HSBI Bielefeld

University of Applied Sciences

Fachbereich Ingenieurwissenschaften und Mathematik

Studiengang Optimierung und Simulation

Lösen von nichtlinearen Gleichungssystemen mit einem Reinforcement-Learning-Agent

Bericht

Vorgelegt von:	Nicolas Schneider
Matrikelnummer:	1208960
Studiengang:	Optimierung und Simulation
Abgabedatum:	07.04.2024
Betreuer:	Prof. Dr. rer. nat. Bernhard Bachmann

Abstract

Nichtlineare Gleichungssysteme (NGS) spielen eine zentrale Rolle in vielen Bereichen der Wissenschaft und Technik, da sie zur Modellierung und Lösung komplexer Probleme in Physik, Chemie, Ingenieurwesen und anderen Disziplinen eingesetzt werden. Trotz ihrer Bedeutung stellt die Lösung von NGS aufgrund ihrer inhärenten Nichtlinearität und des Fehlens geschlossener analytischer Lösungen eine große Herausforderung dar. Traditionelle numerische Verfahren wie das Newton-Raphson-Verfahren oder Optimierungsansätze stoßen oft an ihre Grenzen, insbesondere bei hochdimensionalen Problemen, chaotischem Verhalten oder starker Abhängigkeit von Anfangsbedingungen.

In jüngster Zeit hat der Bereich des Reinforcement learnings (RL) zunehmend an Bedeutung gewonnen und vielversprechende Ergebnisse bei der Lösung komplexer Probleme geliefert. RL-Agenten lernen durch Interaktion mit einer Umgebung und Belohnungssignale, optimale Strategien zu entwickeln, ohne explizite Programmierung. Dieser Ansatz hat sich in verschiedenen Anwendungsfeldern wie Robotik, Spielen und Optimierungsproblemen als erfolgreich erwiesen.

In dieser Arbeit wird der Ansatz des RL verwendet, um Lösungen für nichtlineare Gleichungssysteme zu finden. Ein RL-Agent wird in einer maßgeschneiderten Umgebung trainiert, die die Struktur des gegebenen nichtlinearen Gleichungssystems widerspiegelt. Durch die Formulierung von Belohnungen für Aktionen, die den Agenten näher an eine Lösung führen, wird dieser befähigt, iterative Strategien zur effizienten Lösungsfindung zu erlernen.

Die vorliegende Arbeit präsentiert einen initiierenden Ansatz und analysiert seine Leistung hinsichtlich Schnelligkeit und Genauigkeit bei der Lösungsfindung von nichtlinearen Gleichungssystemen. Dabei wird eine erste Untersuchung durchgeführt, um die Effektivität dieses Ansatzes im Vergleich zu etablierten Methoden wie dem Newton-Raphson-Verfahren zu bewerten. Besonderes Augenmerk wird auf die Identifizierung und Analyse der limitierenden Faktoren dieses Ansatzes gelegt, um potenzielle Schwächen aufzudecken und zu verstehen, inwiefern dieser Ansatz für praktische Anwendungen geeignet ist.

Inhaltsverzeichnis

1 Grundlagen nichtlinearer Gleichungssysteme

Ein nichtlineares Gleichungssystem kann als Nullstellenproblem wie folgt formuliert werden:

Definition 1.1 (Nichtlineares Gleichungssystem):
Es sei $B \subset \mathbb{R}^n$ und $\mathbf{g} : B \to \mathbb{R}^n$. Gesucht sind Lösungen von

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}.$$

$\mathbf{f}(\mathbf{x}) = \mathbf{0}$ ist ein System von n nichtlinearen Gleichungen für n Unbekannte $x_1, \ldots, x_n$.

$$\begin{array}{rcl} f_1(x_1, \ldots, x_n) & = & 0 \\ f_2(x_1, \ldots, x_n) & = & 0 \\ \vdots & & \\ f_n(x_1, \ldots, x_n) & = & 0 \end{array}$$

[4]

Nichtlineare Gleichungssysteme treten in vielen Bereichen der Wissenschaft und Technik auf, wie beispielsweise in der Physik, Chemie, Biologie, Wirtschaftswissenschaften und Ingenieurwissenschaften. Sie dienen zur Modellierung und Analyse komplexer Systeme, deren Verhalten durch nichtlineare Beziehungen zwischen den Variablen beschrieben wird.
Die Lösung solcher Systeme ist jedoch oft eine große Herausforderung, da nichtlineare Gleichungen im Allgemeinen keine geschlossenen analytischen Lösungen besitzen und zu den NP-schweren Probleme zählen. Stattdessen müssen numerische Verfahren eingesetzt werden, um approximative Lösungen zu finden. Einige gängige Methoden zur Lösung nichtlinearer Gleichungssysteme sind:

1. **Iterative Verfahren**: Hierzu zählen Methoden wie das Newton-Raphson-Verfahren, das Quasi-Newton-Verfahren und das Broyden-Verfahren. Diese Verfahren starten mit einer Anfangsschätzung und verbessern diese iterativ, bis eine ausreichend genaue Lösung gefunden ist.

2. **Globale Optimierungsverfahren**: Wenn das Gleichungssystem als Optimierungsproblem formuliert werden kann, können globale Optimierungsverfahren wie die Branch-and-Bound-Methode oder evolutionäre Algorithmen zur Lösung eingesetzt werden.

Die Schwierigkeit, nichtlineare Gleichungssysteme zu lösen, besteht oft darin, geeignete Anfangsschätzungen für die Iterationsverfahren zu finden und die Konvergenz der Verfahren sicherzustellen. Viele Systeme weisen mehrere Lösungen auf, von denen einige instabil oder nicht physikalisch sinnvoll sein können. Darüber hinaus können nichtlineare Systeme eine komplexe Struktur mit mehreren lokalen Extrema aufweisen, was die globale Konvergenz erschwert. (vgl. [3])

2 Grundlagen Reinforcement Learning

Reinforcement learning (RL) ist ein Teilgebiet des Maschinellen Lernens, bei dem ein Agent lernt, durch Interaktion mit einer Umgebung eine bestimmte Aufgabe oder ein Ziel zu erreichen. Im Gegensatz zu überwachtem Lernen, bei dem ein Modell anhand von Trainingsbeispielen mit bekannten Eingabe-Ausgabe-Paaren gelernt wird, erhält der Agent beim Reinforcement learning nur eine skalare Bewertung (Reward) für seine Aktionen.

