UNIVERSITÄT DER BUNDESWEHR MÜNCHEN
Fakultät für Elektrotechnik und Informationstechnik

Prädiktion von chaotischen Zeitreihen mit rekursiver genetischer Programmierung und Konstantenoptimierung

Witthaya Panyaworayan

Vorsitzender des Promotionsausschusses:	Prof. Dr.-Ing. K. Landes
1. Berichterstatter	Prof. Dr.-Ing. K. Tröndle
2. Berichterstatter	Prof. Dr.-Ing. U. Appel

Tag der Prüfung 16.07.2002

Mit der Promotion erlangter akademischer Grad:
Doktor-Ingenieur
(Dr.-Ing)

Neubiberg, den 1. September 2002

Vorwort

Die vorliegende Arbeit entstand im Rahmen einer Promotion am Institut für Informationstechnik der Universität der Bundeswehr München während meiner militärischen Ausbildung in der Bundeswehr.

Mein besonderer Dank gilt Herrn Professor Dr.-Ing. Karlheinz Tröndle für die Betreuung der Arbeit.

Ausdrücklicher Dank gebührt auch Herrn Dipl.-Ing.(FH) Karl Heinrich Besthorn für die hervorragende Unterstützung bei der Orthographie der Dissertation.

Danken möchte ich allen lieben Kolleginnen und Kollegen vom Institut für Informationstechnik für die gute und hilfsbereite Zusammenarbeit und das persönliches Verhältnis, das die tägliche Arbeit bereichert hat.

Neubiberg, im September 2002

Witthaya Panyaworayan

Zusammenfassung

In der vorliegenden Arbeit wird ein Prädiktionsverfahren der deterministischen chaotischen Zeitreihe unter Anwendung der genetischen Programmierung (GP) vorgestellt. Die Arbeit beschäftigt sich zu Beginn mit der Einführung in die GP. Dabei werden grundlegende Kenntnisse in der GP erläutert.

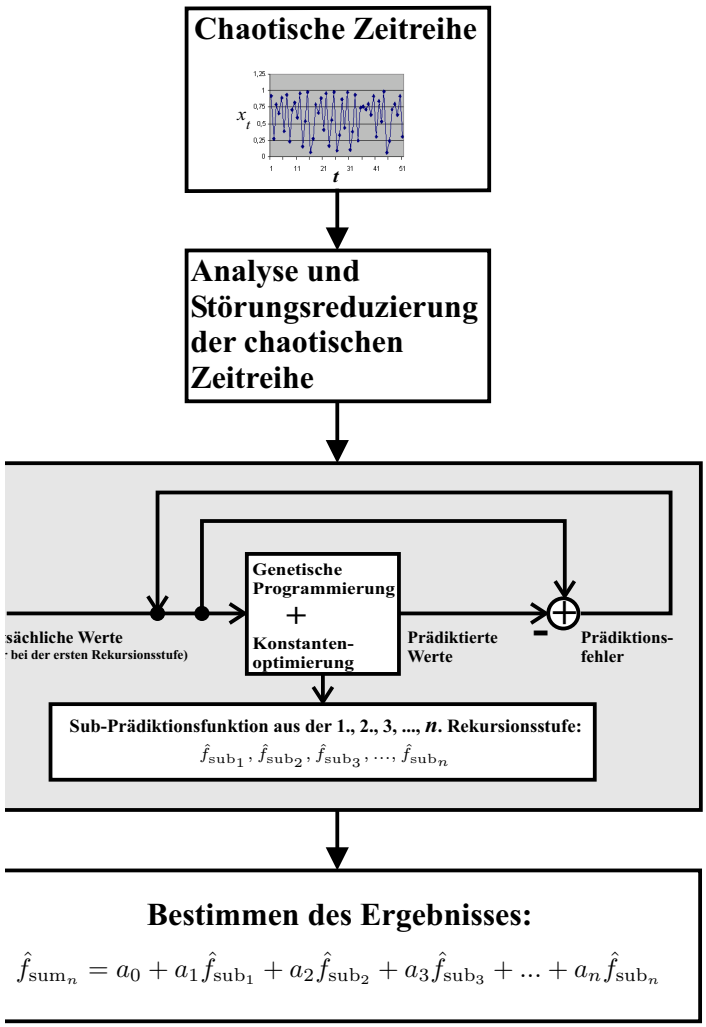
GP ist ein natur-inspiriertes Verfahren mit dem Ziel der automatischen Programmerzeugung. Computer sollen in die Lage versetzt werden, sich weitgehend selbst zu programmieren. Die GP liegt das natürliche Prinzip der Evolution zugrunde, wonach von einer Population diejenigen Individuen die besten Überlebens- und somit Reproduktionswahrscheinlichkeiten besitzen, die der jeweiligen Umgebung am besten angepasst sind. GP funktioniert am besten bei Problemen, die ein Kontinuum von Lösungen besitzen, wobei einige besser sind als andere. Dies ermöglicht der GP zu lernen, indem GP immer bessere Programme oder Lösungen entwickelt.

Das Problem, wie z.B. Kryptoanalyse, ist für die GP ungeeignet, da das Brechen eines Algorithmus der Kryptographie eigentlich nichts bietet, was mit Lernmöglichkeiten vergleichbar ist. Entweder hat man den Algorithmus entdeckt oder man hat ihn nicht entdeckt. Außerdem ist der Suchraum der kryptographischen Algorithmen extrem groß. Unsere Experimente zeigen, dass GP mit standardisierten Parametern nach einigen Trainingsschritten keine besseren Programme mehr entwickeln kann. Nur bei der GP, in der die Mutation intensiv angewendet wird, wird sich diese schwierige Lage verbessern. Mit Hilfe der Mutation werden völlige Neuheiten in die im vorherigen Trainingsschritt entwickelten Programme eingefügt. Dadurch können weit bessere Programme im nächsten Trainingsschritt entstehen.

In einem GP-Versuch entstehen mehrere Programme, deren Güte, sogenannte Fitness, durch eine bestimmte Funktion bestimmt werden muss. Dies führt dazu, dass der Zeitaufwand der GP sehr hoch ist. Durch die Parallelisierung der Fitnessberechnung auf verteilte Rechner lässt sich der Zeitaufwand der GP verringern. Mit diesem Verfahren soll der Zeitaufwand proportional zur Anzahl der verwendeten Rechner reduziert werden. Es ist jedoch nicht möglich, eine lineare Senkung des Zeitaufwands beim Erhöhen der Anzahl der verwendeten Rechner zu erreichen, weil man Zeit für die Kommunikation zwischen den Rechnern benötigt.

Im weiteren Teil der Arbeit wird die Einführung in die Chaostheorie und die bekannten Prädiktionsverfahren vorgestellt. Danach wird das Prädiktionsverfahren mit rekursiver GP und Konstantenoptimierung, der Hauptanteil dieser Arbeit erklärt.

Die folgende Abbildung verdeutlicht das Prädiktionsverfahren mit rekursiver GP und Konstantenoptimierung im Überblick.



Dieses Prädiktionsverfahren fordert keine höheren mathematischen Kenntnisse. Durch das Lernen mit den Trainingsdaten der deterministischen chaotischen Zeitreihe entwickelt GP die geeignete Struktur der Prädiktionsfunktion. Anschließend werden alle Konstanten und Koeffizienten in der durch GP entwickelten nichtlinearen Funktion optimiert, um ihre Effizienz zu erhöhen. Da die durch GP entwickelte Prädiktionsfunktion nichtlinear ist, benötigt man eine Konstantenoptimierung, die zum globalen Optimum hinführt. Eine solche Konstantenoptimierung ist zum Beispiel der genetische Algorithmus (GA).

Das Prädiktionsverfahren arbeitet rekursiv. Die Trainingsdaten in jeder Rekursionsstufe (außer der 1. Rekursionsstufe) sind die Prädiktionsfehler der Prädiktionsfunktion der vorherigen Rekursionsstufe. Dadurch wird in jeder Rekursionsstufe eine Sub-Prädiktionsfunktion, wie z.B. $\hat{f}_{\mathrm{sub}_1}, \hat{f}_{\mathrm{sub}_2}, \hat{f}_{\mathrm{sub}_3}, ..., \hat{f}_{\mathrm{sub}_n}$ entwickelt, die einen Teil der richtigen Prädiktionsfunktion darstellt. Am Ende des Prädiktionsprozesses bildet die Summe aller Sub-Prädiktionsfunktionen gemeinsam die Prädiktionsfunktion $\hat{f}_{\mathrm{sum}_n}$. Das Rekursionsverfahren hat den Vorteil, dass die durch GP gut entwickelten Prädiktionsfunktionen während des GP-Laufes nicht verloren gehen. Um die Effizienz der Prädiktionsfunktion $\hat{f}_{\mathrm{sum}_n}$ nochmals zu erhöhen, werden Koeffizienten

$a_0, a_1, a_2, ..., a_n$ eingefügt und durch GA optimiert.

Wegen der sensitiven Abhängigkeit vom Anfangszustand der deterministischen chaotischen Zeitreihe ist die Langzeitprädiktion einer deterministischen chaotischen Zeitreihe kaum möglich. Zur Verbesserung der Langzeitprädiktion sollen Störungen oder Rauschen in den deterministischen chaotischen Zeitreihen reduziert werden. Die Störungs- oder Rauschreduzierung durch die Filterung mit Tiefpässen ist eine Möglichkeit. Jedoch ist die Filterung mit Tiefpässen nicht in jeder Situationen zu verwenden, da die chaotischen Zeitreihen meistens einen hohen Frequenzanteil besitzen, der durch die Filterung verloren geht.

Ein wichtiges Problem bei der Prädiktion ist das Overfitting, d.h. die Auffindung einer Prädiktionsfunktion die zwar innerhalb der Trainingsdaten gut prädiktieren kann, jedoch für die Prädiktion der Daten außerhalb der Trainingsdaten ungeeignet ist. Das Overfitting-Problem erscheint deutlich, wenn die Trainingsdaten verrauscht sind. In dieser Arbeit werden folgende Maßnahmen zur Vermeidung von Overfitting durchgeführt.

- Die Trainingsdaten werden in zwei Mengen unterteilt, nämlich die Trainingsmenge und die Validierungsmenge. Das Training wird mit Hilfe der Trainingsmenge durchgeführt. Dabei überwacht man während des Trainings den Fehler bei der Validierungsmenge, der dem Validierungsfehler entspricht.
- Tritt beim Training Overfitting ein, d.h. der Validierungsfehler steigt nach dem Erreichen des minimalen Wertes an, wird das Training gestoppt. Das gefundene beste Ergebnis ist das Ergebnis mit dem niedrigsten Validierungsfehler.
- Der Verlauf des Validierungsfehlers hat aber mehrere lokale Minima bevor das endgültige Overfitting einsetzt. Aus dem Vorhandensein mehrerer lokaler Minima in der Validierungsfehlerkurve folgt, dass ein optimales Kriterium für das Stoppen sich nicht einfach auf das erste Minimum beschränken darf. Deshalb soll das Training dann abbrechen, wenn eine maximale Anzahl von Traingsschritten erreicht wird, oder der Trainingsfehler länger als eine bestimmte Anzahl von Trainingsschritten stagniert.

Dieses Verfahren zur Vermeidung des Overfittings wird sowohl in GP und als auch bei der Konstantenoptimierung eingesetzt.

