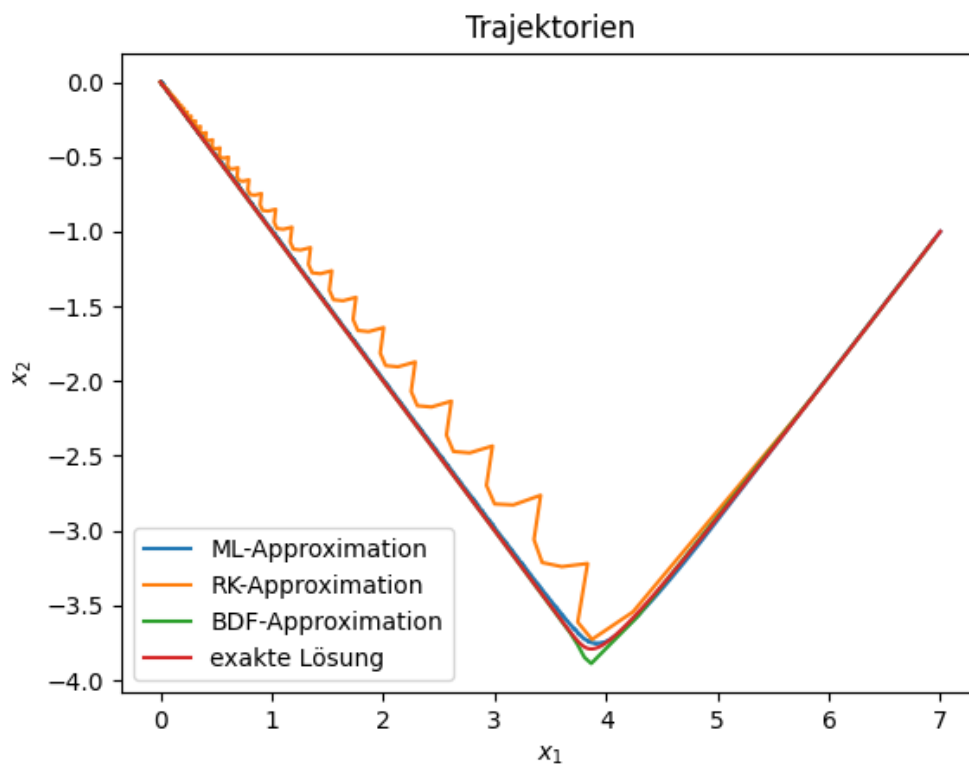


Abbildung 7: Trajektorien der verschiedenen Lösungen.

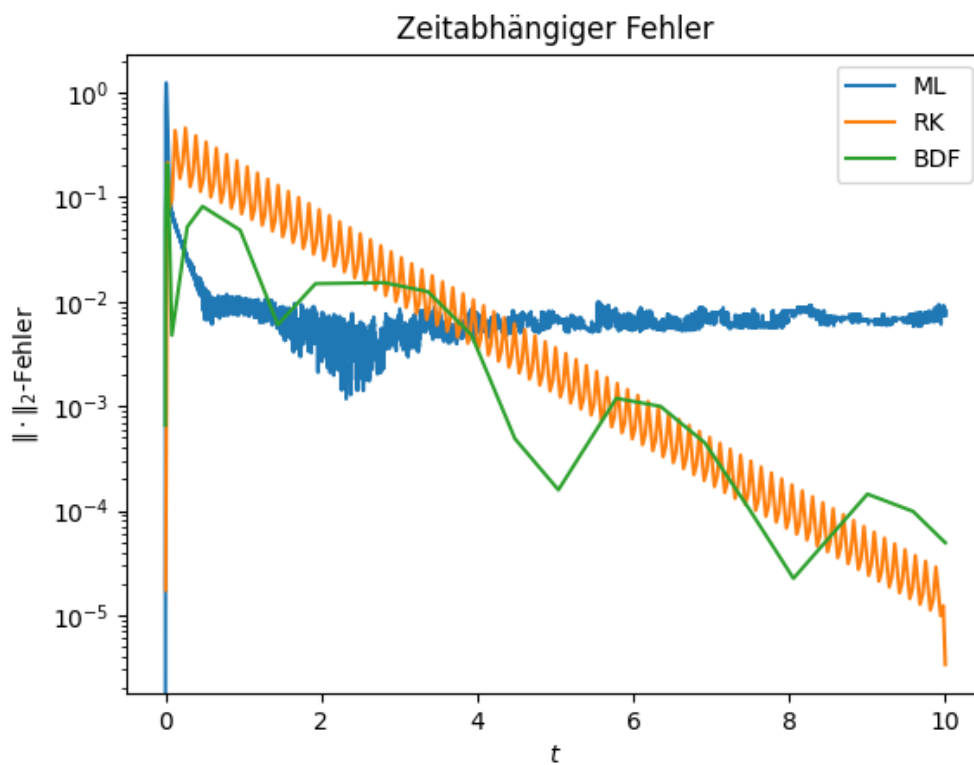


Abbildung 8: Globaler Fehler der jeweiligen Verfahren in Abhängigkeit der Zeit.