Durch Ausprobieren und Lernen aus den erhaltenen Rewards versucht der Agent, eine Strategie (Policy) zu finden, die die kumulative Belohnung über die Zeit maximiert.

Markov-Entscheidungsprozess:

Reinforcement Learning Probleme werden häufig als Markov-Entscheidungsprozesse (Markov Decision Processes, MDPs) formuliert. Ein MDP besteht aus:

- Einem Zustandsraum $\mathcal{S}$, der alle möglichen Zustände der Umgebung enthält.
- Einem Aktionsraum $\mathcal{A}$, der alle möglichen Aktionen des Agenten definiert.
- Einer Übergangswahrscheinlichkeitsfunktion $\mathcal{P}(s, a, s')$, die die Wahrscheinlichkeit angibt, dass der Agent beim Ausführen der Aktion a im Zustand s in den Zustand s' übergeht.
- Einer Belohnungsfunktion $\mathcal{R}(s, a, s')$, die die Belohnung definiert, die der Agent erhält, wenn er aus dem Zustand s durch Ausführen der Aktion a in den Zustand s' übergeht.

Das Ziel des Agenten ist es, eine Policy $\pi : \mathcal{S} \to \mathcal{A}$ zu finden, die die erwartete kumulative Belohnung über die Zeit maximiert.
Abbildung 1 stellt den Lernprozess eines Agenten in einer Umgebung vereinfacht dar.

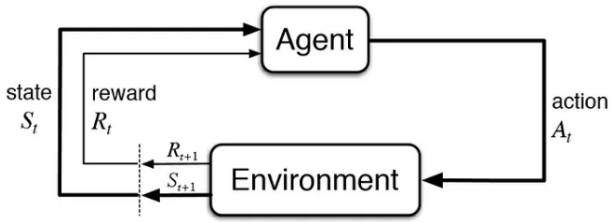

Abbildung 1: Vereinfachter Lernprozess eines Agenten in einer Umgebung[1]

Value Functions und Bellman-Gleichungen:

Eine zentrale Rolle beim Reinforcement Learning spielen die Value Functions, die den erwarteten kumulativen Reward für einen gegebenen Zustand oder eine Zustand-Aktions-Paar angeben. Es gibt zwei wichtige Value Functions:

- Die State-Value Function $V^\pi(s)$ gibt den erwarteten kumulativen Reward an, wenn der Agent im Zustand s startet und der Policy π folgt.
- Die Action-Value Function $Q^\pi(s, a)$ gibt den erwarteten kumulativen Reward an, wenn der Agent im Zustand s startet, die Aktion a ausführt und danach der Policy π folgt.

Die Value Functions erfüllen die Bellman-Gleichungen, die eine rekursive Beziehung zwischen den Value Functions benachbarter Zustände herstellen. Für die State-Value Function lautet die Bellman-Gleichung:

$$V^\pi(s) = \mathbb{E}_\pi \left[R(s, a, s') + \gamma V^\pi(s') \right] \tag{1}$$

Und für die Action-Value Function:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[R(s, a, s') + \gamma \sum_{s'} \mathcal{P}(s, a, s') V^\pi(s') \right] \tag{2}$$

[1] https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

Hier ist $\gamma \in [0, 1]$ der Diskontierungsfaktor, der bestimmt, wie viel Gewicht zukünftigen Rewards beigemessen wird. Ein Wert von γ nahe 0 bedeutet, dass der Agent nur die unmittelbaren Rewards optimiert, während ein Wert nahe 1 längerfristige Belohnungen stärker gewichtet. (vgl. [2])

Dieses Kapitel bietet eine grundlegende Einführung in das Reinforcement Learning (RL), einem Teilgebiet des Maschinellen Lernens. RL ermöglicht es einem Agenten, durch Interaktion mit seiner Umgebung eine bestimmte Aufgabe zu erlernen, indem er Belohnungen für seine Aktionen maximiert, ohne explizite Trainingsdaten zu erhalten.
Der Markov-Entscheidungsprozess (MDP) bildet den Kern eines RL-Problems, definiert durch einen Zustandsraum, einen Aktionsraum, eine Übergangsfunktion und eine Belohnungsfunktion. Dabei spielen Value Functions und Bellman-Gleichungen eine entscheidende Rolle, um den erwarteten kumulativen Reward zu bewerten.
Es ist wichtig zu betonen, dass die Umsetzung eines RL-Agenten stark vom Anwendungsfall und den verfügbaren Daten abhängt, was zu einer großen Variabilität in der Implementierung führt. Dieses grundlegende Verständnis dient als Ausgangspunkt für die weiterführende Diskussion darüber, wie RL bei der Lösung nichtlinearer Gleichungssysteme eingesetzt werden kann.

3 Reinforcement Learning für die Lösung nichtlinearer Gleichungssysteme

Bevor ein RL-Agent nichtlineare Gleichungssysteme lösen kann, sollten zunächst einige wichtige Fragen beantwortet werden, die für die Qualität der Lösung und die Lösungsfindung von Relevanz sind:

1. **Umgebung:** Was ist die Umgebung, in dem der Agent agiert?
2. **Zustand:** Was sind zulässige und sinnvolle Zustände?
3. **Aktionen:** Was sind zulässige und sinnvolle Aktionen, die der Agent wählen kann?
4. **Belohnung:** Wie sieht eine zielführende Belohnungsfunktion aus?

Diese Projektarbeit fokussiert sich auf einen ersten Ansatz, der hinsichtlich seiner Effektivität untersucht wird.
Für die oben genannten Fragen wurden folgende erste Lösungsansätze entwickelt:

Umgebung:

Im Kontext dieses Projekts wird die Umgebung als der Raum betrachtet, in dem sich die Gleichungen befinden. Ein besonderes Merkmal dabei ist, dass die Umgebung als statisch betrachtet werden kann, sofern sich die Gleichungen und damit einhergehend auch die Lösungen während des Lösungsprozesses nicht ändern.

Zustand:

Der Zustand wird aufgefasst, als die gewählte Aktion.

Aktion:

Abhängig von der Dimensionalität des Problems kann der Agent aus einem kontinuierlichen Raum von Punkten wählen. Der Agent wählt eine Aktion $a \in \mathcal{A}$, wobei $\mathcal{A} \subseteq \mathbb{R}^{n-1}$ und n die Dimensionalität des Problems

widerspiegelt.