Zum Schluss werden Prädiktionsexperimente mit vier verschiedenen deterministischen chaotischen Zeitreihen demonstriert, nämlich: die logistische Abbildung, monatliche durchschnittliche Sonnenfleckenrelativzahl, jährliche durchschnittliche Sonnenfleckenrelativzahl und der Wechselkurs zwischen Euro € und Baht ฿ (die thailändische Währung). Die Ergebnisse zeigen, dass unsere Prädiktionsverfahren bei der Prädiktion aller vier chaotischen Zeitreihen effizient sind. Die Experimente sind zwar unterschiedlich kompliziert. Je komplexer die chaotische Zeitreihe oder je stärker das übergelagerte Rauschen ist, desto schwieriger ist die Entwicklung einer Prädiktionsfunktion.

Inhaltsverzeichnis

1 Einleitung

1.1 Grundlagen

Wenn kleinste Veränderungen am Ausgangszustand riesige Auswirkungen auf das Ergebnis haben, sprechen Wissenschaftler von *Chaos*. Weltmärkte, Börsenkurse, das Wetter oder viele Eigenschaften der menschlichen Gesellschaft gehören dazu und ebenso die Turbulenzbildung in Flüssigkeitsströmungen oder die Aggregation von Bakterien in biologischen Systemen. Aber auch die Sonnenfleckenaktivität kann chaotisch verlaufen.

Chaos im ursprünglichen Wortsinne (griechisch *kaos*) ist das Gegenteil von Kosmos und bezeichnete bei den griechischen Philosophen den ungeordneten Stoff vor der Entstehung unserer übersichtlichen Welt.

Aus den philosophischen Meinungen über das Chaos ragen die Worte der beiden französischen Mathematiker Laplace und Poincaré heraus. Pierre Simon de Laplace (1749-1827) meinte, dass man die Zukunft aller Systeme dann voraussagen kann, wenn man alle Einzelheiten eines Systems kennt. Diese Theorie ging als der *Laplacesche Dämon* [Boy91] in die Geschichte ein. Dieser Dämon sollte eine Maschine sein, die alle Dinge der Welt weiß und daraus die Zukunft mechanisch vorhersagen könne.

Henri Poincaré (1854-1912) [Lev89] war anderer Meinung. Er meinte, dass kleine und unsichtbare Ursachen beachtliche Effekte bewirken können, die wir nicht mehr übersehen können. Poincaré lehrte, dass wir die Dinge dieser Welt niemals genau genug messen können um die Zukunft sicher vorhersagen zu können. Wir sprechen dann von zufälligen Entwicklungen, obwohl gar kein Zufall wirkt, sondern nur eine unbekannte Zahl nicht genau messbarer Kräfte.

Im Jahr 1963 stellte der Meteorologe E. N. Lorenz in [Lor69] ein System von lediglich drei gekoppelten Differentialgleichungen erster Ordnung vor, dessen Zeitentwicklung für gewisse Parameter einen völlig irregulären Verlauf aufweist. Er entdeckte damit ein Beispiel von *deterministischem Chaos*.

Heute versteht man im Allgemeinen, dass die zeitliche Entwicklung deterministischem Chaos folgt, also einerseits deterministischen Differenzen- bzw. Differentialgleichungen, andererseits zeichnen sie sich durch irreguläres, scheinbar zufälliges (chaotisches) Zeitverhalten aus.

Eine Beschreibung, Analyse und Prädiktion der deterministischen chaotischen Zeitreihe ist von großem wissenschaftlichen und wirtschaftlichen Interesse.

Bei der Prädiktion einer chaotischen Zeitreihe mit bekannten Prädiktionsverfahren, wie z.B. die globale und lokale Annäherung in [Cas89, Cas92, Tso92, Ger93, Kan97], wird zunächst in einem Phasenraum die chaotische Zeitreihe rekonstruktiert. Danach wird eine nichtlineare Funktion entwickelt, die den Verlauf des Phasenraums beschreiben kann. Eine andere Möglichkeit zur Prädiktion der chaotischen Zeitreihe sind künstliche neuronale Netze (KNN), wie z.B. in [Ger93, Kan97, Pham99] beschrieben. Bei der Prädiktion einer chaotischen Zeitreihe mit KNN fehlt die Rekonstruktion des Phasenraums. Ein geeignetes Netz der KNN wird zuerst ausgewählt. Dann trainiert das Netz mit Trainingsdaten (verfügbaren vorherigen Daten), damit KNN die zukünftigen Werte prädiktieren kann.

Die Prädiktion einer chaotischen Zeitreihe mit der genetischen Programmierung (GP) [Koza92] ist eine Alternative zu den vorher erwähnten Prädiktionsverfahren. GP ist ein weiter entwickelter Zweig der Evolutionären Algorithmen (EA) mit dem Ziel der automatischen Programmerzeugung. Durch das Training mit Trainingsdaten der chaotischen Zeitreihe kann GP eine Prädiktionsfunktion entwickeln.

Der Ursprung der EA kann bis zum Jahr 1950 zurückverfolgt werden (siehe [Bae97a]). Mangels verfügbarer Rechenleistung blieb die Entwicklung auf dem Gebiet der EA fast drei Jahrzehnte relativ unbekannt. In den 80er Jahren bewirkten die grundlegenden Arbeiten von J. H. Holland [Hol62], I. Rechenberg [Rec65], L. J. Fogel [Fog66] und H.-P. Schwefel [Schw68] eine langsame Veränderung bei der EA, seitdem nimmt die Anzahl der Veröffenlichung und Konferenz auf diesem Gebiet ständig zu.

EA bilden Klassen bio-inspirierter Verfahren zur Lösung unterschiedlicher Optimierungsaufgaben. EA werden vor allem dort eingesetzt, wo klassische Optimierverfahren aufgrund fehlender Voraussetzungen nicht eingesetzt werden können oder solche Methoden bekanntermaßen zu schlechten Resultaten führen. Zu diesen Situationen zählen z.B. hochdimensionale Suchräume mit vielen lokalen Optima, in denen andere Verfahren entweder eine Lösung von zu geringer Qualität liefern oder zur Berechnung einer Lösung ausreichender Qualität eine zu lange Zeit benötigt wird.

Das den evolutionären Algorithmen zugrunde liegende natürliche Prinzip ist das aus der Evolutionstheorie bekannte Prinzip, wonach von einer Population diejenigen Individuen die besten Überlebens- und somit Reproduktionswahrscheinlichkeiten besitzen, die der jeweiligen Umgebung am besten angepasst sind.

Im Jahr 1992 veröffentlichte J. R. Koza die GP in seiem Buch [Koza92]. GP wurde ursprünglich zur automatischen Erzeugung von Programmen in der LISP-Programmiersprache entwickelt. Computer werden dabei in die Lage versetzt, sich selbst zu programmieren.

GP bietet eine Vielzahl der Einstellung von Strategieparametern, die vor dem Start der GP einzustellen sind. Die unterschiedlichen Parametereinstellungen sind vorteilhaft, da man GP mit geeigneten Mechanismen ausstatten kann, die GP an die jeweilige Problemssituation anpassen.

Während eines GP-Laufes werden mehrere Individuen (Programme) erzeugt, deren Güte oder Qualität durch eine bestimmte Fitnessfunktion bestimmt werden muss. Dies führt dazu, dass der Zeitaufwand der GP sehr hoch ist.

In den letzten zehn Jahren wurde GP in verschiedenen Gebieten, wie z.B. in [Kin94, Ang96, Spec99], mit Erfolg eingesetzt. Bereits in [Koza92] wird die Anwendung der GP für die Prädiktion der deterministischen chaotischen Zeitreihe vorgeschlagen. Für die Prädiktionsaufgabe ist GP sehr flexibel und erfordert außerdem keine höheren mathematischen Kenntnisse.

Ein häufiges Problem, das bei allen Prädiktionsverfahren auftritt, ist das *Overfitting*, d.h. die gefundene Prädiktionsfunktion ist innerhalb der Trainingsdaten gut, jedoch ist diese Prädiktionsfunktion für Daten außerhalb der Trainingsdaten ungeeignet. Das Overfitting-Problem erscheint deutlich, wenn die Trainingsdaten verrauscht sind. Dadurch versucht das Prädiktionsverfahren, während des Trainings sich dem Rauschanteil anzupassen.

1.2 Problemstellung

Daraus ergeben sich die folgenden Fragen, die in dieser Arbeit zu beantworten sind:

- Welcher Aufgabenbereich ist für GP mit der standardisierten Parametereinstellung ungeeignet?
- Durch welche Möglichkeiten kann der Zeitaufwand des GP-Laufes reduziert werden.
- Welche Verfahren zur Prädiktion einer deterministischen chaotischen Zeitreihe unter Anwendung der GP sind geeignet?
- Welche Gegenmaßnahme kann das Overfitting-Problem in dem vorgeschlagenen Prädiktionsverfahren vermeiden?

In den nachfolgenden Kapiteln wird versucht, Antworten auf diese Fragen zu finden.

1.3 Inhaltsübersicht

Nach einer kurzen Einführung in die EA in Kapitel 2 folgen in Kapitel 3 Grundlagen, nämlich das Grundkonzept und der Ablauf von GP. Weiterhin wird die Einführung in die genetische Programmierung mit kontextfreier Grammatik erläutert.

Im Kapitel 4 stellen wir als Beispiel eine Aufgabe vor, für die die GP nicht geeignet ist, da GP die Aufgabe nicht lernen kann. Eine solche Aufgabe ist z.B. die Kryptoanalyse.

Kapitel 5 beschäftigt sich mit der globalen Parallelisierung, die eine Möglichkeit zur Reduzierung des Rechenaufwands von GP ist. Wir stellen die Grundlagen der globalen Parallelisierung, ein Programmpaket für ein Experiment und die Ergebnisse aus dem Experiment vor.

Kapitel 6 gibt eine Einführung in die Chaostheorie. Wir beginnen mit den Begriffen dynamisches System und Attraktor. Danach wird das deterministische Chaos definiert. Dann werden zwei bekannte Methoden zur Charakterisierung chaotischen Verhaltens erläutert. Dies sind die Lyapunov-Exponenten und die Anzahl der Dimensionen. Zum Schluss werden einige Beispiele des dynamischen Chaos demonstriert.

Im Kapitel 7 stellen wir die Einführung in die Prädiktion der deterministischen chaotischen Zeitreihen vor. Zunächst werden Begriffe wie Modellieren, Identifikation und Prädiktion definiert. Danach werden die Arten der Prädiktion erläutert. Zum Schluss stellen wir die verschiedenen Arten der Prädiktionsverfahren vor.

Kapitel 8 ist das Hauptkapitel dieser Arbeit. Es beschäftigt sich mit dem Aufbau und der Funktionsweise des Prädiktionsverfahrens einer deterministischen chaotischen Zeitreihe unter Anwendung der GP. Für die Entwicklung einer nichtlinearen Funktion zur Prädiktion der deterministischen chaotischen Zeitreihe ist GP besonders geeignet. GP fehlen bei der Suche jedoch noch die angepassten Konstanten in dieser Prädiktionsfunktion. Durch die Kombination von GP mit anderen Konstantenoptimierungsverfahren soll das Prädiktionsverfahren mit GP verbessert werden. Bei KNN gibt es bereits verschiedene Methoden, mit dem Overfitting-Problem umzugehen. Eine einfache und bekannte Methode ist das frühe Stoppen. Dies kann sowohl für GP als auch für die Konstantenoptimierung sehr nützlich sein. Außerdem stellen wir die Funktion zur Berechnung der Prädiktionsfehler und den Verlauf des Prädiktionsfehlers vor.