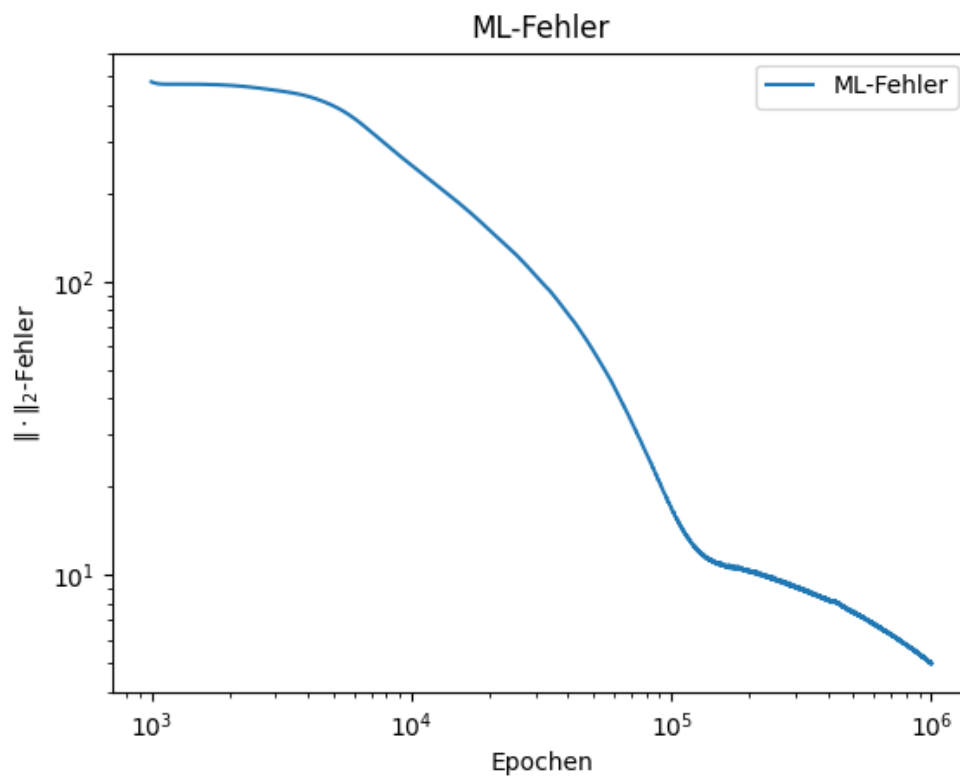


Abbildung 9: Globaler Fehler für steigende Epochen.

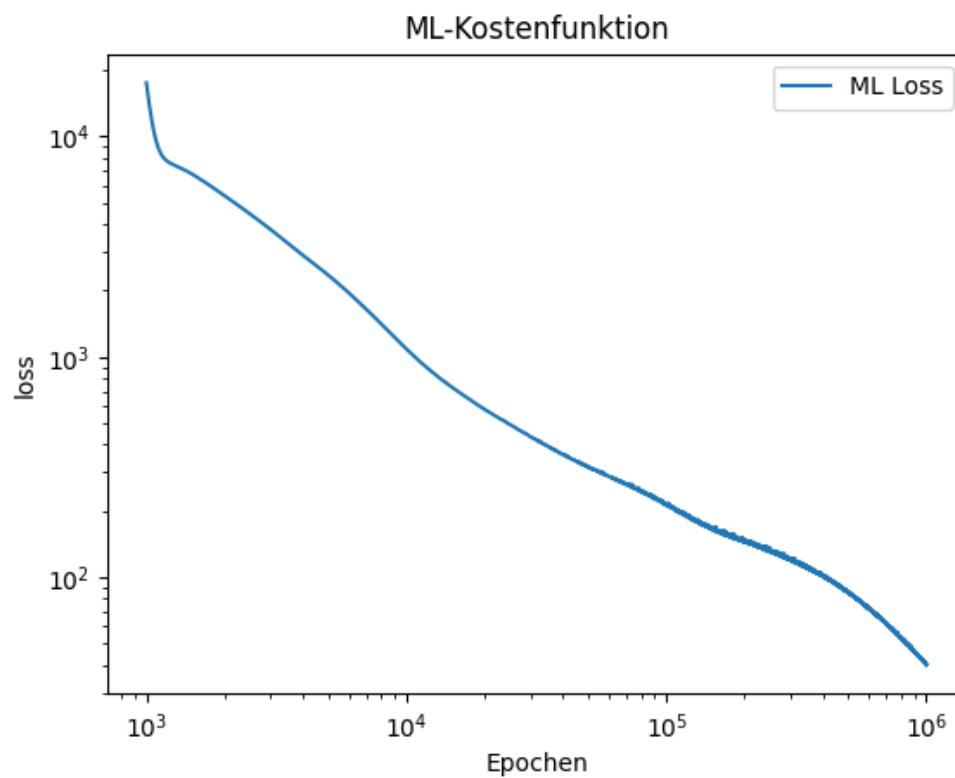


Abbildung 10: Kostenfunktion für steigende Epochen.

Hyperparameter und Initialisierungsintervalle	
Anfangswertbereiche	$(x_0, v_0) \in [-1.0, 1.0] \times [-1.0, 1.0]$
Paramaterbereiche	$k \in [0.5, 2.0]$
Zeitraum	$t \in [0, 2\pi]$
Optimierung	
Gradientenverfahren	Adam
Gewichtsfunktion	$b(t) = 1$
Batch Size	$ B  = 10000$
Epochen	100000
Lernrate	0.0001
Gewichtsinitialisierung	Xavier-Initialisierung

Tabelle 3: Hyperparameter, Initialisierungsintervalle und Optimierungsparameter für den harmonischen Oszillator (5.3).

Tabelle 4 enthält die genauen Netzwerkstrukturen. In Abbildung 11 und Abbildung 12 lässt sich leicht erkennen, dass das bei steigender Anzahl der Neuronen sowohl die Kostenfunktion als auch der globale Fehler das größte Minimum erreicht. Auch in der Abbildung der Trajektorien 14 und dem  $x(t)$ -Graphen in Abbildung 15 sehen wir, dass das neuronale Netz mit den meisten Neuronen pro Schicht die beste Lösung liefert. Die Lösung des Runge-Kutta-Verfahrens approximiert die exakte Lösung jedoch viel besser, da die Trajektorie direkt unter der Trajektorie der exakten Lösung liegt. Dies sieht man auch am globalen Fehler des 4. Netzwerkes 12, denn dieser besitzt die maximale Größenordnung  $10^0$  und in Abbildung 13 ist zu erkennen, dass der globale Fehler für die einzelnen Zeitpunkte für das Runge-Kutta-Verfahren um 4 Größenordnungen kleiner ist. Zuletzt werden wir Netzwerke mit 4, 8 und 16 versteckte Schichten verglichen. Dazu wird die Länge der jeweiligen Schichten  $n^{(l)} = 32$ . Genauere Angaben zu den Netzwerken befinden sich in Tabelle 5. Hier fällt auf, dass Netzwerk 1 die niedrigste Kostenfunktion erreicht, was in Abbildung 16 zu sehen ist. In Abbildung 17 lässt sich jedoch erkennen, dass Netzwerk 2 den niedrigsten globalen Fehler erzielt. Ähnlich wie bei der ersten Variation der Neuronen befindet sich der globale Fehler des besten Netzwerkes in der Größenordnung  $10^0$ . Die Trajektorien in Abbildung 19 bestätigen, dass Netzwerk 3 die schlechteste Approximation des Anfangswertproblems (5.3) liefert. Außerdem lässt sich an den Trajektorien und Abbildung 20 erkennen, dass Netzwerk 2 die beste Approximation liefert, wobei die Trajektorien und auch der  $(t_i, u_i)$ -Graph der Lösung  $u_i$  des Runge-Kutta-Verfahrens wiederum direkt unter der exakten Lösung liegen. Abbildung 18 bestätigt diese Aussage, da der globale Fehler des Runge-Kutta-Verfahrens um 2 Größenordnungen kleiner ist. Zusammenfassend können wir sagen, dass eine Erhöhung der Neuronen und Schichten abhängig von der Komplexität des Anfangswertproblems ist. Außerdem muss die Trainingszeit erhöht werden, um eine vergleichbar gute Approximation gegenüber dem Runge-Kutta-Verfahren zu erhalten.