Belohnungsfunktion:

Die Formulierung der Belohnungsfunktion konzentriert sich darauf, die Lösung(en) des Gleichungssystems zu finden, sofern sie denn existieren. In dieser Arbeit wird das Residuum verwendet, welches den Fehler zur eigentlichen Lösung darstellt. Das Residuum R an der Optimallösung $\mathbf{x}^*$ ist $\mathbf{0}$, sodass das Ziel dabei ist diesen Fehler zu minimieren.

Definition 3.1 (Residuum):
Gegeben seien n nichtlineare Gleichungen mit $\mathbb{R}^n \to \mathbb{R}^n$. Es sei vorausgesetzt, dass mindestens eine Lösung $\mathbf{x}^*$ existiert, die das Gleichungssystem löst. Dann ist das Residuum an der Stelle $\mathbf{x}$ mit $\mathbf{x} \in \mathbb{R}^n$ definiert als

$$R(\mathbf{x}) = \sum (f_i(\mathbf{x}) - f_j(\mathbf{x}))^2 \quad \forall 1 \leq i < j \leq n. \tag{3}$$

Da der Agent das Ziel verfolgt die Belohnung über die Zeit zu maximieren, kann das negative Residuum als Belohnung interpretiert werden. Es gilt also: $Reward = -R(\mathbf{x})$.

Dieses Kapitel soll die grundsätzlichen Fragen beantworten, die bei der Kombination von RL und nichtlinearen Gleichungssystemen auftreten können. Diese müssen vor der tatsächlichen mit Bedacht beantwortet werden, sodass eine zielführende Lösungsfindung garantiert ist.
Besonderes Augenmerk gilt dabei der Umgebung, dem Zustands- sowie Aktionsraum und der Belohnungsfunktion des Agenten.
Da die Grundlagen für nichtlineare Gleichungssysteme, sowie für RL gelegt worden sind, wird im nachfolgenden Kapitel der Frage nachgegangen, wie diese Problemstellung genau umgesetzt werden kann.

4 Umsetzung

Bevor mit der Implementierung der Problemstellung, der Umgebung und des Agents begonnen wird, ist es von entscheidender Bedeutung, die Auswahl der richtigen Programmiersprache und Bibliothek sorgfältig zu prüfen. Diese Entscheidung bildet das Fundament für das gesamte Projekt und kann einen erheblichen Einfluss auf die Effizienz, Skalierbarkeit und den Erfolg der Umsetzung haben.

Für dieses Projekt wurde die Programmiersprache *Python* als Programmiersprache gewählt, zusammen mit der Reinforcement-Learning-Bibliothek *stable-baselines3*. Diese Entscheidung basiert darauf, dass *stable-baselines3* eine breite Palette von wichtigen RL-Algorithmen implementiert und gleichzeitig eine benutzerfreundliche Integration mit *Python* bietet. Außerdem bietet diese Bibliothek eine ausgezeichnete Kompatibilität mit der Programm-Bibliothek *gymnasium* von OpenAI, die essenziell für Erstellung von RL-Umgebungen ist. Die Kombination aus *Python*, *stable-baselines3* und *gymnasium* ermöglicht eine effiziente Entwicklung und Umsetzung von Reinforcement-Learning-Algorithmen für die gegebenen Anforderungen des Projekts (vgl. [5], vgl. [7]). Der Programm-Code kann im folgenden Repository eingesehen werden: `https://github.com/Nicolas2912/OuS_Projekt`.

Für die Umsetzung des Agenten bzw. der Umgebung sind folgende Methoden für die Problemstellung relevant:

- `get_distance(x)`: Berechnet das Residuum für einen Vektor $\mathbf{x} \in \mathbb{R}^{n-1}$.
- `reward(residuum, *args)`: Berechnet den Reward basierend auf dem Residuum und weiteren zusätzlichen Parametern.

- `step(action)`: Führt einen Schritt während des Lernens in der Umgebung aus und aktualisiert dabei die Belohnung auf Grundlage verschiedener, vorher festgelegter Regeln.
- `reset()`: Setzt die Umgebung auf einen zufälligen Zustand zurück, sobald eine bestimmte Bedingung erfüllt ist.

Auch der RL-Algorithmus ist entscheidend für die Lösungsfindung, da er die Strategie bestimmt, nach der der Agent in der Umgebung handelt und lernt. Im Verlauf des Projekts wurden verschiedene Algorithmen aus der *stable-baselines3*-Bibliothek getestet. Unter diesen hat sich der *Proximal Policy Algorithmus* (PPO) als besonders geeignet für diese spezifische Problemstellung erwiesen. Für eine detaillierte Beschreibung dieses Algorithmus sei auf [6] verwiesen.

Umgebung:

Neben vielen vorgegebenen Umgebungen, die in der Programm-Bibliothek *gymnasium* implementiert sind wie z.B. *Cart Pole*, *Acrobot* oder *Pendulum* ist es auch möglich eine eigene Umgebung für einen speziellen Anwendungsfall zu erstellen. Für die Problemstellung nichtlineare Gleichungssysteme mit einem RL-Agent zu lösen, wird eine eigene Umgebung erstellt.
Nachfolgend ist eine verkürzte Initialisierung der Umgebung dargestellt.

Listing 1: Initialisierung der Umgebung

```
1  class CustomEnv(gym.Env):
2      def __init__(self, dimension):
3          super(CustomEnv, self).__init__()
4          self.dimension = dimension
5          self.low_bounds = -5.0 * np.ones(dimension)
6          self.high_bounds = 5.0 * np.ones(dimension)
7          self.action_space = gym.spaces.Box(low=self.low_bounds, high=self.
               high_bounds, dtype=np.float64)
8          self.observation_space = gym.spaces.Box(low=-np.infty, high=np.
               infty, shape=(dimension,))
9          self.state = np.array([random.uniform(self.x_min, self.x_max),
               random.uniform(self.y_min, self.y_max)] + [0.0] * dimension)
```

Bei der Initialisierung der Umgebung muss der Bereich der möglichen Aktionen (`action_space`) angegeben werden. Da mögliche Lösungen für ein nichtlineares Gleichungssystem sich im ganzen Raum $\mathbb{R}^n$ befinden können, bietet sich dieser auch für die möglichen Aktionen an. Jedoch ist es nicht möglich den ganzen Raum (von $-\infty$ bis ∞) anzugeben, sodass Grenzen gesetzt werden müssen. Diese sind im Code mit den Variablen `self.low_bounds` und `self.high_bounds` beschrieben. Hierbei zeichnet sich auch einer der Nachteile ab. Da bei den allermeisten nichtlinearen Gleichungssystemen von Beginn an nicht abzuschätzen ist, in welchem Teil des Lösungsraumes sich die Lösung(en) befindet/befinden, stellt die Wahl des Aktionsraumes eine Herausforderung dar. Ähnlich wie beim Newton-Raphson-Verfahren, bei dem ein initialer Startpunkt notwendig ist, der die Laufzeit des Verfahrens maßgeblich beeinflussen kann, so kann auch die Größe des Aktionsraumes die Laufzeit der Lösungsfindung maßgeblich beeinflussen.