In Kapitel 9 demonstrieren wir die Experimente der Prädiktion von chaotischen Zeitreihen. Im ersten Teil dieses Kapitels werden die Prädiktionen von chaotischen Zeitreihen mit dem einfachen Verfahren und dem Rekursionsverfahren verglichen. Im zweiten Teil werden die Experimente von vier weiteren chaotischen Zeitreihen ohne und mit Reduzierung des Rauschens gezeigt, nämlich: die logistische Abbildung, monatliche durchschnittliche Sonnenfleckenrelativzahl, jährliche durchschnittliche Sonnenfleckenrelativzahl und der Wechselkurs zwischen Euro € und Baht ฿

(die thailändische Währung). Anschließend werden die Ergebnisse der Experimente diskutiert.

2 Evolutionäre Algorithmen

Bei der Bewältigung komplexer Optimierungsprobleme haben in jüngster Zeit Methoden beträchtlich an Bedeutung gewonnen, die man als *naturanalog* bezeichnen kann, da sie sich an Vorbildern der Natur orientieren. Zu den bekanntesten neueren Entwicklungen auf diesem Gebiet zählen die *Evolutionären Algorithmen (EA)*. Praktisch synonym verwendet man vielfach auch den englischen Begriff *Evolutionary Computation*.

Dieses Kapitel stellt eine Einführung in die EA dar. Wir beginnen im Abschnitt 2.1 mit dem biologischen Vorbild der Natur. Im Abschnitt 2.2 und 2.3 werden allgemeine Konzepte für EA und die vier bekanntesten Hauptströmungen der EA erläutert. Da EA gut auf Parallelrechnern implementiert werden können, wird im Abschnitt 2.4 ein Konzept für die mögliche Parallelisierung vorgestellt. Zum Schluss werden Stärken und Schwächen der EA zusammengefasst.

Für weiterreichende Darstellungen der Einführung in die evolutionären Algorithmen wird auf [Bae97a, Bae97b, Fog00], sowie das deutschsprachige Buch von V. Nissen [Nis97] verwiesen.

2.1 Das biologische Vorbild

Die biologische Genetik, insbesondere aber die Evolutionstheorie, diente als Vorbild für die Entwicklung von evolutionären Algorithmen. Die Informationen über Struktur und Fähigkeiten eines biologischen Individuums sind bekanntlich in den *Chromosomen* codiert, diese wiederum sind Bestandteil jeder Zellle. Ein Chromosom besteht chemisch vor allem aus *Desoxyribonukleinsäure (DNA)* und diese ist, in der Sprache der Informatik, ein String über einem Alphabet mit vier Zeichen. Diese vier Zeichen sind Moleküle, die in einer Kette zu einem Chromosom aufgereiht sind: das *Adenin (A)*, *Thymin (T)*, *Cytosin (C)* und *Guanin (G)*. Teilstrings der Chromosomen sind Baupläne für zu bildende Proteine und Enzyme. So ist der gesamte Bauplan eines Individuums in jeder einzeln Zelle abgespeichert. Der vollständige Chromosomensatz des Menschen (Genom) enthält etwa 3.000.000.000 Zeichen.

Die in Erbversuchen erfassbaren Erbeinheiten (Blütenfarbe usw.) bezeichnet man als *Gene*. Die Vererbung der durch Gene charakterisierten Eigenschaften erforschte als erster J. G. Mendel, der seine Ergebnisse 1865 veröffentlichte. Die nach ihm benannten Vererbungsgesetze hatte er durch Kreuzungsversuche an Erbsen erhalten.

Durch Kreuzungsversuche mit der Frucht- oder Taufliege (Drosophilamelanogaster) fand T. H. Morgan Vererbungsgesetze, die nur dadurch erklärbar sind, dass bei der Fortpflanzung, d.h. bei der Reifeteilung der Zelle, zwei homologe Chromosomen sich umschlingen. Dabei erfolgt ein Bruch und die Bruchstellen verknüpfen sich über *Kreuzung*.

Des Weiteren erfolgen, wenn auch selten, Veränderungen von Chromosomenteilen durch Mutation. Durch *Mutationen* entstanden z.B. die verschiedenen Rassen unserer Haustiere und auch die Kulturpflanzen, die sich aus Wildformen entwickelten. Dabei können Mutationen in jeder Zelle des Körpers auftreten, jedoch werden nur Mutationsveränderungen in den Keimzellen genetisch weitergegeben.

Im Jahr 1859 veröffentlichte C. Darwin in *On the Origin of Species by Means of Natural Selection* seine Selektionstheorie. Er beobachtete und konstatierte die folgenden Sachverhalte.

- Die Lebewesen erzeugen mehr Nachkommen, als zum Erhalt der Rasse notwendig wären. Nicht alle können überleben.
- Die Nachkommen eines Elternpaares variieren in ihren Eigenschaften und Fähigkeiten.
- Jene Lebewesen, deren Eigenschaften zum Futtererwerb, zur Ausnutzung des Lebensraumes und zur Paarung am günstigsten entwickelt sind, überleben und verdrängen die anderen.

Dies führt zu einer Selektion der am Besten an ihre Umwelt Angepassten. Diese natürliche Auslese führt über Generationen zu einer allmählichen Umbildung der Arten.

Selektion und damit Evolution funktioniert, wenn die folgenden Voraussetzungen gegeben sind.

- Es gibt eine Population von Individuen.
- Die Individuen einer Population variieren in ihren Eigenschaften.
- Die Fähigkeit zum Überleben (Fitness) hängt von den Eigenschaften des Individuums ab.
- Die Individuen einer Population besitzen die Fähigkeit zur Reproduktion.

Diese Theorie reduziert die Wirklichkeit auf ein Modell, und es ist sicher unzulässig, Realität und Modell gleich zu setzen. Trotzdem erwies sich die Theorie als leistungsfähig bei der Erklärung der Entstehung der Arten.

2.2 Evolutionäre Algorithmen

2.2.1 Grundkonzept

EA modellieren die natürliche Vererbung und Evolution, also die Entwicklung komplexer Organismen aus zeitlich früheren, einfacheren Lebensformen. EA arbeiten mit einer Population von Individuen. Jedem Individuum der Population wird eine bestimmte Fitness zugeordnet. In diesem Fitnessmaß ist die Aufgabe kodiert, die die EA lösen soll. Je besser ein Individuum die Lösung beschreibt, desto besser ist dessen Fitness.

Bei den EA werden stochastische Verfahrenselemente bewusst eingesetzt. Daraus entsteht jedoch keine reine Zufallssuche, sondern ein intelligenter Suchprozess.

Allgemein lässt sich ein EA-Lauf folgendermaßen beschreiben:

- Zu Beginn wird eine Population mit zufällig erzeugten Individuen initialisiert.
- Für alle Individuen in der Population wird eine bestimmte Fitness berechnet.
- Ist ein festgelegtes Abbruchkriterium erfüllt, bildet das Individuum mit dem besten Fitnesswert in der Population das Ergebnis des Algorithmus, d.h.eine approximative Lösung der Aufgabe. Ein Abbruch erfolgt in der Regel nach Erreichen einer maximalen Generationenanzahl oder eines bestimmten besten Fitnesswertes (Erfolgskriterium).
- Danach werden Individuen bezüglich ihrer Fitness aus der Population ausgewählt. Auf die ausgewählten Individuen, oder Eltern genannt, werden die folgenden genetischen Operationen angewandt:

  1. *Reproduktion*: identisches Kopieren des ausgewählten Individuums
  2. *Mutation*: Veränderung einzelner zufällig gewählter Stringpositionen
  3. *Kreuzung (Crossover)*: Austausch von Teilstrukturen zwischen zwei Individuen

  Durch Anwendung dieser an der Natur angelehnten Operationen werden Nachkommen produziert und in die nächste Generation überführt. Dabei bleibt die Populationsgröße konstant.

- Bei einigen EA-Formen findet die Selektion erst am Ende der Generationsschleife statt. Jedes Individuum hat dann die Möglichkeit zur Erzeugung von mehreren Nachkommen. Danach werden die Nachkommen hinsichtlich ihrer Fitness bewertet. Die Eltern und Nachkommen werden anschließend bezüglich ihrer Fitness ausgewählt und als Individuen in die nächste Generation überführt.

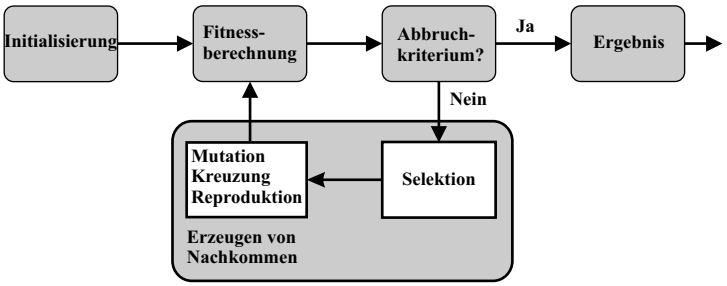


Abbildung 2.1: Ablauf der evolutionären Algorithmen

Der geschilderte Ablauf iteriert, bis ein Abbruchkriterium greift. Abbildung 2.1 verdeutlicht einen EA-Lauf nochmals im Überblick.

Mit wachsender Anzahl von Generationen nimmt die Fitness der Individuen in der Population meist zu. Dies begründet sich einmal darin, dass Individuen mit guter Fitness häufiger selektiert und reproduziert werden als andere, und in der Tatsache, dass die Kreuzung zweier Eltern-Individuen mit guter Fitness mit höherer Wahrscheinlichkeit Nachkommen entstehen lässt, die gleich gute oder bessere Fitnesswerte aufweisen.

Das allgemeine Schema verdeutlicht den probabilistischen Charakter Evolutionärer Algorithmen, der mehrere unabhängige Versuche nötig macht, um ein zuverlässiges Ergebnis für ein Problem zu erzielen.

2.2.2 Evolutionäre Algorithmen als Optimierungsverfahren

In praktischen Anwendungen werden EA überwiegend als globale Optimierungsverfahren eingesetzt. Globale Optimierungsprobleme sind dadurch gekennzeichnet, dass aus einer im Allgemeinen sehr großen Anzahl von möglichen Lösungen diejenige bestimmt werden soll, die sowohl alle Nebenbedingungen der Problemstellung einhält, als auch die in Gestalt der Zielfunktion gegebenen Zielkriterien am besten erfüllt (*global optimale Lösung*). Dabei kann es auch mehrere verschiedene global optimale Lösungsalternativen mit gleichem Zielfunktionswert geben. Eine Lösung stellt ein *lokales Optimum* dar, wenn ihr Zielfunktionswert besser ist als die Zielfunktion aller Lösungen, die in einer definierten Nachbarschaft zu ihr liegen. Die Schwierigkeit der Suche nach Optimierungsverfahren besteht letztlich darin, dass es keine Kriterien gibt, um eine lokal und global optimale Lösung ohne weiteres Vorwissen von einer nur lokal optimalen Lösung zu unterscheiden.

Man kann im Allgemeinen nicht garantieren, dass EA bei praktischen Anwendungen in begrenzter Zeit ein globales Optimum finden. Zahlreiche Beispiele zeigen aber, dass EA häufig in beschränkter Zeit sehr gute oder nahezu optimale Lösungen finden können. EA gehören insofern ihrem Charakter nach zu den *Heuristiken*.

Unter einer Heuristik versteht man eine nichtwillkürlich und häufig iterative Methode, die darauf abzielt, für eine gegebene Problemstellung in begrenzter Zeit eine oder mehrere möglichst gute Lösungen zu finden, ohne dass garantiert werden kann, dass eine globale optimal Lösung gefunden wurde.

Um die Qualität von Heuristiken zu beurteilen, lassen sich die beiden Kriterien Allgemeinheit und Leistungsfähigkeit heranziehen.