Netzwerk 1	
Anzahl der Schichten	$L = 6$
Eingabeschicht	$n^{(0)} = 4$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 4, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Lernrate	$\eta = 0.0001$
Trainingrate	195 Batches/Sekunde
Trainingszeit	0.14 Stunden
Netzwerk 2	
Anzahl der Schichten	$L = 6$
Eingabeschicht	$n^{(0)} = 4$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 8, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Lernrate	$\eta = 0.0001$
Trainingrate	195 Batches/Sekunde
Trainingszeit	0.14 Stunden
Netzwerk 3	
Anzahl der Schichten	$L = 6$
Eingabeschicht	$n^{(0)} = 4$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 16, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Lernrate	$\eta = 0.0001$
Trainingrate	195 Batches/Sekunde
Trainingszeit	0.14 Stunden
Netzwerk 4	
Anzahl der Schichten	$L = 6$
Eingabeschicht	$n^{(0)} = 4$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 32, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Lernrate	$\eta = 0.0001$
Trainingrate	180 Batches/Sekunde
Trainingszeit	0.15 Stunden

Tabelle 4: Netzwerkstrukuren der ersten Variation.

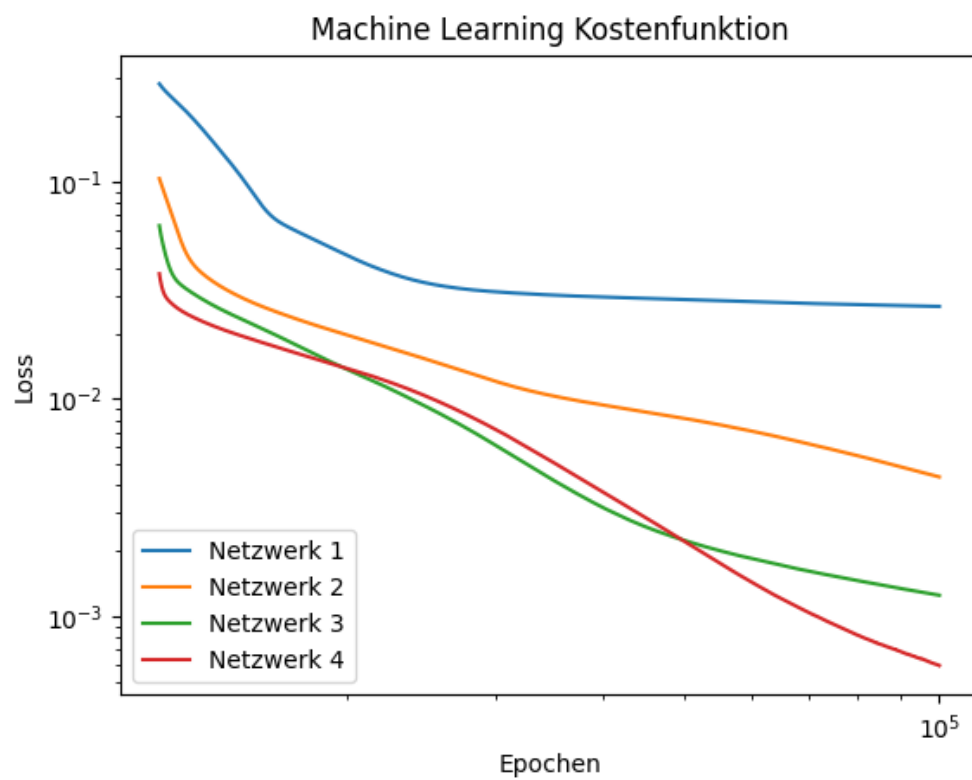


Abbildung 11: Kostenfunktion der verschiedenen neuronalen Netze.

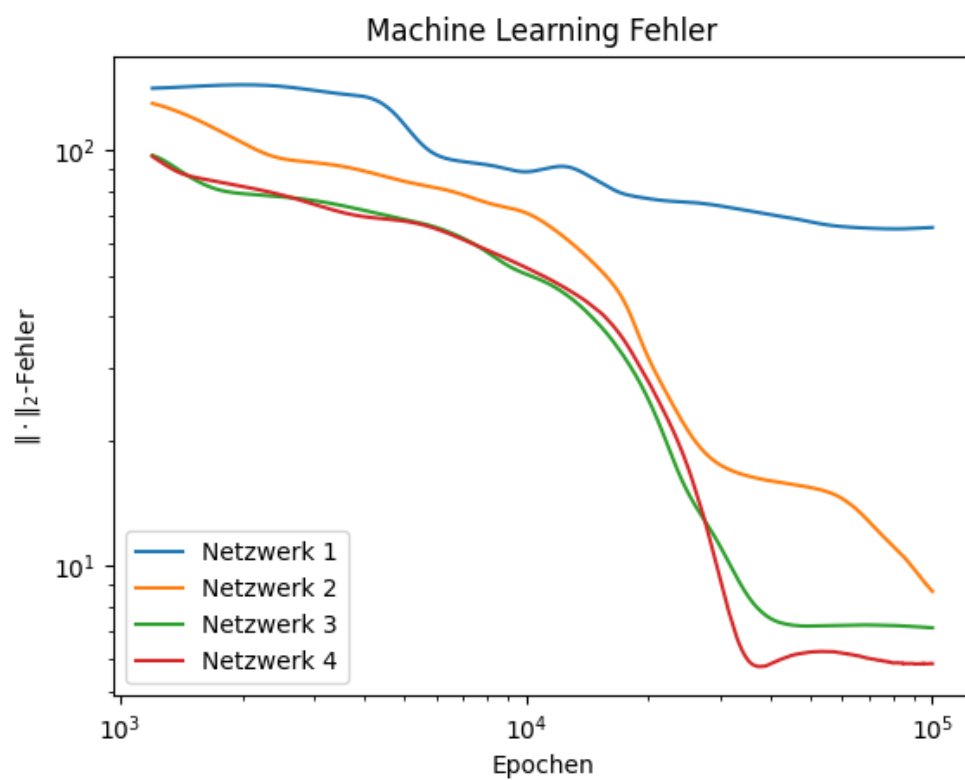


Abbildung 12: Globaler Fehler der verschiedenen neuronalen Netze.