Berechnung des Residuums (Distanz):

Da die Qualität der aktuellen Lösung im Projektcode anhand des Residuums bestimmt wird, ist es wichtig, den Prozess der Residuum-Berechnung kurz darzustellen.

Für manche Arten von Problemen hat es sich als vorteilhaft erwiesen, das Residuum auf eine festgelegte Skala zu skalieren (bspw. logarithmische Skalierung, Min-Max-Skalierung etc.). Im folgenden ist ein Ausschnitt aus der Methode zur Berechnung des Residuums zu sehen.

Listing 2: Berechnung des Residuums

```
def get_distance(point):
    scaling = "logarithmic"
    equations = self.nse()

    # Calculate the function values for all equations
    values = np.array([eq(point) for eq in equations])

    # calculate normal residuum
    normal_res = sum((values[i] - values[j]) ** 2 for i in range(len(values
        )) for j in range(i + 1, len(values)))
    self.all_residuals.append(normal_res)

    if scaling == "logarithmic":
        scaled_values = np.log1p(np.abs(values)) / np.log(10)
        res = sum((scaled_values[i] - scaled_values[j]) ** 2 for i in range
            (len(scaled_values)) for j in
        range(i + 1, len(scaled_values)))
        return res, normal_res
```

Ausführung eines Lernschritts:

Die Ausführung eines Lernschritts ist entscheidend für die Lösungsfindung und ist genauer in Algorithmus 1 beschrieben.
Zuerst seien die wichtigsten Variablen in Tabelle 1 definiert.

Tabelle 1: Variablendefinitionen

Variable	Beschreibung
distances	Eine Liste, die Entfernungen (Residuen) enthält.
best_actions	Eine Liste, die die besten Aktionen enthält.
best_distances	Eine Liste, die die besten erreichten Entfernungen enthält.
improvement_rate_history	Eine Liste, die Verbesserungsraten enthält.
good_points_threshold	Der Schwellenwert für einen „guten Punkt“.
no_improvement_limit	Die Grenze für aufeinanderfolgende Iterationen ohne Verbesserung.
dynamic_reward	Die dynamisch akkumulierte Belohnung.
dynamic_penalty	Die dynamisch akkumulierte Strafe.
total_reward	Die insgesamt akkumulierte Belohnung.
improvement_rate	Die berechnete Verbesserungsrate.
done	Ein boolescher Wert, der den Abschluss anzeigt.
consecutive_no_improvement	Die Anzahl aufeinanderfolgender Iterationen ohne Verbesserung.

Beschreibung:

1. **Initialisierung:**
   - Die Variable `residuum` wird durch `get_distance(action)` initialisiert, wobei `action` die aktuelle Aktion darstellt.
   - Die Variable `reward` wird als das Negative des Residuums initialisiert, um eine Strafe für ungünstige Zustände zu erhalten.
   - Die dynamisch akkumulierte Belohnung (`dynamic_reward`) und Strafe (`dynamic_penalty`) sowie die Gesamtbelohnung (`total_reward`) werden auf Null gesetzt.
   - Der Parameter `no_improvement_limit` wird auf 50 festgelegt, um die Anzahl aufeinanderfolgender Iterationen ohne Verbesserung zu begrenzen.
   - Die Variable `done` wird auf `False` gesetzt, um den Algorithmus als nicht abgeschlossen zu markieren.
2. **Überprüfung der Bedingungen:**
   - Wenn die Länge der Liste `distances` größer als eins ist und das aktuelle Residuum kleiner als das vorherige Residuum ist, wird eine Belohnung für die Verbesserung berechnet und zu `dynamic_reward` addiert.
   - Wenn die Länge der Liste `best_actions` größer als eins ist und das aktuelle Residuum größer als das vorherige beste Residuum ist, wird eine Strafe für die Verschlechterung berechnet und zu `dynamic_penalty` addiert.
3. **Aktualisierung der Belohnung und Strafe:**
   - Die dynamisch akkumulierte Belohnung (`dynamic_reward`) und Strafe (`dynamic_penalty`) werden zu `reward` und `penalty` addiert.
4. **Aktualisierung der Verbesserungsrate und des Schwellenwerts für gute Punkte:**
   - Die Verbesserungsrate wird berechnet und zur Verbesserungsraten-Historie (`improvement_rate_history`) hinzugefügt.
   - Der Schwellenwert für gute Punkte (`good_points_thrs`) wird entsprechend der aktuellen Verbesserungsrate angepasst.
   - Die Grenze für aufeinanderfolgende Nichtverbesserungen (`no_improvement_limit`) wird basierend auf der Verbesserungsrate aktualisiert.
5. **Überprüfung der Terminierungsbedingung:**
   - Wenn das aktuelle Residuum kleiner oder gleich dem Schwellenwert für gute Punkte ist, wird der Algorithmus als abgeschlossen markiert und die Anzahl aufeinanderfolgender Nichtverbesserungen wird auf null zurückgesetzt.
   - Andernfalls wird die Anzahl aufeinanderfolgender Nichtverbesserungen inkrementiert, und wenn diese die Grenze erreicht oder überschreitet, wird der Algorithmus ebenfalls als abgeschlossen markiert und die Belohnung auf null gesetzt.
6. **Belohnungsformung:**