- Der *Grad an Allgemeinheit* einer Heuristik bringt zum Ausdruck, auf wieviele verschiedene Problemstellungen sie anwendbar ist.
- Die *Leistungsfähigkeit* einer Heuristik lässt sich an mindestens drei Dimensionen messen:
  - *Lösungswahrscheinlichkeit*: Wahrscheinlichkeit, mit der die Heuristik eine Lösung von bestimmter Mindestqualität findet,
  - *Lösungsqualität*: Abweichung vom global optimalen Zielfunktionswert,
  - *Ressourcenbedarf*: Bedarf an Ressourcen (Mitteln), um eine bestimmte Lösungsqualität und Lösungswahrscheinlichkeit zu erzielen.

Weniger leistungsfähige aber dafür breit anwendbare Heuristiken gehören zur Klasse der *schwachen Methoden*. Leistungsstarke, aber dafür nur in einem schmalen Bereich anwendbare Heuristiken sind dagegen *starke Methoden*. Eine starke Methode enthält viel anwendungsspezifisches Vorwissen, während schwache Methoden mit wenig oder keinem Anwendungswissen auskommen. EA lassen sich an jeder Stelle auf einem Kontinuum zwischen schwachen und starken Methoden plazieren. So zählen die Basisformen der einzelnen EA-Hauptströmungen zu den schwachen Methoden. Sie sind ohne große Veränderungen auf viele verschiedene Problemstellungen anwendbar. Es existieren jedoch eine ganze Reihe von Möglichkeiten, anwendungsbezogenes Vorwissen in die EA zu integrieren, um ihre Leistungsfähigkeit zu verbessern.

2.2.3 Anwendungsgebiete von evolutionären Algorithmen

Die mögliche Anwendungsgebiete von EA sind [Bae97b]:

- *Planung*: z.B. Bei dem Problem des Handelsreisenden (*Travelling-Salesman-Problem*) sind die EA fähig, nach der kürzesten Verbindung zwischen mehreren Städten zu suchen.
- *Entwurf*: z.B. Entwurf von elektrischen oder digitalen Systemen
- *Modellieren und Identifikation*: z.B. Identifikation eines Systems und seine Modellierung sowie Charakterisierung seiner Ein- und Ausgangsignale oder Prädiktion von dynamischen Zeitreihen

- *Regelung*: EA sind anpassungsfähig oder lernfähig, daher können sie im Bereich von Regelung oder Kontrolle verwendet werden.
- *Klassifikation*: z.B. Mustererkennung und Bildverarbeitung

2.3 Vier Hauptformen der evolutionären Algorithmen

Man kann heute vier verschiedene Hauptformen der EA unterscheiden [Nis97]. Diese sind:

1. Genetische Algorithmen (GA),
2. Genetische Programmierung (GP),
3. Evolutionsstrategien (ES),
4. Evolutionäre Programmierung (EP).

Diese EA-Hauptformen unterscheiden sich nicht zuletzt in der Ausgestaltung der Selektion sowie bei den eingesetzten Variationsoperatoren. Als Gedankenstütze seien einige wesentliche Unterscheidungsmerkmale der vier EA-Hauptströmungen erwähnt:

Genetische Algorithmen

GA [Scho94, Bae97b, Nis97] verwenden häufig eine binäre Lösungsdarstellung. Die Selektion erfolgt stochastisch, so dass auch schlechten Individuen eine gewisse Chance zur Reproduktion gegeben wird. Als Hauptsuchoperator dient die Kreuzung, während die Mutation nur mit geringer Wahrscheinlichkeit auftritt.

Genetische Programmierung

GP [Koza92, Bae97b, Nis97, Ban00] ist im Grunde nur eine GA-Variante. Besonders hervorzuheben ist die Lösung in Form von Computerprogrammen. GP wird im 3. Kapitel näher betrachtet.

Evolutionsstrategien

ES [Scho94, Nis97] verwenden im allgemeinen einen Vektor reeller Zahlen zur Lösungsdarstellung. Die Selektion findet am Ende der Generationsschleife statt und erfolgt deterministisch, so dass nur die besten Individuen überleben. Die Mutation auf der Basis normalverteilter Zufallsgrößen ist als Suchoperator besonders wichtig, doch spielt auch die Kreuzung eine große Rolle.

Evolutionäre Programmierung

EP [Bae97b, Nis97] weist starke Ähnlichkeiten zu ES auf und verwendet häufig ebenfalls einen Vektor reeller Zahlen zur Lösungsdarstellung. Es wird jedoch mit einer stochastischen Form der Selektion gearbeitet und die Mutation bildet den einzigen Suchoperator.

2.4 Parallelisierbarkeit von evolutionären Algorithmen

EA ist gut auf Parallelrechnern zu implementieren. Die in EA nachgeahmten Evolutionsphänomene sind hochgradig parallel verlaufende Prozesse, daher können EA von paralleler Hardware in verschiedener Weise profitieren.

- EA sind aufgrund ihres inhärent parallelen Charakters relativ leicht und effizient zu parallelisieren.
- Parallele EA ermöglichen eine realistischere Abbildung des Evolutionsgeschehens als serielle EA.
- Auf Parallelrechnern kann mit zum Teil wesentlich größeren Gesamtpopulationen gearbeitet werden als auf seriellen Maschinen. Dadurch, und aufgrund der zu erreichenden Beschleunigung, können komplexere Anwendungen bearbeitet werden.

Folgende grundsätzlichen Möglichkeiten der Parallelisierung von EA werden üblicherweise unterschieden:

Globale Parallelisierung: Bei der globalen Parallelisierung werden gegenüber seriellen Implementierungen die Fitnessberechnung, sowie teilweise auch die Ausführung evolutionärer Operatoren parallelisiert, ohne inhaltliche Änderungen am Verfahrensablauf vorzunehmen. Es existiert nur eine Population, jedes Individuum der Population kann mit jedem anderen gekreuzt werden. Die Fitnessberechnung eines Individuums ist im Normalfall unabhängig von anderen Mitgliedern der Population und daher leicht zu parallelisieren (Abbildung 2.2).

Migrationsmodell: Das Migrationsmodell unterteilt die Gesamtpopulation in Teilpopulationen, sogenannte *Deme* oder *Teilpopulationen*, die einzelnen Prozessoren zugeordnet werden (Abbildung 2.3).

Zwischen den Teilpopulationen sind Migrationspfade definiert. In jeder dieser Teilpopulation werden die herkömmlichen EA ausgeführt. Die Teilpopulationen entwickeln sich weitgehend unabhängig voneinander und tauschen nur gelegentlich Individuen aus.

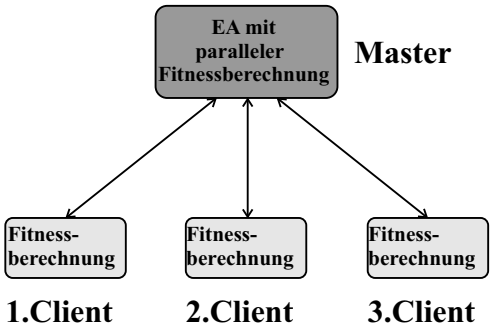


Abbildung 2.2: Einfache globale Parallelisierung mit paralleler Fitnessberechnung

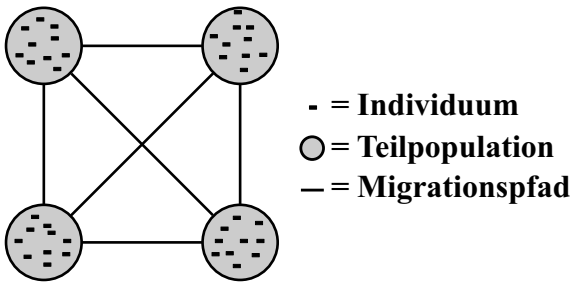


Abbildung 2.3: Einfaches Migrationsmodell mit 4 Teilpopulationen

Diffusionsmodell: Beim Diffusionsmodell ist der Grundgedanke folgender: Der im Allgemeinen sehr großen Gesamtpopulation wird eine bestimmte Struktur zugrunde gelegt, z.B. ein zweidimensionales Gitter (Abbildung 2.4).

Jedes Individuum ist dabei einem Gitterpunkt zugeordnet. Außerdem wird die Population in viele kleine lokale Nachbarschaftsstrukturen zergliedert. Die Nachbarschaft eines Individuums bilden alle jene Individuen, die innerhalb einer bestimmten Entfernung liegen. Selektion und Kreuzung sind auf Individuen in der definierten Nachbarschaft beschränkt. Die Nachbarschaften verschiedener Individuen überlappen sich, so dass Elemente guter Lösungen sich langsam über die ganze Population ausbreiten können. Dieses Diffusionsmodell reduziert die Heterogenität von Individuen in der Gesamtpopulation viel langsamer als es bei EA, in der nur eine Population existiert, der Fall wäre.

Empirische Untersuchungen, in denen die grundsätzlichen Parallelisierungskonzepte für EA verglichen werden, sind in der Literatur selten. Eine effiziente Parallelisierungsstrategie muss letztlich auf die Architektur des benutzten Rechnersystems abgestimmt werden. Insgesamt scheint das Migrationsmodell die meisten Anhänger zu haben.

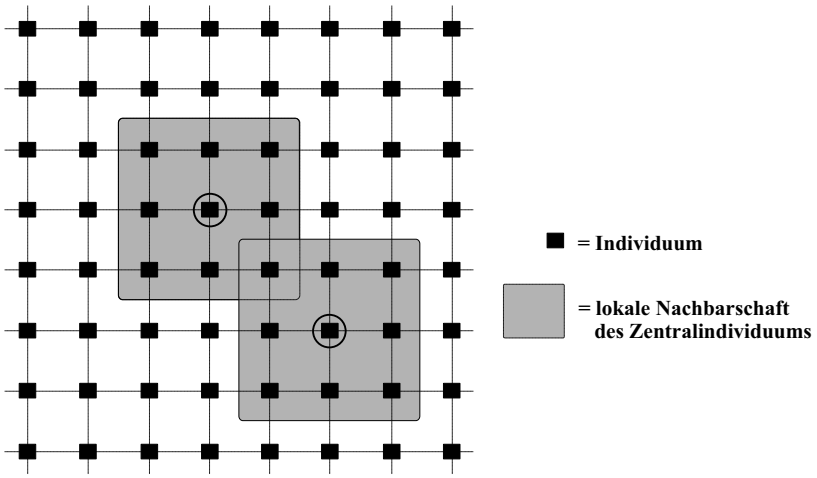


Abbildung 2.4: Einfaches Diffusionsmodell (zweidimensionales Gitter)

2.5 Stärken und Schwächen von evolutionären Algorithmen

2.5.1 Stärken von evolutionären Algorithmen

Breite Anwendbarkeit der EA-Basisformen

EA stellen breit anwendbare schwache Methoden dar, weil diese als Basisformen ohne anwendungsspezifisches Vorwissen arbeiten und mit sehr geringen Anforderungen an die Struktur des gegebenen Problems auskommen.

Flexible Verfahrensgestaltung oder Anpassung an Problemstellungen

EA bieten sehr flexible Gestaltungsmöglichkeiten um die Leistungsfähigkeit im konkreten Einzellfall zu verbessern, z.B. durch:

- problemspezifische Lösungscodierung, angepasste Suchoperatoren, intelligente Decodierung und Lösungsoptimierung,
- Gestaltung der Fitnessfunktion,
- gezielte Initialisierung der Ausgangspopulation,
- Maßnahmen zur Berücksichtigung von Nebenbedingungen,
- Hybridisierung mit nichtevolutionären Lösungsverfahren.

Eignung für komplexe Suchräume

Durch den Populationsansatz und die stochastischen Komponenten verringert sich bei EA die von vielen anderen Optimierungsverfahren bekannte Abhängigkeit vom Startpunkt der Optimierung. Weil EA tolerant sind gegenüber vorübergehenden Verschlechterungen des Zielfunktionswertes, können außerdem lokale Suboptima oft wieder verlassen werden, solange die Population noch hinreichend heterogen ist.