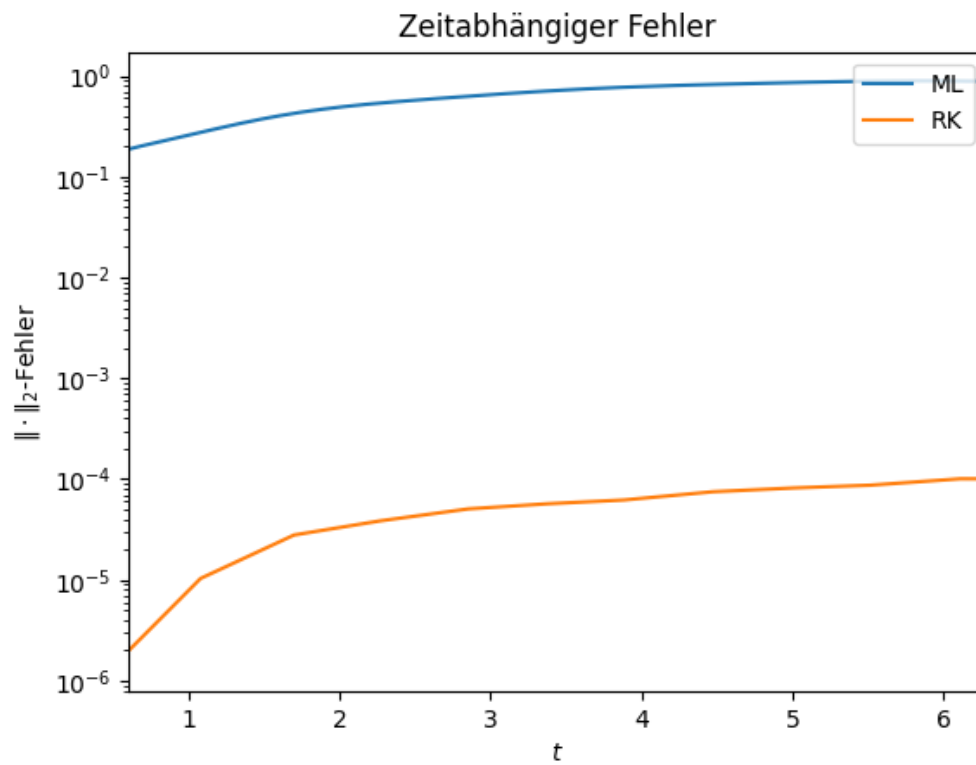


Abbildung 13: Globaler Fehler von Netzwerk 4 und dem Runge-Kutta-Verfahren in Abhängigkeit der Zeit.

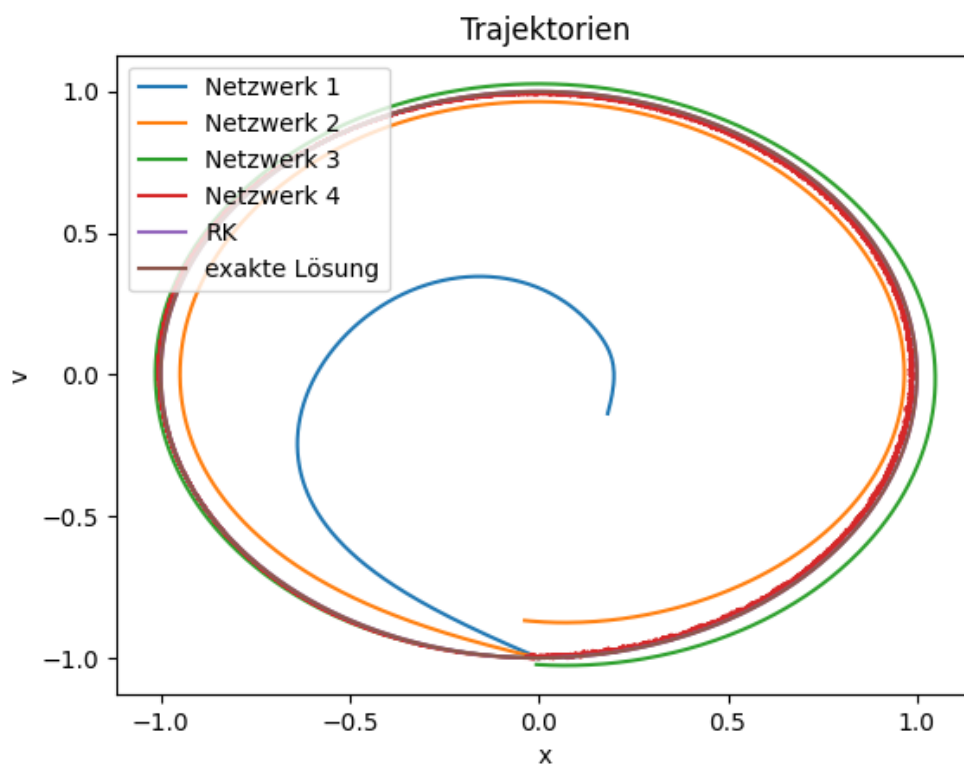


Abbildung 14: Trajektorien der von den verschiedenen neuronalen Netzen approximierten Lösung, der Runge-Kutta-Lösung und der exakten Lösung.