Algorithm 1: Ausschnitt zur Ausführung eines Schrittes

Input: action
Output: reward, done

Initialization:
residuum = get_distance(action), reward = −residuum, dynamic_reward = dynamic_penalty = 0, total_reward = 0, no_improvement_limit = 50, done = False;
# Belohnung, wenn die derzeitige Aktion besser als die vorherige ist
if *len*(*distances*) > 1 ***and*** *residuum* < *distances[n-1]* **then**
    residuum_difference_reward = distances[n-1] − residuum
    dynamic_reward = dynamic_reward + residuum_difference_reward · (distances[n-1] − residuum)
end
# Bestrafung, wenn die Aktion schlechter als die vorherige ist
if *len*(*best_actions*) > 1 ***and*** *residuum* > *best_distances[-2]* **then**
    residuum_difference_penalty = residuum − distances[n-1]
    dynamic_penalty = dynamic_penalty · (residuum − distances[n-1])
end
reward = reward + dynamic_reward
penalty = penalty + dynamic_penalty
# Berechnet die Verbesserungsrate, passt den Schwellenwert für gute Punkte an und setzt das Limit für aufeinanderfolgende Nichtverbesserungen.
if *len*(*best_distances*) > 1 **then**
    $\text{improvement\_rate} = \frac{\text{distances[n-1]} - \text{distances[n]}}{\text{distances[n-1]}}$
    append improvement_rate to improvement_rate_history
    good_points_thrs = $\max\{0.9, \text{good\_points\_thrs} \cdot (1 - \text{improvement\_rate})\}$
    no_improvement_limit = $\max\{50, 100 \cdot \text{improvement\_rate}\}$
end
else
    no_improvement_rate = 50
end
# Algorithmus terminiert, wenn Residuum besser als Schwellenwert
if *residuum* ≤ *good_points_threshold* **then**
    done = True
    consecutive_no_improvement = 0
end
else
    consecutive_no_improvement = consecutive_no_improvement +1
    if *consecutive_no_improvement* ≥ *no_improvement_limit* **then**
        done = True
        reward = 0
    end
end
# Zusätzliche Belohnung, wenn die Verbesserungsrate besser ist als 0.01
if *improvement_rate* > 0.01 **then**
    reward = reward +1
end
return state, reward, done

- Eine zusätzliche Belohnung wird vergeben, wenn die Verbesserungsrate größer als 0,01 ist.

7. **Rückgabe:** Die Zustandsinformation, die Belohnung und der Abschlussstatus werden zurückgegeben.

Folgende nichtlineare Gleichungssysteme wurden für die Auswertung des Verfahrens herangezogen:
Eindimensionales nichtlineares Gleichungssystem:

$$x^5 - 3x^4 + x^3 + 0.5x^2 = 0 \tag{4a}$$
$$\sin(2x) = 0 \tag{4b}$$

Zweidimensionale Rosenbrock-Funktion:

$$10(x_2 - x_1^2) = 0 \tag{5a}$$
$$1 - x_1 = 0 \tag{5b}$$

n-dimensionales *Economics Modeling Application*-Problem:

$$\left(x_k + \sum_{i=1}^{n-k-1} x_i x_{i+k}\right) x_n - c_k = 0 \quad 1 \leq k \leq n-1 \tag{6a}$$
$$\sum_{l=1}^{n-1} x_l + 1 = 0 \tag{6b}$$

Die globalen Optima der jeweiligen Gleichungssystem sind:
Nichtlineares Gleichungssystem 4:

$$x_1^* = \{-0.6998397867\},\ x_2^* = \{0\},\ x_3^* = \{2.4936955491\}$$

Zweidimensionale Rosenbrock-Funktion 5:

$$x^* = \{1, 1\}$$

n-dimensionales *Economics Modeling Application*-Problem (für $n = 10$ und $c_k = 0$) 6:

$$x^* = \{-1.105, 0.461, 0.939, 0.396, -0.181, -0.459, -0.620, -0.293, -0.138, 0.000\}$$

In diesem Kapitel wurden die wesentlichen Schritte zur Implementierung der Problemstellung beleuchtet, beginnend mit der Auswahl der Programmiersprache und Bibliothek bis hin zur Beschreibung relevanter Methoden für die Umsetzung des Agenten und der Umgebung. Dabei wird *Python* als Programmiersprache gewählt, da sie eine benutzerfreundliche Integration mit der Reinforcement-Learning-Bibliothek „stable-baselines3“ bietet, die eine breite Palette von wichtigen RL-Algorithmen bereitstellt. Die Umgebung wird speziell für die Problemstellung nichtlinearer Gleichungssysteme entwickelt, wobei eine angepasste Initialisierung und Berechnung des Residuums eine zentrale Rolle spielen. Ein Ausschnitt zur Ausführung eines Lernschritts wurde ebenfalls präsentiert, um den Prozess der Lösungsfindung innerhalb der Umgebung zu verdeutlichen.
Im weiteren Verlauf wurden drei nichtlineare Gleichungssysteme vorgestellt, die für die Auswertung des Verfahrens herangezogen wurden: ein eindimensionales nichtlineares Gleichungssystem, die zweidimensionale

Rosenbrock-Funktion und ein n-dimensionales *Economics Modeling Application*-Problem (vgl. [1]). Die globalen Optima der jeweiligen Gleichungssysteme werden ebenfalls präsentiert, um einen Überblick über die erwarteten Lösungen zu geben.
Die Umsetzung dieses Kapitels legt das Fundament für die Experimente und Ergebnisse, in denen die Leistungsfähigkeit des entwickelten RL-Agenten bei der Lösung nichtlinearer Gleichungssysteme bewertet wird. Diese werden im nachfolgenden Kapitel dargestellt.

5 Ergebnisse

Für den Ansatz und für die gegebenen Problemstellungen haben sich folgende Ergebnisse ergeben:

NSE	**Epochs**	**Time** (in s)	**Best Action**	**Best Residuum**	**Best Action (Global Optimum)**	**Best Residuum (Global Optimum)**
NSE 1 (4)	$1,000$	3.507	0.0005	$9.6867e-07$	-	-
	$5,000$	15.117	0.0001	$6.1323e-08$	-	-
	$10,000$	32.300	$-9.2649e-05$	$3.4338e-08$	-	-
	$50,000$	503.706	$-2.2873e-05$	$2.0928e-09$	-	-
NSE 2 (5)	$1,000$	3.0820	$[0.2543, 0.1391]$	$9.1665e-07$	$[0.9751, 1.0427]$	0.0494
	$5,000$	9.8100	$[-1.0716, 1.3556]$	$1.5952e-07$	$[0.9848, 0.9994]$	0.0152
	$10,000$	17.5740	$[0.7799, 0.6300]$	$8.6779e-06$	$[0.9775, 1.0144]$	0.0268
	$50,000$	92.9910	$[-0.0906, 0.1172]$	$7.7158e-09$	$[1.0025, 1.0112]$	0.0115
NSE 3 (6)	$1,000$	3.181	$[-0.3101, -0.4941, ...]$	0.1293	$[-1.4969, 1.237, ...]$	0.9246
	$5,000$	9.702	$[-0.4831, 0.2992, ...]$	0.0652	$[-0.5703, 0.7588, ...]$	0.9094
	$10,000$	17.513	$[-0.9745, -0.3335, ...]$	0.0177	$[-1.0468, 1.275, ...]$	1.1439
	$50,000$	102.152	$[0.0151, 1.1305, ...]$	0.0071	$[-1.2344, 0.3361, ...]$	0.6918