Keine restriktiven Anforderungen an die Zielfunktion

Viele klassische Optimierungsverfahren sind auf eine gegebene Problemstellung nur anwendbar, soweit die Zielfunktion stetig und ein- bzw. zweimal differenzierbar ist. EA kommen ohne diese restriktiven Anforderungen aus, wodurch sich ihr potentieller Einsatzbereich verbreitert.

Gut verständliche Basisprinzipien

Es ist ohne weiteres möglich, einem Laien die grundsätzlichen Prinzipien von EA in einfachen Worten verständlich und plausibel zu machen.

Anwendbarkeit bei unbekannter Struktur des Anwendungsproblems

Die EA-Basisformen können bei Problemstellungen eingesetzt werden, bei denen Vorwissen bezüglich der Anwendung nicht gegeben ist. Sie werden zwar in solchen Fällen keine überragenden Ergebnisse liefern, aber die bisherige Situation sollte sich mit ihnen verbessern lassen.

Gute Kombinationsmöglichkeiten

EA lassen sich sehr gut mit lokalen Verbesserungsverfahren kombinieren.

Gute Parallelisierbarkeit

Die Parallelisierbarkeit wurde im Abschnitt 2.3 erläutert.

2.5.2 Schwächen von evolutionären Algorithmen

Fehlende Optimalitätsgarantie bei beschränkter Rechenzeit

Man kann nicht garantieren, dass EA in begrenzter Zeit für ein beliebiges Optimierungsproblem ein globales Optimum finden.

Relativ hoher Rechenaufwand

Als populationsbasierte Methoden sind EA oft rechenintensiv. Viele der in einem Optimierungslauf erzeugten Lösungen führen nicht unmittelbar zu einer Verbesserung, obwohl sie unter anderem dazu beitragen könnten, das Verfahren robust zu machen.

Ineffektivität in der Schlussphase

Die Suchoperatoren von EA sind nicht speziell auf schnelle lokale Optimierung ausgelegt, weshalb EA in der Schlussphase eines Optimierungslaufes oft nicht effizient sind.

Schwierige Anpassung an die Problemstellung

EA bieten sehr flexible Gestaltungmöglichkeiten, wie z.B. die Lösungsrepräsentation, Suchoperatoren, Fitnessfunktion und viele Strategieparameter, um die Leistungsfähigkeit im konkreten Einzellfall zu verbessern. Diese vielen Freiheitsgrade bei der Verfahrensgestaltung können für unerfahrene Entwickler zum Problem werden.

2.6 Zusammenfassung

EA orientieren sich am Vorbild der natürlichen Evolution. EA arbeiten mit einer Population von Individuen, welche Suchpunkte im Raum der Entscheidungsvariablen darstellen, entweder direkt oder indirekt mit Codierungsstufen. Der Übergang von einer Generation zur nächsten erfolgt durch den Austausch von genetischem Material zwischen Individuen, z.B. durch Kreuzung oder Mutation. Die Selektion erfolgt gemäß dem Gütekriterium (Fitness) durch Auswertung der als potentielle Lösungen des Optimierproblems aufzufassenden Individuen. Bei jedem Selektionsschritt werden dabei die besseren Individuen beibehalten, die dann zu Eltern der Folgegeneration werden, während die schlechteren Individuen keine Nachkommen produzieren dürfen, oder die mittlere Nachkommenzahl ist proportional zur elterlichen Fitness.

Mutation und Kreuzung sind im Algorithmus meist Prozesse mit nichtdeterministischen Komponenten, während Selektion sowohl streng deterministisch als auch probabilistisch modelliert werden kann. Im Laufe des Evolutionsprozesses steigt so die durchschnittliche Qualität der Individuen an und ermöglicht das Auffinden einer guten, oft sogar der global optimalen Lösung des Problems.

Die bekanntesten Vertreter von EA sind die in den USA entwickelten GA, GP und EP sowie die in Deutschland entwickelten ES.

Der Vorteil der EA sind ihre breite Anwendbarkeit der Basisverfahren, ohne restriktive Anforderungen an die Zielfunktion (es ist insbesondere keine Stetigkeit oder Differenzierbarkeit erforderlich). EA sind gut auf Parallelrechnern zu implementieren. EA arbeitet aber mit relativ hohem Rechenaufwand. Außerdem kann man nicht garantieren, dass EA in begrenzter Zeit für ein beliebiges Optimierungsproblem ein globales Optimum finden.

3 Genetische Programmierung

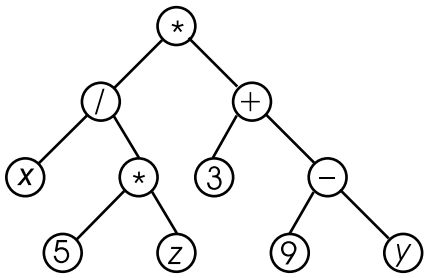
Dieses Kapitel beschäftigt sich mit den Grundlagen der genetischen Programmierung (GP). GP ist ein von Koza [Koza92] entwickelter Zweig der evolutionären Algorithmen mit dem Ziel der automatischen Programmerzeugung. In diesem Kapitel wird außerdem die genetische Programmierung mit kontextfreier Grammatik (CTF-GP), die eine Erweiterung von GP ist, dargestellt. Für weiterreichende Darstellungen der GP, als sie hier ausgeführt werden, wird auf die Arbeiten von W. Banzhaf [Ban98, Ban00] verwiesen.

3.1 Grundkonzept

Die Aufgabe, anhand von Trainingsdaten automatisch ein Computerprogramm zu generieren das eine gegebene Aufgabenstellung löst, wird als Programm-Induktion bezeichnet. GP ist als eine eigenständige Variante der evolutionären Algorithmen für die Programm-Induktion anzusehen. Die Individuen oder Strukturen, die in der GP durch evolutionäre Operatoren modifiziert werden, sind Computerprogramme unterschiedlicher Größe und Komplexität. Sie setzen sich aus Funktionen, Variablen und Konstanten zusammen.

3.1.1 Funktionenmenge und Terminalmenge

Die Ausgangspopulation besteht bei der GP aus mehreren stochastisch generierten Programmen. Die GP-Individuen (Programme) sind im Regelfall baumartig strukturiert (Siehe Abbildung 3.1). Sie werden gebildet aus den Elementen einer Menge problemangemessener elementarer Funktionen (*Funktionenmenge, function set,* $\mathcal{F}$) sowie den Elementen einer Menge problemangemessener Variablen und Konstanten (*Terminalmenge, terminal set,* $\mathcal{T}$). Beide Mengen bilden die Grundlage, auf deren Basis GP versucht, Programme zu erzeugen, die das gegebene Anwendungsproblem bestmöglich lösen. Gleichzeitig determiniert die Wahl der Funktionenmenge und der Terminalmenge den Raum aller potentiell generierbaren Programme. Damit haben diese Vorgaben großen Einfluss auf die Komplexität des Suchraumes und beeinflussen auch die Qualität des GP-Ergebnisses.



$(\frac{x}{5 \cdot z}) \cdot (3 + (9 - y))$

Abbildung 3.1: Programmbaum und korrespondierender Ausdruck

Elemente der Funktionenmenge $\mathcal{F}$ können insbesondere sein:

- arithmetische Operationen (wie $+, -, *$),
- mathematische Funktionen ($\sin, \cos$, usw.),
- boolsche Operationen (**AND**, **NOT**, usw.),
- Verzweigungen (**IF-THEN-ELSE**),
- Iterationen (**FOR**, **DO-UNTIL**, usw.),
- anwendungsspezifische Funktionen (wie z.B. *protected division function*).

Elemente der Terminalmenge sind:

- Konstanten und
- Variablen.

In praktischen Anwendungen sollte man grundsätzlich versuchen, möglichst nützliche Funktionen in die Funktionenmenge aufzunehmen, die voraussichtlich das rasche Auffinden guter Lösungen erleichtern. Hier besteht also eine Möglichkeit zur Berücksichtigung von anwendungsspezifischem Vorwissen. **Durch geeignete Auswahl der Funktionenmenge kann Vorwissen berücksichtigt werden.** Die Größe des Suchraumes hängt direkt von den Elementen der Funktionenmenge und der Terminalmenge ab. Man sollte diese Mengen von GP-Grundbausteinen also nicht unnötig aufblähen. Andererseits darf man sie aber nicht so stark einschränken, dass gute Lösungen nicht mehr gefunden werden können.

An die benutzerdefinierten Grundbausteine, also Funktionenmenge und Terminalmenge, werden die folgenden zwei Anforderungen gestellt:

- ***Angemessenheit*: Es muss mit den Elementen der Funktionenmenge und Terminalmenge möglich sein, eine Lösung für das gegebene Anwendungsproblem zu finden.**

- ***Abgeschlossenheit*: Jede Funktion der Funktionenmenge sollte als Argument jeden Wert zulassen, den beliebige andere Funktionen dieser Menge potentiell zurückgeben können sowie jeden Wert, den beliebige terminals (das sind Variablen und Konstanten) annehmen könnten.**

Um die Abgeschlossenheit sicherzustellen und Laufzeitfehler zu vermeiden kann es erforderlich werden, bestimmte, seitens der Programmiersprache vorhandene Funktionen zu ergänzen, bevor man sie in die Funktionenmenge übernimmt. So müssen beispielsweise Divisionen durch Null verhindert werden, weshalb man eine spezielle Form der Divisionsfunktion verwendet, bei der Division durch Null zu einem Ergebniswert von 1 führen. Diese in der GP häufig eingesetzte Funktion (*protected division function*) wird mit dem Symbol % dargestellt.

3.1.2 Initialisierung

In der Initialisierungsphase wird eine Ausgangspopulation von baumstrukturierten Programmen stochastisch erzeugt. Man generiert Programme, indem zunächst mit gleicher Wahrscheinlichkeit zufällig eine Funktion der Funktionenmenge ausgewählt und zur Wurzel des Programmbaumes bestimmt wird. Damit Programme nicht übermäßig groß und komplex werden, legt man eine maximale Baumtiefe fest. Sie entspricht der Anzahl der Kanten des längsten Pfades von der Wurzel zu einem Endpunkt des Baumes. Üblich ist bei Koza eine Maximaltiefe von sechs in der Ausgangspopulation und von 17 in allen späteren Populationen.

Umöglichst vielfältige Programmstrukturen in der Ausgangspopulation zu haben, benutzt man bei der Initialisierung häufig eine als *half-ramping* bezeichnete Vorgehensweise. Alle Baumtiefen zwischen zwei und der Maximaltiefe werden gleich oft erzeugt. Für jede dieser Baumtiefen bestimmt man je die Hälfte der Vertreter auf eine der beiden folgenden Weisen:

- *full*: Alle Pfade im Baum zwischen Wurzel und Endpunkten des Baumes sind gleichlang und entsprechen der Tiefe des Baumes. Dies erreicht man, indem Knoten auf einer Stufe des Baumes, die geringer sind als die festgelegte Baumtiefe, nur durch stochastisch und mit identischen Wahrscheinlichkeiten ausgewählte Elemente aus der Funktionenmenge belegt werden dürfen. Knoten auf der Stufe des Baumes, die seiner Tiefe entsprechen, werden dagegen durch stochastische Auswahl aus der Terminalmenge bestimmt.