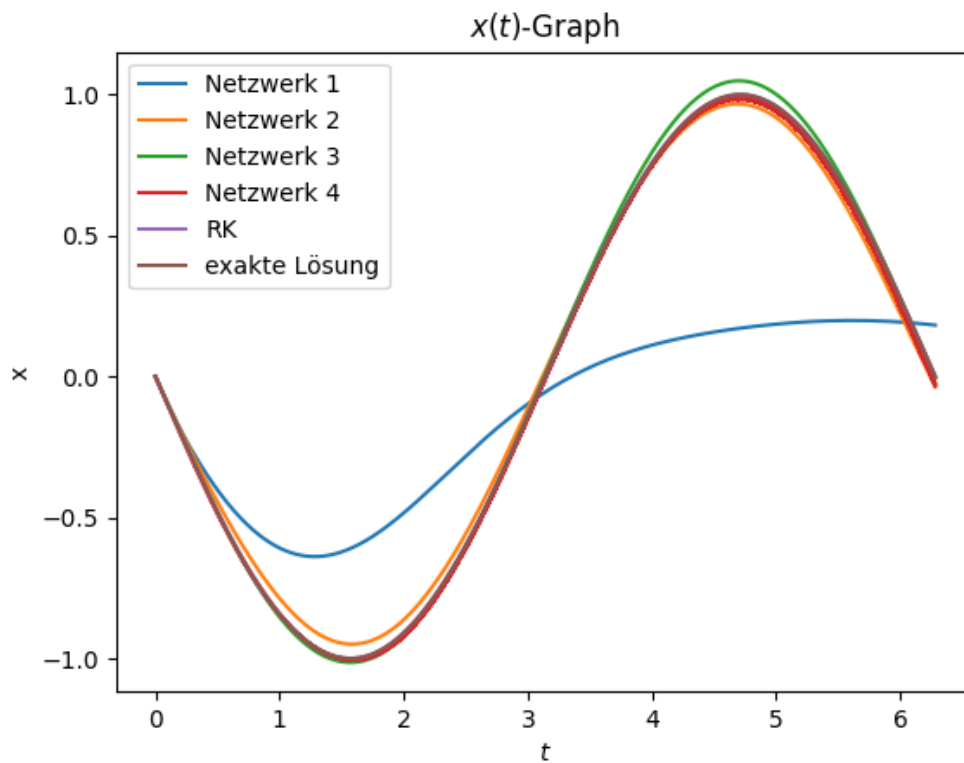


Abbildung 15:  $(t, x(t))$  Graph der Lösungen des Runge-Kutta-Verfahrens, der Approximation der Lösungspakete und der exakten Lösung.

Netzwerk 1	
Anzahl der Schichten	$L = 6$
Eingabeschicht	$n^{(0)} = 4$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 32, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Lernrate	$\eta = 0.0001$
Trainingrate	185 Batches/Sekunde
Trainingszeit	0.15 Stunden
Netzwerk 2	
Anzahl der Schichten	$L = 10$
Eingabeschicht	$n^{(0)} = 4$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 32, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Lernrate	$\eta = 0.0001$
Trainingrate	129 Batches/Sekunde
Trainingszeit	0.22 Stunden
Netzwerk 3	
Anzahl der Schichten	$L = 18$
Eingabeschicht	$n^{(0)} = 4$ mit $\Phi(x) = \tanh(x)$
Versteckte Schichten	$n^{(l)} = 32, l = 1, \dots, L - 1$ mit $\Phi(x) = \tanh(x)$
Ausgabeschicht	$n^{(L)} = 2$ mit $\Phi(x) = x$
Lernrate	$\eta = 0.0001$
Trainingrate	76 Batches/Sekunde
Trainingszeit	0.36 Stunden

Tabelle 5: Netzwerkstrukturen der zweiten Variation.

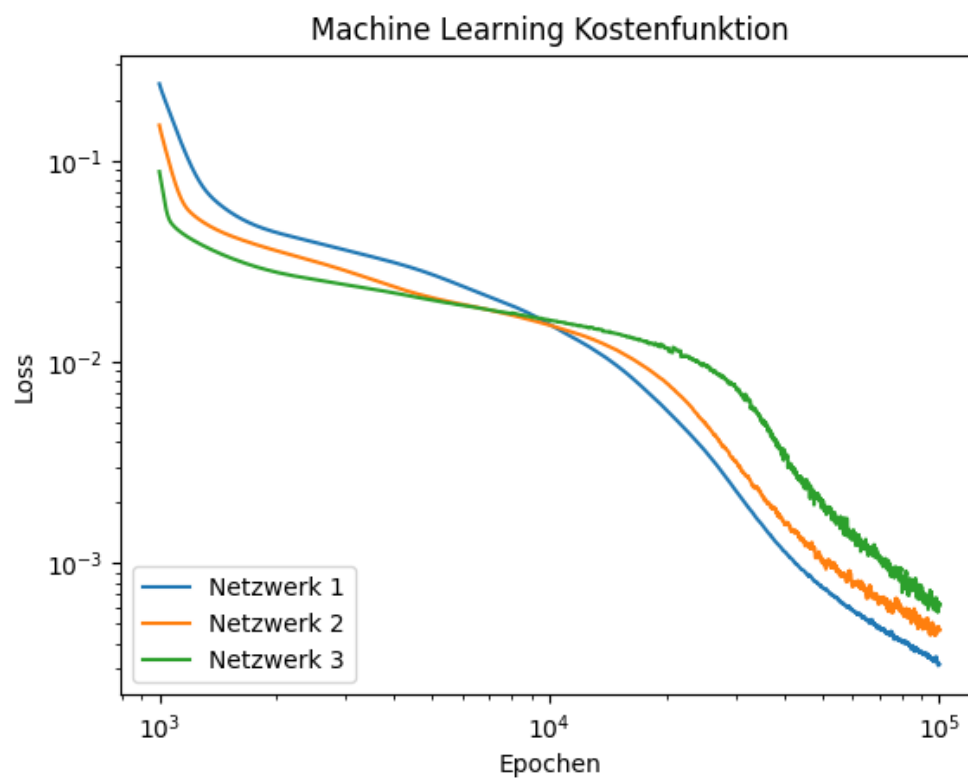


Abbildung 16: Kostenfunktion der verschiedenen neuronalen Netze.