Tabelle 2: Ergebnisse des Agenten für unterschiedliche Problemstellungen

NSE	**Steps**	**Time** (in s)	**Startpunkt** $\mathbf{x_0}$	**Solution**	**Tolerance**
NSE 1 (4)	5	0.005	5.0	2.5777	0.1919
	10	0.007	5.0	2.4937	$7e-15$
	15	0.010	5.0	2.4936	0.0
	20	0.013	5.0	2.4936	0.0
NSE 2 (5)	5	0.004	$[-5.0, 5.0]$	$[1, 1]$	0.0
	10	0.006	$[-5.0, 5.0]$	$[1, 1]$	0.0
	15	0.008	$[-5.0, 5.0]$	$[1, 1]$	0.0
	20	0.010	$[-5.0, 5.0]$	$[1, 1]$	0.0

Tabelle 3: Ergebnisse des Newton-Raphson-Verfahrens für NSE 1 & NSE 2

In den vorliegenden Ergebnissen fällt besonders auf, dass das vermeintlich einfachste Problem (NSE 1, 4) die längste Zeit benötigt, um alle 50.000 Epochen abzuschließen - etwa 500 Sekunden. Im Gegensatz dazu liegt die Dauer für die Absolvierung der gleichen Anzahl von Epochen für die Probleme 5 und 6 in einer Größenordnung von nur knapp 100 Sekunden. Eine mögliche Erklärung dafür liegt darin, dass das Residuum für das erste Problem (NSE 1, 4) sehr schnell in einen extrem kleinen Bereich fällt. Dadurch erfordert selbst eine geringfügige Verbesserung des Ergebnisses bereits sehr kleine Anpassungen in den Aktionen des Agenten. Allerdings könnte der Agent aufgrund von Beschränkungen in seiner Aktionsraumgröße Schwierigkeiten haben, Änderungen im Bereich von $1e-9$ zu erreichen. Dadurch wird möglicherweise keine signifikante

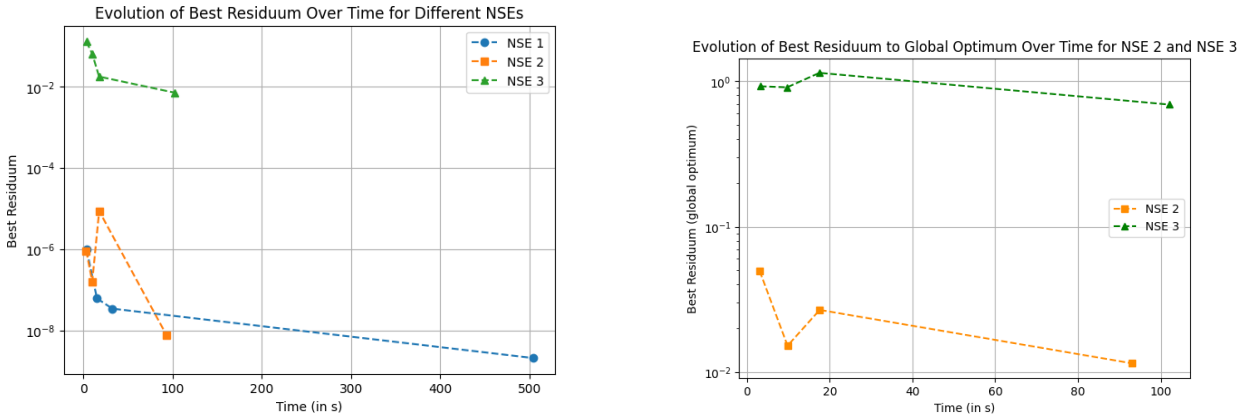

Abbildung 2: Darstellung der besten gefundenen Residuen über die Zeit für alle Probleme und der besten Residuen zum globalen Optimum für NSE 1 & NSE 2

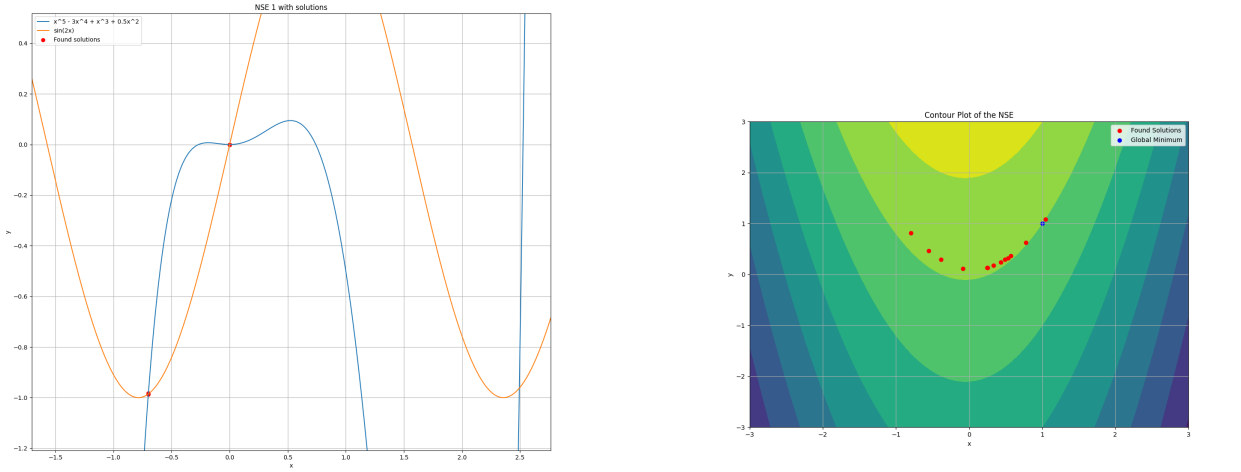

Abbildung 3: Gefundene Lösungen für NSE 1 (mit Schwellenwert $\leq 1e-5$) und NSE 2 (mit Schwellenwert $\leq 1e-3$)

Verbesserung erzielt, sodass die Optimierung stagniert und die Zeit zur Absolvierung der 50000 Epochen zunimmt.
Jedoch wäre dies nur eine anfängliche Idee, die tiefer untersucht werden müsste, da für die Konvergenz des Verfahrens viele weitere Parameter einen Einfluss ausüben.