- *grow*: Die andere Hälfte der Bäume spezifizierter Tiefe weist unterschiedliche Strukturen auf. Hier werden alle Knoten auf Stufen unterhalb der Baumtiefe durch Elemente aus der Vereinigungsmenge von Funktionenmenge und Terminalmenge belegt. Für die Knoten der letzten Stufe des Baumes werden dagegen nur Elemente der Terminalmenge ausgewählt. In beiden Fällen erfolgt die Auswahl wieder stochastisch mit identischen Wahrscheinlichkeiten.

Duplikate, also identische Individuen, werden für die Ausgangspopulation nicht zugelassen.

3.1.3 Fitnessberechnung

Die Gestaltung der Fitnessfunktion ist von sehr großer Bedeutung für die Effektivität von GP. Folgende Überlegungen sollten einfließen:

- **Die unterschiedliche Qualität verschiedener Programme muss sich im Fitnesswert möglichst differenziert wiederspiegeln.**
- **Auch Teillösungen müssen differenziert bewertbar sein.**
- **In die Fitnessfunktion können auch mehrere Beurteilungskriterien einfließen. Häufig berücksichtigt man an dieser Stelle neben der funktionalen Qualität eines Programmes auch andere Eigenschaften, wie z.B. seine Komplexität.**

Je nach Anwendung kann die Fitness auf sehr unterschiedliche Weise gemessen werden. Der unmittelbar aus der jeweiligen Anwendung abgeleitete Fitnesswert wird als Rohfitness f_r bezeichnet. Häufig arbeitet man bei der GP jedoch nicht mit der Rohfitness, sondern transformiert diesen Wert.

Mit M als Populationsgröße und $b_{j,G}$ ($j = 1, 2, ..., M$) als Bezeichner für ein Individuum j der aktuellen Population in der Generation G gelten folgende Zusammenhänge:

- *Rohfitness (raw fitness, f_r)*: Der ursprünglich ermittelte Fitnesswert ohne prinzipielle Begrenzung. Er ist je nach Aufgabenstellung und Art der Berechnung zu minimieren oder zu maximieren.
  - In vielen Fällen lässt sich Fitness z.B. als Fehlerterm angeben, also als absolute Differenz zwischen der Ausgabe des Programms und der korrekten Ausgabe, summiert über eine Menge von repräsentativen Beispielen (Trainingsdaten). Diese Differenz wird bei jedem Trainingsfall häufig noch quadriert. Je geringer die Summe der quadrierten Abweichungen ist, desto besser ist das Programm.
    Die folgende Gleichung ist ein Beispiel für eine Rohfitnessfunktion:

$$f_{\mathrm{r}_{b_{j,G}}} = \sum_{l=1}^{N} (f_l(b_{j,G}) - a_{l,\mathrm{Korrekt}})^2 \tag{3.1}$$

    Wobei:
    * $f_{\mathrm{r}_{b_{j,G}}}$ die Rohfitness des Individuums j in der Generation G,

  - $*$ N die Anzahl der Trainingsdaten,
  - $*$ $a_{l,\text{Korrekt}}$ die korrekte Ausgabe im Fall l, und
  - $*$ $f_l(b_{j,G})$ Ausgabe des Individuums j im Fall l ist.
- In anderen Fällen kann die Fitness mit Hilfe einer Punktzahl ausgedrückt werden, z.B. im Sinne von richtig erkannten Fällen aus einer Menge von Trainingbeispielen. Hier würde eine hohe Punktzahl einer guten Fitness entsprechen.

- *Standardisierte Fitness (standardized fitness, f_{st})*: Linear umgerechnete Rohfitness, wobei bessere Individuen kleinere standardisierte Fitnesswerte zugeordnet bekommen als schlechtere Individuen. Der bestmögliche standardisierte Fitnesswert sei Null.

$$f_{\text{st}_{b_{j,G}}} = \begin{cases} f_{\text{r}_{b_{j,G}}} & \text{, falls min. } f_{\text{r}_{b_{j,G}}} \text{ gut ist} \\ f_{\text{r}_{\text{max}},G} - f_{\text{r}_{b_{j,G}}} & \text{, falls max. } f_{\text{r}_{b_{j,G}}} \text{ gut ist} \end{cases} \tag{3.2}$$

  Wobei:

  - $f_{\text{st}_{b_{j,G}}}$ die standardisierte Fitness des Individuums j in der Generation G,
  - $f_{\text{r}_{\text{max}},G}$ die mögliche maximale Fitness in der Generation G ist.

- *Angepasste Fitness*, *Justierte Fitness* oder *Adjustierte Fitness (adjusted fitness, f_{a})*: Sie liegt für jedes Individuum zwischen 0 und 1, wobei bessere Individuen größere Werte zugewiesen bekommen. Die Ermittlung der adjustierten Fitness aus der standardisierten Fitness kann durch jede lineare oder nichtlineare Funktion erfolgen, die diese Bedingungen erfüllt. z.B.:

$$f_{\text{a}_{b_{j,G}}} = \frac{1}{1 + f_{st_{b_{j,G}}}} \tag{3.3}$$

  Wobei $f_{\text{a}_{b_{j,G}}}$ die angepasste Fitness des Individuums j in der Generation G ist.

- *Normalisierte Fitness (normalized fitness, f_{n})*: Liegt zwischen 0 und 1, wobei die Summe der normalisierten Fitnesswerte aller Individuen einer Population genau 1 ergibt. Größere Werte entsprechen besseren Individuen.

$$f_{\text{n}_{b_{j,G}}} = \frac{f_{\text{a}_{b_{j,G}}}}{\sum_{j=1}^{M} f_{\text{a}_{b_{j,G}}}} \tag{3.4}$$

  Wobei $f_{\text{n}_{b_{j,G}}}$ die normalisierte Fitness des Individuums j in der Generation G ist.

3.1.4 Selektion

Der Selektionsoperator kann in zwei algorithmische Teilschritte gegliedert werden:

1. Selektionsalgorithmus
2. Auswahlalgorithmus

Selektionsalgorithmus

Der Selektionsalgorithmus ist die Berechnung einer Auswahlwahrscheinlichkeit (*target sampling rate*, t_{sr}) für jedes Individuum, die angibt, mit welcher Wahrscheinlichkeit das Individuum zur Selektion herangezogen wird.

Die wichtigsten Selektionsalgorithmen sind:

1. Fitnessproportionale Selektion
2. Rangbasierte Selektion
3. Wettkampfselektion

Alle drei sind grundsätzliche Formen der *nicht diskriminierenden (not extinctive)* Selektion.[1] Zur näheren Charakterisierung der verschiedenen Selektionsalgorithmen eignet sich unter anderem das Konzept des *Selektionsdrucks*.

Der Selektionsdruck kann anhand der *takeover time* charakterisiert werden. Dies ist jene Anzahl an Generationen, nach der durch wiederholte Anwendung der Selektion alleine eine Population entsteht, die $M - 1$ Kopien des besten Individuums der Ausgangspopulation enthält.

Je länger die *takeover time* ist, desto weicher ist die Selektion, umso niedriger ist also der Selektionsdruck. Niedriger Selektionsdruck entspricht in erster Näherung einer eher globalen Form der Optimumsuche, während hoher Selektionsdruck einer Art stochastischer Gradientensuche entspricht.

[1] Bei der *diskriminierenden* Selektionsform werden einige Individuen in der Population keine Chance erhalten, Nachkommen zu haben.

Fitnessproportionale Selektion Bei der fitnessproportionalen Selektion wird ein Individuum mit umso größerer Wahrscheinlichkeit ausgewählt und in die nächste Generation kopiert, je größer sein Anteil an der Gesamtfitness der Population ist.

Die fitnessproportionale Selektion ist, nicht zuletzt wegen ihrer Einfachheit, die wohl bekannteste Selektionsmethode und wird bei den meisten Einführungen in genetische Lernverfahren zuerst vorgestellt. Bei der fitnessproportionalen Selektion errechnet sich die Auswahlwahrscheinlichkeit proportional zur normalisierten Fitness

$$t_{\mathrm{sr}_{b_{j,G}}} = \frac{f_{\mathrm{n}_{b_{j,G}}}}{\bar{f}_{\mathrm{n}b_G}} \tag{3.5}$$

wobei:

- $t_{\mathrm{sr}_{b_{j,G}}}$ die Auswahlwahrscheinlichkeit des Individuums j in der Generation G,
- $f_{\mathrm{n}_{b_{j,G}}}$ die normalisierte Fitness des Individuums j in der Generation G, und
- $\bar{f}_{\mathrm{n}b_G}$ die durchschnittliche normalisierte Fitness in der Generation G

darstellen.

Bei der Wahl dieser Methode ist jedoch zu bedenken, dass die Gefahr eines abnehmenden Selektionsdruckes bei zunehmender Angleichung der Fitnesswerte in der Population besteht. Dem kann durch eine geeignete Berechnungsmethode der Fitnesswerte entgegengewirkt werden.

Rangbasierte Selektion Um der Gefahr des abnehmenden Selektionsdrucks zu entgehen, kann man auf die *rangbasierte Selektion* ausweichen. Hierbei werden die Individuen nach ihrer normalisierten Fitness aufsteigend sortiert und mit Rangnummern $\mathbf{rank}(b_{j,G})$ (beginnend mit Rang 1) versehen. Die Auswahlwahrscheinlichkeit errechnet sich dann wie folgt:

$$t_{\mathrm{sr}_{b_{j,G}}} = a_{\min} + (a_{\max} - a_{\min})\frac{\mathbf{rank}(b_{j,G}) - 1}{M - 1} \tag{3.6}$$

wobei:

- $t_{\mathrm{sr}_{b_{j,G}}}$ die Auswahlwahrscheinlichkeit des Individuums j in der Generation G,
- $a_{\min}$ die Auswahlwahrscheinlichkeit für das schlechteste Individuum,
- $a_{\max}$ die Auswahlwahrscheinlichkeit für das beste Individuum,
- M die Populationsgröße und
- $\mathbf{rank}(b_{j,G})$ der Rang des Individuums j in der Generation G nach f_{n} sortiert

darstellt.

Da die Auswahlwahrscheinlichkeit eines Individuums größer sein muss als 0, und die Summe aller Auswahlwahrscheinlichkeiten der Individuen einer Population genau M zu ergeben hat, gelten für $a_{\min}$ und $a_{\max}$ folgende einschränkende Bedingungen:

$$1 < a_{\max} < 2 \tag{3.7}$$

$$a_{\min} = 2 - a_{\max} \tag{3.8}$$

Die Auswahlwahrscheinlichkeit jedes Individuums hängt von seiner Position in einer auf der Fitness basierenden Rangordnung aller Mitglieder der aktuellen Population ab. Die absolute Höhe von Fitnessunterschieden zwischen verschiedenen Individuen wird dadurch nebensächlich. Der Selektionsdruck kann direkt über den Wert von $a_{\max}$ gesteuert werden.

Wettkampfselektion Dieser Algorithmus vermeidet Skalierung und erfordert, anders als die rangbasierte Selektion, keine rechenaufwendige Sortierung der Population nach Fitnesswerten. Die Wettkampfselektion integriert Selektions- und Auswahlalgorithmus.

Die übliche Vorgehenweise besteht darin, dass aus der Population ξ Individuen ($2 \leq \xi < M$) mit gleicher Selektionswahrscheinlichkeit $P_{\mathrm{s}} = 1/M$ ausgewählt werden, um dann das beste unter ihnen auszuwählen. Dieser Vorgang wird M-mal wiederholt. Über den Wert von ξ lässt sich der Selektionsdruck steuern. Häufig nimmt man den Wettkampfumfang $\xi = 2$. Man spricht dann von binärer Wettkampfselektion. Mit steigendem ξ nimmt bei konstanter Populationsgröße M der Selektionsdruck zu. Hält man dagegen den Wettkampfumfang ξ konstant und erhöht M, so sinkt der Selektionsdruck.