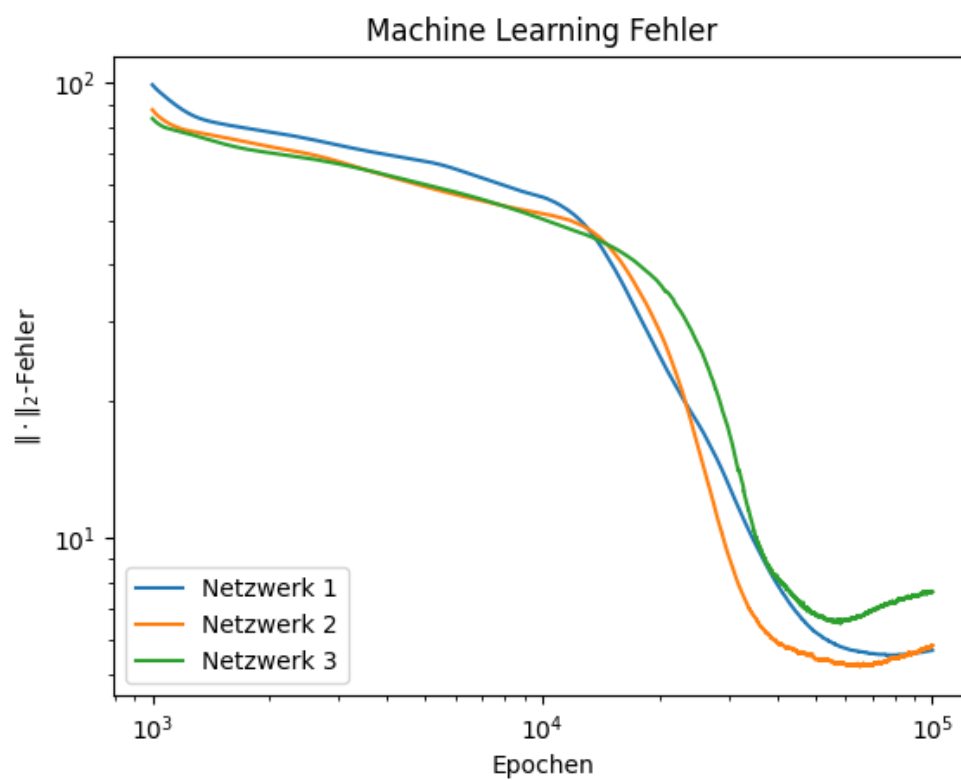


Abbildung 17: Globaler Fehler der verschiedenen neuronalen Netze.

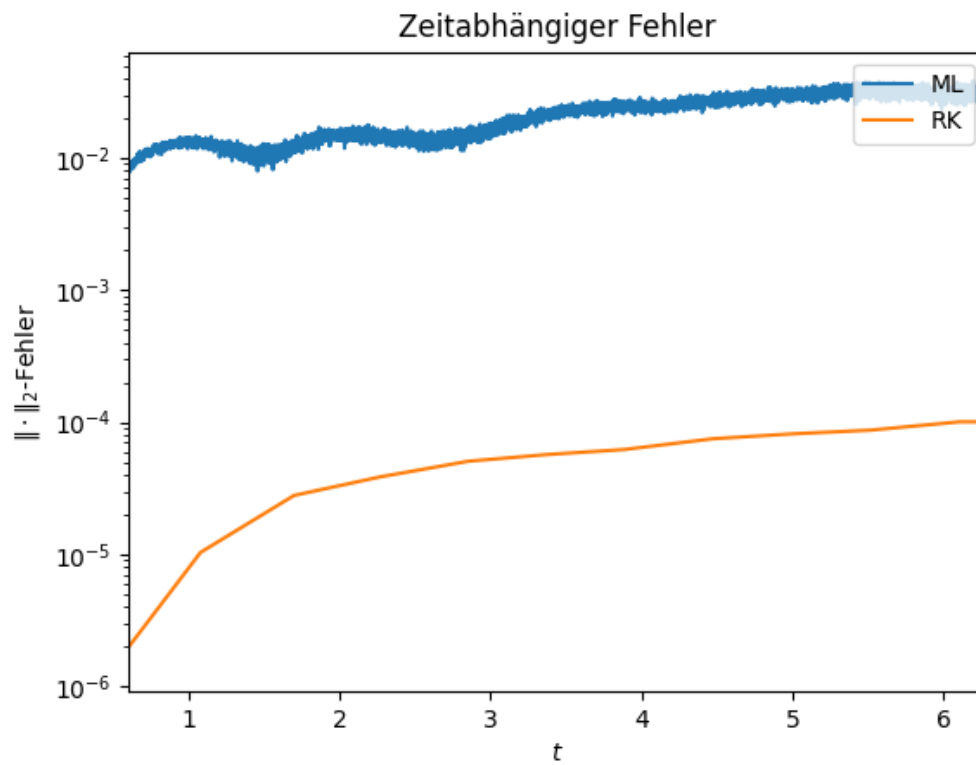


Abbildung 18: Globaler Fehler von Netzwerk 2 und dem Runge-Kutta-Verfahren in Abhängigkeit der Zeit.