Bei Problem 1 (NSE 1 4) ist zudem die Besonderheit, dass das nichtlineare Gleichungssystem mehrere globale Optima besitzt (in diesem Fall genau drei). Die meisten Verfahren, wie zum Beispiel das Newton-Raphson-Verfahren benötigen einen anfänglichen Startpunkt, und nähern sich dann iterativ genau **einer** Lösung an, sofern sie denn existiert. Beim vorliegenden Verfahren ist die Besonderheit, dass zum Teil mehrere Lösungen gefunden werden können, da der Agent allein lernt, das negative Residuum (Belohnung) zu maximieren. Da für alle Lösungen gilt, dass das Residuum = 0 ist, können entsprechend auch mehrere Lösungen gleichzeitig mit einer hinreichenden Genauigkeit gefunden werden. In mehreren Testläufen wurde auch die Lösung $x^* \approx -0.6998$ mit einer Genauigkeit, die sich in einer Größenordnung von $1e-7$ befindet, gefunden. In den seltensten Fällen jedoch wurde die Lösung bei $x^* \approx 2.4936$ mit einer Genauigkeit von $\leq 1e-5$ gefunden. Grund dafür ist die große Sensibilität beim Residuum, denn in diesem Bereich weist die Funktion $f(x) = x^5 - 3x^4 + x^3 + 0.5x^2$ eine große Steigung auf, sodass kleine Änderungen im Wert von x hier zu erheblichen Änderungen

im Funktionswert y führen. Dies hat zur Folge, dass die Belohnung für kleinere Änderungen in der Aktion stärker schwankt, sodass diese Aktionen im Mittel nicht vielversprechend genug sind. Der Agent konzentriert sich dann eher auf die Aktionen, die im Mittel eine höhere Belohnung liefern, sodass der Agent diese auch weiter verfolgt und in der näheren Umgebung dieser Aktionen mehr erkundet.
In der Abbildung 4 wird deutlich, wie sich die Belohnung in Abhängigkeit von der gewählten Aktion verhält. Dort ist deutlich zu sehen, dass die Belohnung spärlich um das eigentliche globale Optimum bei $x^* \approx 2.4936$ verteilt ist, sodass der Agent diese Aktion nicht genug bevorzugt und die Lösung nicht mit erhöhter Genauigkeit gefunden werden kann.

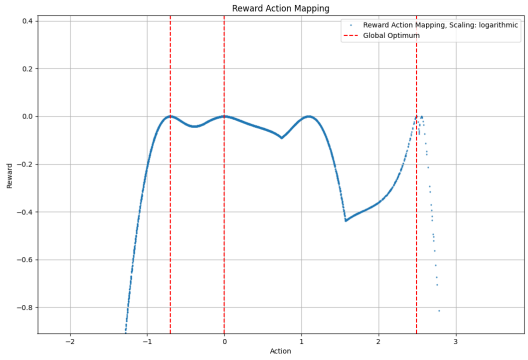

Abbildung 4: Belohnung in Abhängigkeit von der Aktion für NSE 1 (4). Logarithmische Skalierung des Residuums

Des Weiteren ist bei NSE 2 (5) zu erkennen, dass bei einer Epochen-Anzahl von 10000 die beste gefundene Aktion schlechter ist als im Vergleich dazu die beste gefundene Aktion bei einer geringeren Epochen-Anzahl wie 1000.
Die könnte mit der Zufälligkeit des Agenten zusammenhängen, da der Agent zu Beginn zufällig im gegebenen Raum initialisiert wird und auch mit jedem Aufruf der `reset` Methode zufällig zurückgesetzt wird.
Zudem wird auch hier die Problematik des nichtlinearen Gleichungssystems deutlich. Die Rosenbrock-Funktion besteht aus einem „Banana“-förmigen Tal mit mehreren lokalen Optima, sodass viele Optimierungsverfahren Schwierigkeiten haben das globale Optimum zu finden. Von dieser Problematik ist auch dieses Verfahren nicht ausgeschlossen. Der Agent findet zwar Lösungen mit einer Genauigkeit, die sich in einer Größenordnung von $1e-8$ befinden, jedoch sind diese zumeist lokale Optima, die beliebig weit vom globalen Optima entfernt sein können. Aus diesem Grund wurde auch die Spalte *Best Action (Global Optimum)* (mit dem zugehörigen Residuum in der Spalte *Best Residuum (Global Optimum)*) der Tabelle 2 hinzugefügt, um die Aktion zu betrachten, die sich am nächsten zum tatsächlichen globalen Optimum befindet. Werden diese zwei Spalten für die beiden Probleme (5 & 6) betrachtet, lässt sich erkennen, dass das Residuum für diese Aktionen deutlich höher ist als das Residuum für die beste gefundenen Aktion.

Es ist nicht überraschend, dass die Genauigkeit der Lösung für NSE 3 (6) bei gleicher Anzahl an Epochen geringer ist. Dies liegt daran, dass der Aktionsraum in diesem Fall 10-dimensional ist. Der Agent hat daher Schwierigkeiten, die Richtung zu bestimmen, in der sich das Residuum verbessern könnte. Mit steigender Dimension könnte sich diese Problematik weiter verschärfen, sodass der Agent bzw. der grundlegende Algorithmus für diese Problemstellung optimiert werden muss.

Grundlegend bei RL ist vor allen Dingen die „teure“ Berechnung. Da der PPO-Algorithmus verwendet wurde, müssen dabei in jedem Schritt zwei neuronale Netze aktualisiert werden (Actor- und Critic-Netzwerk).

Dies ist insbesondere im Vergleich zum relativ simplen Newton-Raphson-Verfahren sehr teuer und ein nicht zu vernachlässigender Faktor, der sich auf die Zeit auswirkt, bis zu dem eine Lösung mit einer bestimmten Genauigkeit gefunden wird.

In diesem Kapitel wurden die wichtigsten Ergebnisse vorgestellt, die mit dem Ansatz erzielt wurden und untereinander verglichen. Es wurde auf Besonderheiten eingegangen und versucht mögliche Erklärungen zu finden. Abschließend dazu folgt nun das Fazit.