Auswahlalgorithmus

Nachdem jedem Individuum der Population eine Auswahlwahrscheinlichkeit t_{sr} zugeordnet wurde, wird ein entsprechender Auswahlalgorithmus bestimmt, der Individuen mit großer Auswahlwahrscheinlichkeit t_{sr} bevorzugt. Die Auswahlalgorithmen sind:

- Stochastic Sampling with Replacement
- Stochastic Universal Sampling

Stochastic Sampling with Replacement Es handelt sich hierbei um den gebräuchlichsten Auswahlalgorithmus. Die einzelnen Individuen der Population werden in einem gewichteten Zufallsverfahren ausgewählt, wobei die Gewichtung anhand der zuvor berechneten Auswahlwahrscheinlichkeit (t_{sr}) erfolgt. Dieser Auswahlalgorithmus kann sehr gut mit Hilfe eines Roulettrades veranschaulicht werden, welches in M Sektoren unterschiedlicher Größe unterteilt ist (wobei M die Anzahl der Individuen in der Population darstellt). Die Größe jedes Sektors wird durch die t_{sr} des entsprechenden Individuums bestimmt. Mit m Versuchen kann man nun m Individuen selektieren, wobei die Chance für jedes Individuum ausgewählt zu werden, mit der Größe des entsprechenden Sektors, und somit mit t_{sr} des Individuums korreliert. Man spricht wegen dieser anschaulichen Darstellung auch vom *Roulett-Wheel-Verfahren.*

Problematisch hierbei ist jedoch, dass im Prinzip jedes Individuum mit einer positiven t_{sr} durch Zufall die ganze Folgepopulation ausfüllen kann. Dies stellt vor allem bei kleinen Populationsgrößen ein echtes Problem dar.

Dass t_{sr} und die tatsächliche Anzahl von Kopien eines Individuums weit auseinanderklaffen können, ist der gravierendste Nachteil des Stochastic Sampling with Replacement.

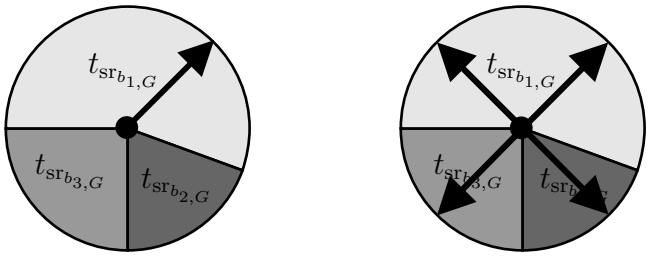


Abbildung 3.2: Auswahlalgorithmus: Stochastic Sampling with Replacement (*links*), Stochastic Universal Sampling (Auswahl von 4 Individuen) (*rechts*)

Stochastic Universal Sampling Dieses Verfahren bietet die beste Lösung für das angesprochene Problem. Die Grundidee besteht darin, alle m Individuen in nur einer Zufallsselektion auszuwählen. Will man z.B. 4 Individuen selektieren, dann legt man um beim Roulett-Beispiel zu bleiben, über das in Sektoren unterteilte Roulettrad ein weiteres Rad mit genau 4 Speichen in gleichem Abstand. Dreht man nun dieses Rad, so kommt nach dem Stillstand jede Speiche über einem bestimmten Sektor des darunter liegenden Roulettrades zu stehen. Damit sind z.B. 4 Individuen in einem Schritt selektiert, und es besteht auch bei kleinen Populationen keine Gefahr, dass ein einzelnes Individuum die ganze Nachfolgepopulation ausfüllt. In Abbildung 3.2 (rechts) wird ein Beispiel von *Stochastic Universal Sampling* gezeigt, bei dem das Individuum mit der größten t_{sr} zwei mal, und die beiden anderen Individuen jeweils ein mal ausgewählt werden.

3.1.5 Elite-Selektion

In vielen Anwendungen wird, unabhängig vom sonst verwendeten Selektionsalgorithmus, das beste Individuum der aktuellen Generation auf jeden Fall in die Population der nächsten Generation übernommen. Man bezeichnet diese Vorgehensweise als *Elite-Selektion (elitist selection)* oder *Elitismus*.

Durch Elite-Selektion ist gewährleistet, dass die besten bisher gefundenen Lösungen eines GP-Laufes nicht mehr aus der Population verlorengehen. Dies wäre sonst aufgrund des stochastischen Charakters von Kreuzung und Mutation nicht sichergestellt. Andererseits vergrößert Elite-Selektion die Gefahr dauerhafter Stagnation auf einem Suboptimum.

3.1.6 Genetische Operatoren

Genetische Operatoren dienen im GP zur Erzeugung von Nachkommen, damit die Population der nächsten Generation eine möglichst bessere Durchschnittsfitness besitzen kann. Kreuzung und Reproduktion werden als primäre Operationen für die GP bezeichnet. Darüber hinaus gibt es weitere Operationen, die man einsetzen kann, aber nicht unbedingt muss. Solche Operationen sind Mutation, Permutation, Editieren und Einkapselung. Diese Operationen seien im Folgenden genauer beschrieben.

Reproduktion (Reproduction)

Im Fall des Reproduktionsoperators wird ein Individuum ausgewählt und unverändert in die neuen Population kopiert.

Kreuzung (Rekombination oder Crossover)

Bei der Kreuzung werden zunächst zwei Individuen als Eltern-Individuen ausgewählt. In beiden Eltern wird anschließend stochastisch und unabhängig voneinander ein Kreuzungspunkt bestimmt. In Abbildung 3.3 sind dies die Knoten + und c von den Eltern-Individuen

$$((a * 5) - (4 + b)) \quad \text{und} \quad (c * (9 + 1)).$$

Im Allgemeinen werden innere Knoten und Endpunkte des Baumes (Endknoten) mit unterschiedlichen Wahrscheinlichkeiten als Kreuzungspunkte gewählt. In [Koza92] wurden innere Knoten mit einer Wahrscheinlichkeit von $p_{\text{ip}} = 0,9$ und Endknoten mit einer Wahrscheinlichkeit von $p_{\text{ep}} = 0,1$ als Kreuzungspunkte ausgewählt. Ist diese Entscheidung getroffen, so wird bei jedem Eltern-Individuum unter dem jeweiligen Knoten stochastisch und mit gleicher Wahrscheinlichkeit einer als Kreuzungspunkt

ausgewählt. Anschließend tauscht man die zwei entstehenden Kreuzungsfragmente (Abbildung 3.3), also die mit dem Kreuzungspunkt als Wurzelknoten beginnenden Teilbäume, zwischen den Elternkopien aus und erhält so zwei Nachkommen von in der Regel unterschiedlicher Größe und Struktur.

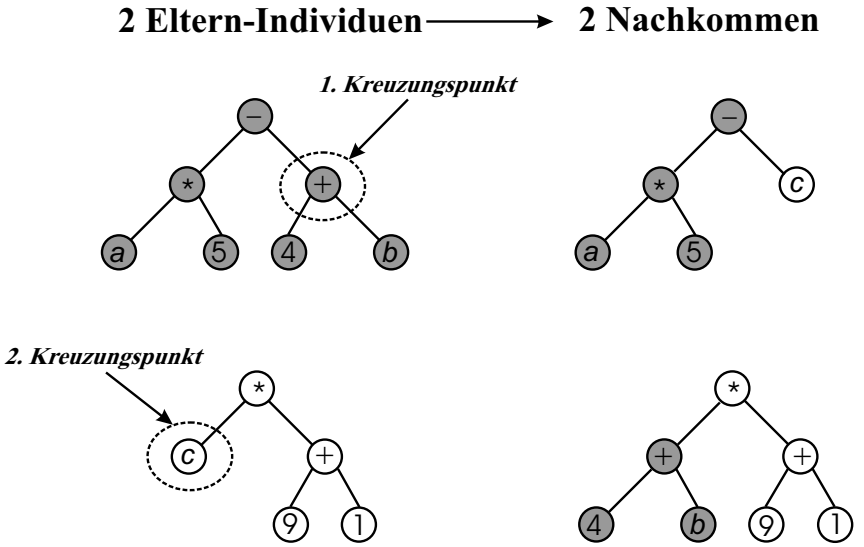


Abbildung 3.3: Kreuzung zwischen zwei Eltern-Individuen in GP

Allerdings werden Nachkommen, welche die maximal erlaubte Tiefe überschreiten, verworfen und stattdessen für Nachkomme 1 das Eltern-Individuum 1 (für Nachkomme 2 das Eltern-Individuum 2) in die neue Population übernommen. Ebenfalls ausgeschlossen wird oft die Möglichkeit, ein Kreuzungsfragment, das nur aus einer Variablen oder Konstanten besteht, an die Wurzel (als Kreuzungspunkt) dem anderen Eltern-Individuum anzufügen.

Mutation (Mutation)

Der durch den Reproduktions- oder Kreuzungsoperator erzeugte Nachkomme kann anschließend mutiert werden. Die Mutationswahrscheinlichkeit p_{m} ist normalerweise sehr klein oder wird null gesetzt.

Eine übliche Form der Mutation ist in Abbildung 3.4 dargestellt. Ein Knoten eines Individuums wird stochastisch als Mutationspunkt bestimmt. Dann wird ebenfalls stochastisch ein neuer Programmbaum generiert, dessen maximal erlaubte Tiefe geringer ist als der sonst zulässige Wert der Baumtiefe. Nun ersetzt man im Individuum den Teilbaum, dessen Wurzel der Mutationspunkt ist, durch den eben generierten Baum. Die so entstandene Mutante wird in die neue Population übernommen, sofern sie die zulässige maximale Baumtiefe nicht überschreitet. Ansonsten kann der Mutationsvorgang entweder wiederholt werden, oder man übernimmt das Ausgangs-Individuum.

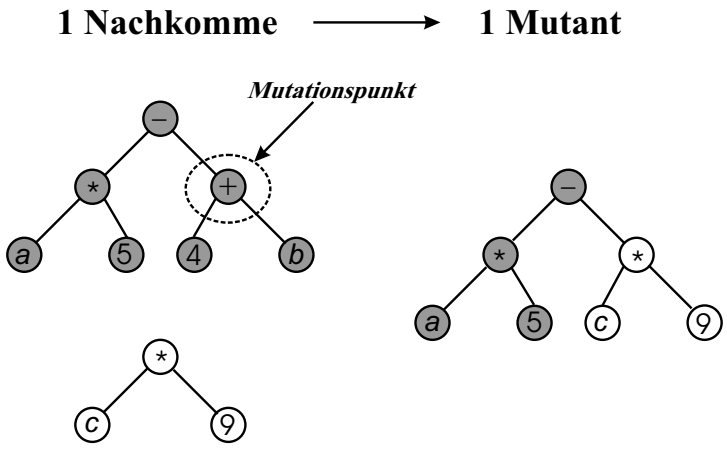


Abbildung 3.4: Mutation in GP

Editieren (Editing)

Die Operation Editieren verkürzt ein Individuum, ohne dass es seine Bedeutung oder seine Fitness verliert. Es gibt nämlich Unterausdrücke, die man durch ein Terminalsymbol ersetzen kann. Als Beispiel betrachtet man den mathematischen Ausdruck $1 + 4$. Er kann durch das Terminal 5 ersetzt werden.

Beispiel: $1 + 4 \overset{\text{Editieren}}{\longrightarrow} 5$

Um die Komplexität der bereits gefundenen Ausdrücke zu reduzieren, wird die Operation Editieren während eines Programmlaufes hin und wieder ausgeführt.

Permutation (Permutation)

Die Operation Permutation vertauscht einen Teilbaum mit einem anderen Teilbaum innerhalb eines Individuums.