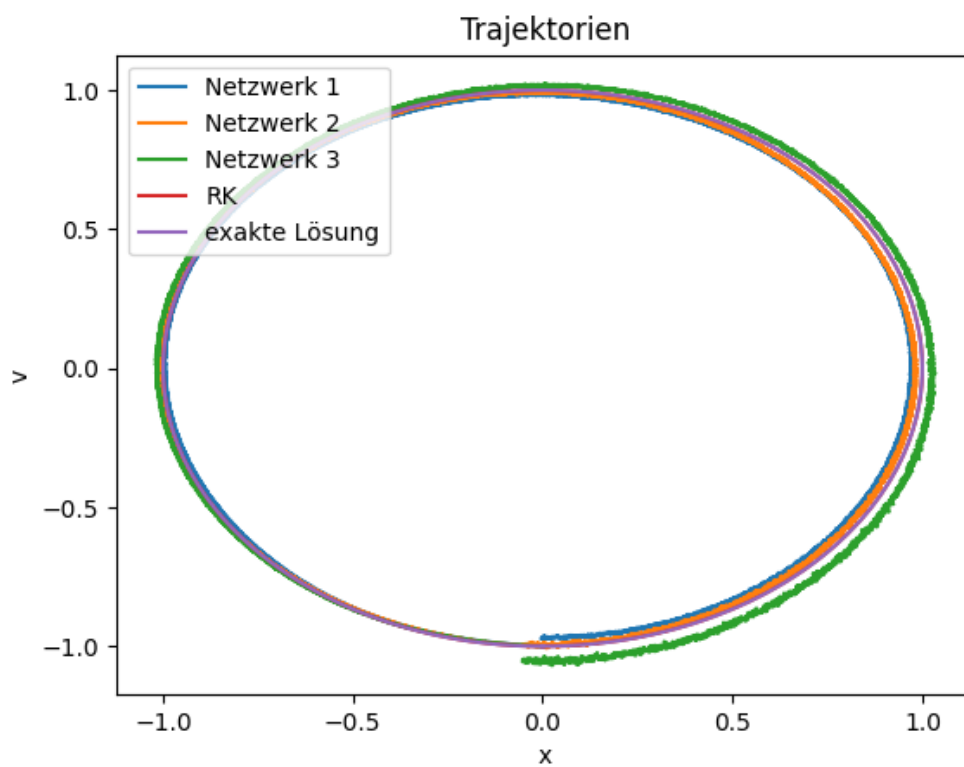


Abbildung 19: Trajektorien der von den verschiedenen neuronalen Netzen approximierte Lösung, der Runge-Kutta-Lösung und der exakten Lösung.

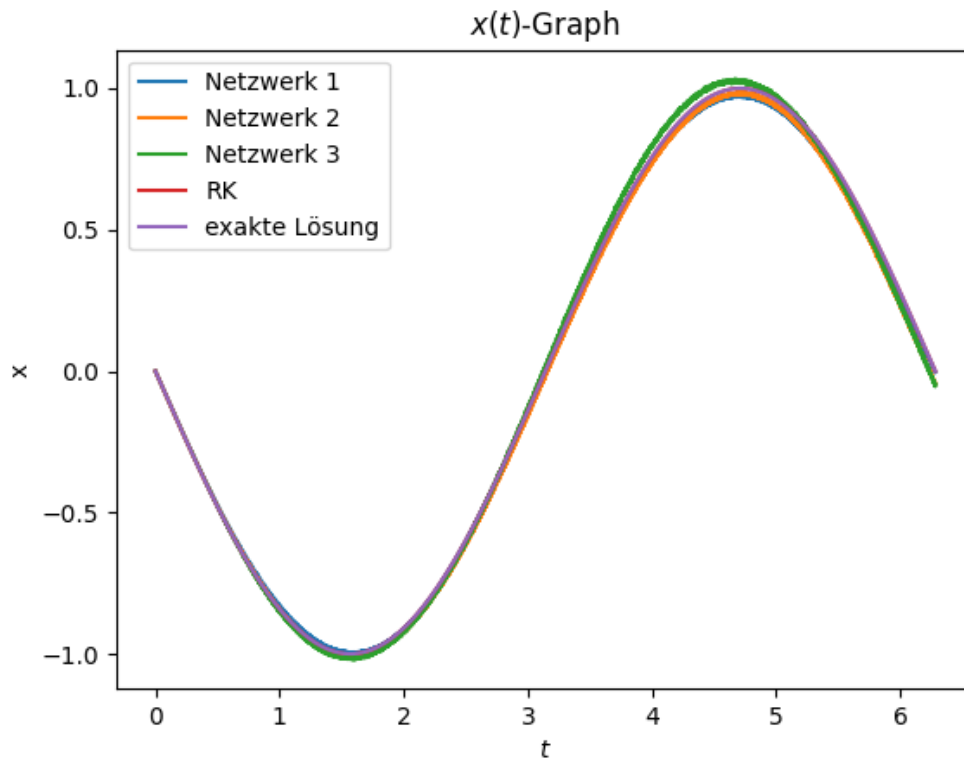


Abbildung 20:  $(t, x(t))$ -Graph der von den verschiedenen neuronalen Netzen approximierte Lösung, der Runge-Kutta-Lösung und der exakten Lösung.

## 6 Fazit

Wir haben an verschiedenen Anfangswertproblemen gesehen, wie sich Trajektorien, Graphen in Abhängigkeit der Zeit, Kostenfunktionen und weitere Fehler verhalten. Zusammenfassend können wir sagen, dass unter Berücksichtigung der Hyperparameter und Trainingszeit der neuronalen Netze die Ergebnisse der numerischen Verfahren, namentlich dem Runge-Kutta-Verfahren und dem BDF-Verfahren, den Approximationen der neuronalen Netze weit überlegen sind. Sofern wir eine exakte Lösung gegeben haben, können wir die globalen Fehler beider Verfahren vergleichen und sehen, dass der numerische Fehler um einige Größenordnungen kleiner ist als der Fehler des neuronalen Netzes. Auch ohne exakte Lösung können wir unter anderem die Trajektorien der berechneten Lösungen vergleichen, wobei auch hier die Trajektorie des Runge-Kutta-Verfahrens viel näher an dem Vergleichswert liegt, als die des neuronalen Netzes. Außerdem sehen wir, dass die Wahl der Schichtanzahl und Größe abhängig von der Komplexität und Form des gegebenen Systems ist. Deshalb ist die Wahl der Hyperparameter nicht trivial und wird hauptsächlich durch Ausprobieren ermittelt.