6 Fazit

Als Fazit kann formuliert werden, dass das präsentierte Verfahren grundsätzlich in der Lage ist Lösungen von nichtlinearen Gleichungssystemen zu finden. Besonders gut funktioniert das Verfahren für eindimensionale Probleme, bei denen die Lösungen robust sind, als nicht sensibel auf kleine Änderungen der Aktion reagieren. Anders formuliert funktioniert das Verfahren eben für solche eindimensionale nichtlineare Gleichungssysteme gut, bei denen die Summe der Beträge der Steigungen aller Gleichungen klein ist. Für solche Probleme ist das Verfahren mit dargestellter Umsetzung in der Lage Lösungen mit einer Genauigkeit von $1e-8$ bis zu $1e-9$ zu finden. Vor dem Hintergrund der Zeit und der Berechnungskomplexität allerdings, schließt dieses Verfahren insbesondere im Vergleich zu dem etablierten Newton-Raphson-Verfahren schlecht ab. Dieses Verfahren findet mit weniger Berechnungen und in kürzerer Dauer die Lösung mit einer höheren Genauigkeit. Jedoch sollte dabei auch erwähnt sein, dass dieses Verfahren einen initialen Startpunkt benötigt, von dem das Verfahren sich iterativ der nächsten Lösung annähert. Demzufolge auch nur eine Lösung findet.
Es hängt vom spezifischen Anwendungsfall ab, ob es genügt eine Lösung zu finden, oder mehr. Sollte der Fall vorliegen, dass mehrere Lösungen gefunden werden sollen, so könnte das hier präsentierte Verfahren geeigneter sein als das Newton-Raphson-Verfahren, sofern auch die zeitliche Komponente keine große Rolle spielt. Zusätzlich dazu kann das hier vorgestellte Verfahren möglicherweise eher dazu verwendet werden, eine oder mehrere mögliche Startpunkte für andere iterative Verfahren zu generieren, wenn keinerlei Informationen über die grobe Lage von Lösungen vorliegt. Die iterativen Verfahren könnten im Anschluss daran die bisherigen gefundenen Lösungen nutzen, um diese weiter zu verbessern und schnell eine Lösung größerer Genauigkeit zu finden.

Bei der Anwendung dieses Verfahrens auf die Rosenbrock-Funktion (NSE 2, 5), wurde deutlich, dass auch dieses Verfahren Schwierigkeiten hat, das globale Optimum zuverlässig zu finden. Wie viele andere Optimierungsalgorithmen ist es anfällig für das Feststecken in lokalen Optima, was seine Fähigkeit zur zielgerichteten Suche nach dem globalen Optimum einschränkt. Diese Herausforderung betrifft eine Vielzahl von Optimierungsmethoden und erfordert eine kontinuierliche Verbesserung, um damit umzugehen. Zusätzliche Informationen, die spezifisch für das Problem sind, können helfen, das globale Optimum besser und effizienter zu finden. Allerdings könnten solche Informationen für andere Problemstellungen weniger relevant sein, was ihre breite Anwendbarkeit einschränken könnte.

Insbesondere bei nichtlinearen Gleichungen höherer Dimension deutet sich an, dass die Genauigkeit der gefundenen Lösungen stark abnimmt. Während für NSE 1 und NSE 2 Lösungen mit einer Genauigkeit von $1e-7$ bis $1e-9$ gefunden werden konnten, befand sich die beste gefundene Lösung für NSE 3 (mit einer Dimension von 10) in einer Größenordnung von $1e-3$ bei gleichbleibender Epochen-Anzahl.
Es bleibt zu untersuchen, ob eine generelle Tendenz besteht, dass die Genauigkeit der Lösung mit zunehmender Dimension des Problems abnimmt. Um diese Frage zu beantworten, sind ausführliche Tests und Untersuchungen erforderlich, die eine umfassende Analyse der Leistung des Reinforcement-Learning-Agenten in Bezug auf verschiedene Problemgrößen ermöglichen. Erst nach gründlicher Analyse und Evaluation können

valide Schlussfolgerungen gezogen werden.

Abschließend ist anzumerken, dass neben den bereits diskutierten Parametern eine Vielzahl weiterer Faktoren die Lösungsfindung und die Qualität der Lösung beeinflussen können. Diese zusätzlichen Faktoren können die Effektivität des Verfahrens beeinträchtigen und sollten daher ebenfalls berücksichtigt werden. Dazu gehören:

- Die Größe des Aktionsraumes
- Die Definition des Zustandes
- Die Berechnung und Skalierung des Residuums
- Die Gestaltung der Belohnungsfunktion
- Der zugrunde liegende RL-Algorithmus mit allen zugehörigen beeinflussbaren Hyperparametern, wie Lernrate, Batch-Size usw.

Zusätzlich zu den diskutierten Parametern beeinflussen eine Vielzahl weiterer Faktoren die Lösungsfindung und -qualität. Eine tiefere Analyse des Einflusses dieser Parameter in Bezug auf die spezifische Problemstellung ist daher von entscheidender Bedeutung.
Diese Arbeit bietet somit Einblicke in die Möglichkeiten und Herausforderungen bei der Anwendung von Reinforcement-Learning zur Lösung nichtlinearer Gleichungssysteme und legt den Grundstein für weitere Untersuchungen auf diesem Gebiet.

Literatur

[1] Ajith Abraham Crina Grosan: "A New Approach for Solving Nonlinear Equations Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* (2008).

[2] Zuowen Lia und Shuijia Li: "Solving Nonlinear Equations Systems with an Enhanced Reinforcement Learning Based Differential Evolution". In: 2.1 (März 2022).

[3] Klaus Mainzer: "Komplexe Systeme und Nichtlineare Dynamik in Natur und Gesellschaft". In: *Springer* (1999).

[4] Gerhard Merziger u. a.: "Formeln + Hilfen Höhere Mathematik". In: *Binomi Verlag* ().

[5] Antonin Raffin u. a.: "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), S. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[6] John Schulman u. a.: "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[7] Mark Towers u. a.: "Gymnasium". In: *Zenodo* (März 2023). DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025.

7 Eigenständigkeitserklärung

Ich habe diese Arbeit selbstständig und ohne fremde Hilfe Dritter angefertigt. Bei der Übernahme fremder Gedanken habe ich dies als Zitat kenntlich gemacht.
Bielefeld, 06.04.2024
Nicolas Schneider