Der Fokus liegt also auf dem Trainingsprozess der neuronalen Netze, denn sofern dieser optimiert wird, gibt einen nennenswerten Vorteil für das Approximieren durch neuronale Netze. Dieses kann zur Approximation anderer Anfangswertprobleme, dessen Parameter in den Intervallen der Lösungspakete liegen, genutzt werden, da die Gewichte durch die zufällige Wahl der Trainingsdaten auf viele Kombinationen von Anfangswertproblemen angepasst wurden. Ein numerisches Lösen aller dieser Kombinationen kann je nach vorausgesetzter Genauigkeit zu größerem Rechenaufwand führen, als die Auswertung des neuronalen Netzes, das durch eine Verkettung von Matrix-Vektor-

Multiplikationen dargestellt ist. Des Weiteren ist es möglich, dass unter Berücksichtigung des gegebenen Fehlers eines neuronalen Netzes, eine beliebige Zeitauswertung  $t_i \in [t_0, t_f]$  der approximierten Lösung möglich ist. Im Gegenzug dazu ist eine Zeitauswertung für numerische Verfahren nicht möglich, da für den gesuchten Wert  $u_i$  die Werte  $u_0, \dots, u_{i-1}$  zuerst berechnet werden. Je nach vorausgesetzter Genauigkeit, Größe des Zeitintervalls und verwendetes numerisches Verfahren kann diese Berechnung große Rechenaufwände aufweisen, während das neuronale Netz lediglich Matrix-Vektor-Multiplikationen durchführt. Der geringe Rechenaufwand hat den Vorteil, dass zur Berechnung hochkomplexe, aufwendige Anfangswertprobleme mit Computern ohne hohe Rechenleistung, also auch kleine Prozessoren die beispielsweise in Drohnen verbaut werden, approximiert werden können [10, S. 9].

In Zukunft kann die Methode der Lösungspakete also Anwendungsbereiche finden, da mit steigender Rechenleistung sich auch die Ergebnisse der Approximationen verbessern, welche dann wiederum gespeichert, verbreitet und für ähnliche Anfangswertprobleme verwendet werden können.



## Literatur

- [1] J. F. Sundermeier: *Der Fixpunktsatz von Schauder*.
- [2] Harro Heuser: *Gewöhnliche Differentialgleichungen*. 6. Auflage. ISBN: 978-3-8348-0705-2.
- [3] Lisa Beck: *Gewöhnliche Differentialgleichungen*. 12. Feb. 2018.
- [4] *Matrixexponential*. URL: <https://www.biancahoegel.de/mathe/analysis/matrixexponential.html> [17. Feb. 2022].
- [5] P. Deuffhard, F. Bornemann: *Numerische Mathematik 2 Gewöhnliche Differentialgleichungen*. 3. Auflage. Berlin, New York: Walter de Gruyter.
- [6] Guido Walz: *Lexikon der Mathematik*. 1. Bd. 1. Spektrum Akademischer Verlag. ISBN: 3-8274-0439-8.
- [7] P. Deuffhard, A. Hohmann: *Numerische Mathematik 1*. 5. Auflage. De Gruyter, 17. Dez. 2018.
- [8] Tatjana Stykel: *Skript zur Vorlesung Numerik Gewöhnlicher Differentialgleichungen*. Sommersemester 2020.
- [9] Ovidiu Calin: *Deep Learning Architectures A Mathematical Approach*. Springer. ISBN: 978-3-030-36720-6.
- [10] Cedric Flamant, Pavlos Protopapas und David Sondak: „Solving Differential Equations Using Neural Network Solution Bundles“. 16. Juni 2020. arXiv: 2006.14372 [physics]. URL: <http://arxiv.org/abs/2006.14372> [15. Feb. 2022].
- [11] Yoshua Bengio u. a.: „Curriculum Learning“. In: *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*. The 26th Annual International Conference. Montreal, Quebec, Canada: ACM Press, 2009, S. 1–8. ISBN: 978-1-60558-516-1. URL: <http://portal.acm.org/citation.cfm?doid=1553374.1553380> [19. Feb. 2022].
- [12] Remco van der Meer: *Solving Partial Differential Equations with Neural Networks*. URL: <https://repository.tudelft.nl/islandora/object/uuid:c77e1bcc-7212-4234-af34-6586b628ab1c/datastream/OBJ/download> [24. März 2022].
- [13] *Pandas.DataFrame.Mean — Pandas 1.4.2 Documentation*. URL: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mean.html> [7. Apr. 2022].
- [14] *Scipy.Integrate.RK45 — SciPy v1.8.0 Manual*. URL: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.RK45.html> [1. Apr. 2022].
- [15] *Scipy.Integrate.BDF — SciPy v1.8.0 Manual*. URL: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.BDF.html> [1. Apr. 2022].

## Abbildungsverzeichnis

1	Neuronales Netzwerk mit 3 Eingabeneuronen (grün), Bias (gelb), zwei versteckte Schichten (blau) und 2 Ausgabeneuronen (rot). . . . .	29
2	Graphen der Sigmoid-Aktivierungsfunktion $\Psi(x) = \frac{1}{1+e^{-x}}$ und dessen Ableitung. .	33
3	Diagramm eines Rebound Pendels[10, S. 6] . . . . .	39
4	Trajektorie der verschiedenen Lösungen. . . . .	41
5	$(t, \theta(t))$ Graph der berechneten Lösungen. . . . .	42
6	Kostenfunktion des neuronalen Netzes bei steigender Epochenanzahl. . . . .	42
7	Trajektorien der verschiedenen Lösungen. . . . .	45
8	Globaler Fehler der jeweiligen Verfahren in Abhängigkeit der Zeit. . . . .	45
9	Globaler Fehler für steigende Epochen. . . . .	46
10	Kostenfunktion für steigende Epochen. . . . .	46
11	Kostenfunktion der verschiedenen neuronalen Netze. . . . .	49
12	Globaler Fehler der verschiedenen neuronalen Netze. . . . .	49
13	Globaler Fehler von Netzwerk 4 und dem Runge-Kutta-Verfahren in Abhängigkeit der Zeit. . . . .	50
14	Trajektorien der von den verschiedenen neuronalen Netzen approximierte Lösung, der Runge-Kutta-Lösung und der exakten Lösung. . . . .	50
15	$(t, x(t))$ Graph der Lösungen des Runge-Kutta-Verfahrens, der Approximation der Lösungspakete und der exakten Lösung. . . . .	51
16	Kostenfunktion der verschiedenen neuronalen Netze. . . . .	52
17	Globaler Fehler der verschiedenen neuronalen Netze. . . . .	52
18	Globaler Fehler von Netzwerk 2 und dem Runge-Kutta-Verfahren in Abhängigkeit der Zeit. . . . .	53
19	Trajektorien der von den verschiedenen neuronalen Netzen approximierte Lösung, der Runge-Kutta-Lösung und der exakten Lösung. . . . .	53
20	$(t, x(t))$ -Graph der von den verschiedenen neuronalen Netzen approximierte Lösung, der Runge-Kutta-Lösung und der exakten Lösung. . . . .	54