Beispiel: $\underbrace{(4 \times a)}_{\text{1.Teilbaum}} - \overbrace{y}^{\text{2.Teilbaum}} \overset{\text{Permutation}}{\longrightarrow} \underbrace{y}_{\text{2.Teilbaum}} - \overbrace{(4 \times a)}^{\text{1.Teilbaum}}$

Einkapselung (Encapsulation)

Bei dieser Operation wird ein Teil eines Individuums durch ein Symbol ersetzt. So kann man z.B. in $(2 \times s) + d$ die Multiplikation $(2 \times s)$ durch einen Ausdruck E ersetzen und erhält $E + d$. Die weitere Verarbeitung erfolgt dann mit dem Symbol E, was für die Ausdrücke unter Umständen eine erhebliche Vereinfachung darstellen kann.

Beispiel: $\underbrace{(2 \times s)}_{E} + d \overset{\text{Einkapselung}}{\longrightarrow} E + d$

3.1.7 Steady-state

In den meisten GP-Implementierungen werden während einer Generation immer alle Individuen der Population durch ihre Nachkommen ersetzt. Man bezeichnet diese Ersetzungs-Strategie als *generational replacement*. In einer als *steady-state* bezeichneten Variante werden dagegen bei jeder Iteration nur wenige Individuen der Population, im Allgemeinen die schlechtesten, ersetzt. Dadurch verbleiben gute Lösungen tendenziell länger in der Population und können den Optimierungsprozess dauerhafter beeinflussen als bei dem *generational replacement*.

3.1.8 Vereinfachungszwang

Der Vereinfachungszwang (*Parsimony Pressure*) ist eine Maßnahme zur Verminderung der übermäßigen großen Komplexität des Ergebnisses. Die Komplexität kann die Programmlänge oder die Tiefe des Individuums sein. Bei der Anwendung des Vereinfachungszwangs wird die Komplexität in die Fitnessfunktion eingeschlossen. Die Gleichung 3.9 zeigt eine einfache Fitnessfunktion mit der hinzu addierten Komplexität.

$$f_{\mathrm{r,p}_{b_{j,G}}} = f_{\mathrm{r}_{b_{j,G}}} + \alpha(G) \cdot C_{b_{j,G}} \tag{3.9}$$

Wobei:

- $f_{\mathrm{r}_{b_{j,G}}}$ die Rohfitness des Individuums j in der Generation G ohne Berücksichtigung des Vereinfachungszwangs
- $f_{\mathrm{r,p}_{b_{j,G}}}$ die Rohfitness des Individuums j in der Generation G unter Berücksichtigung des Vereinfachungszwangs
- $C_{b_{j,G}}$ die Komplexität des Individeums j in der Generation G, und
- $\alpha(G)$ der bestimmter Faktor des Vereinfachungszwangs in der Generation G

ist.

Der Faktor des Vereinfachungszwangs $\alpha(G)$ ist von der Generation abhängig. Am Anfang des GP-Laufes sollte er sehr klein bleiben, damit die GP eine große Freiheit für die Evolution besitzt. Falls die GP ein Ergebnis mit einem akzeptierbaren Fehler findet, kann $\alpha(G)$ erhöht werden.

Der Vorteil des Vereinfachungszwangs kann heute noch nicht völlständig zusammengefasst werden. Ein konstanter Faktor des Vereinfachungszwangs $\alpha(G)$ führt sehr oft zu einem schlechten Ergebnis, und die GP stagniert vorzeitig. Dagegen kann man beim variierten Faktor des Vereinfachungszwangs $\alpha(G)$ eine gute und kurze Lösung bekommen.

3.1.9 Automatisch Definierte Funktionen

Komplexe Problemstellungen lassen sich häufig in verschiedene, besser überschaubare Teilprobleme zerlegen. Im nächsten Schritt versucht man dann, diese Teilprobleme zu lösen. Anschließend verwendet man die Lösung der Teilprobleme, um das komplexe Gesamtproblem zu lösen. So entsteht eine hierarchische und modulare Lösung des Gesamtproblems.

Ein solches Vorgehen ist typisch und führt häufig zu besserer Verständlichkeit der Programme und macht somit den Prozess der Problemlösung und Systementwicklung effizienter. Lösungen von Teilproblemen können wiederholt aufgerufen oder, eventuell mit kleinen Modifikationen, an verschiedenen Stellen eines Programmes verwendet werden.

Um die Effizienz von GP bei komplexen Problemstellungen zu verbessern, sind verschiedene Modularisierungskonzepte vorgeschlagen worden, von denen sich die *Automatisch Definierten Funktionen (ADFs)* in [Koza94] am weitesten durchgesetzt haben. Sie sind inzwischen eine Standarderweiterung in vielen GP-Systemen geworden.

Im Wesentlichen geht es bei dieser Erweiterung der GP darum, für konkrete Problemstellungen modular aufgebaute Programme zu generieren, die aus ADFs und einem Hauptprogramm bestehen. Dabei werden die konkreten Inhalte des Hauptprogrammes und der ADFs durch evolutionäre Mechanismen automatisch erzeugt. Es ist nicht vorgegeben, ob und in welcher Weise die ADFs im Rahmen eines GP-Individuums aufgerufen und eingesetzt werden.

Allerdings muss im Vorfeld eines GP-Laufes entschieden werden, wie die grobe Programmarchitektur aussehen soll, insbesondere, wieviele ADFs jedes Programm der GP-Population hat und wieviele Parameter (Argumente) einer Funktion übergeben werden.

3.2 Durchführung des GP-Laufes

Bevor ein GP-Lauf gestartet werden kann, sind einige Entscheidungen zu treffen, die in die folgenden fünf Schritte untergliedert werden:

1. Festlegen der Terminalmenge,
2. Festlegen der Funktionenmenge,
3. Bestimmen eines geeigneten Fitnessmaßes
4. Bestimmen von Strategieparameterwerten,
5. Bestimmen des Abbruchkriteriums und einer Entscheidungsregel, um das GP-Ergebnis festzulegen.

Da sich Computerprogramme in der GP aus Elementen der Funktionenmenge und der Terminalmenge zusammensetzen, definiert der Benutzer in den ersten zwei vorbereitenden Schritten faktisch eine Sprache, in der GP die spätere Lösung ausdrückt.

Ein wichtiger Strategieparameter von GP ist die Populationsgröße M. GP verwendet in der Grundform vergleichsweise große und häufig nicht weiter strukturierte Populationen von oft mehreren Tausend Individuen.

Für den Wert der maximalen Anzahl von Generationen $G_{\max}$ gibt es keine generellen Empfehlungen. Er hängt überwiegend von der Komplexität der Anwendung und vom Leistungsumfang der verfügbaren Hardware ab. Koza verwendet in seinen Experimenten häufig 51 Generationen (inkl. Ausgangspopulation). Ein GP-Lauf wird andererseits oft auch vorzeitig abgebrochen, wenn eine vollkommen korrekte Lösung für das bearbeitete Problem gefunden ist. Ansonsten gilt in der Regel das Beste im gesamten GP-Lauf gefundene Programm als Schlussergebnis der GP.

GP-Parameter	Einstellung
Populationsgröße M	4000
Max. Generation $G_{\max}$	50
Kreuzungswahrscheinlichkeit p_{c}	0,9
Reproduktionswahrscheinlichkeit p_{r}	0,1
Lage Kreuzungspunkt: $p_{\mathrm{ip}}/p_{\mathrm{ep}}$	$0,9/0,1$
Max. Baumtiefe Ausgangspopulation D_{initial}	6
Max. Baumtiefe während des restl. Laufes D_{created}	17
Mutationswahrscheinlichkeit p_{m}	0,0
Premutationswahrscheinlichkeit p_{p}	0,0
Editierenswahrscheinlichkeit p_{ed}	0,0
Einkapselungswahrscheinlichkeit p_{en}	0,0
Initialisierungsmethode	*half-ramping*
Selektionsverfahren	Wettkampf mit $\xi = 7$
Elite-Selektion	Nein
Fitnesstyp	standardisierte Fitness

Tabelle 3.1: Standardeinstellungen für wichtige GP-Parameter

Tabelle 3.1 nennt Standardeinstellungen der wesentlichen GP-Strategieparameter, die von Koza in [Koza94] vorgeschlagen wurden. Aufgrund verschiedener stochastischer Einflüsse in GP können die Ergebnisse einzelner Läufe, auch bei der gleichen Parametereinstellung, stark streuen. Nachdem die fünf vorbereitenden Schritte abgeschlossen sind, kann ein GP-Lauf durchgeführt werden.

Schritt 1: Initialisierung

In der Initialisierungsphase wird eine Ausgangspopulation $P(0)$ von M baumstrukturierten Programmen stochastisch erzeugt.

Schritt 2: Bewerten der Ausgangslösungen

In diesem Schritt wird jedes Individuum der Ausgangspopulation $P(0)$ anhand der Fitnessfunktion in der bereits beschriebenen Weise bewertet. Da alle Programme zufälligen Charakter haben, sind die Fitnesswerte in der Ausgangspopulation im Allgemeinen extrem schlecht. Dennoch werden bessere und schlechtere Lösungen vorliegen, die sich anhand ihrer Fitnesswerte differenzieren lassen.

Schritt 3: Selektion

- In diesem Schritt wird zuerst eine Auswahlwahrscheinlichkeit (t_{sr}), die auf der Basis der Fitnesswerte erfolgt, für jedes Individuum berechnet.
- Danach werden M Individuen anhand dieser Auswahlwahrscheinlichkeit gezogen und damit als Eltern-Individuen ausgewählt.

Schritt 4: Bilden der neuen Population

Die folgenden Teilschritte werden so oft durchlaufen, bis die zunächst leere neue Population den Umfang von M Individuen (Programmen) erreicht hat.

Schritt 4.1: Manche GP, z.B. [Wein97], haben die Möglichkeit, ein neues Individuum mit der Wahrscheinlichkeit p_{creat} stochastisch zu erzeugen. Im anderen Fall wird der Reproduktions-, Kreuzungs- oder Mutationsoperator verwendet.

Schritt 4.2: Operatorwahl Ein Operator (entweder Kreuzungsoperator oder Reproduktionsoperator) wird auf der Basis der Kreuzungswahrscheinlichkeit p_{c} und der Reproduktionswahrscheinlichkeit p_{r} ausgewählt.

Schritt 4.3: Stochastische Elternwahl und Erzeugen von Nachkommen

- Im Fall des Reproduktionsoperators wird ein Eltern-Individuum mit gleicher Wahrscheinlichkeit $1/M$ ausgewählt und unverändert in die neue Population kopiert.
- Im Fall der Kreuzung werden zunächst zwei Eltern-Individuen mit gleicher Wahrscheinlichkeit $1/M$ ausgewählt. Bei beiden Eltern-Individuen wird danach der Kreuzungsoperator angewendet, um zwei Nachkommen zu erzeugen. Anschließend werden die zwei Nachkommen in die neue Population übernommen.

Mit der Wahrscheinlichkeit p_{m} können die Nachkommen anschließend durch den Mutationsoperator verändert werden.

Schritt 5: Weiter mit Schritt 2 bis ein Abbruchkriterium greift

Der geschilderte Ablauf iteriert, bis ein Abbruchkriterium greift. So entstehen im Verlauf der Zeit aus den zufälligen Ausgangslösungen sukzessiv bessere Lösungen des gegebenen Anwendungsproblems.

Oft empfiehlt sich eine manuelle Nachbearbeitung der Schlusslösung, um die Lesbarkeit und Verständlichkeit zu verbessern. Abbildung 3.5 verdeutlicht den GP-Lauf nochmals im Überblick.

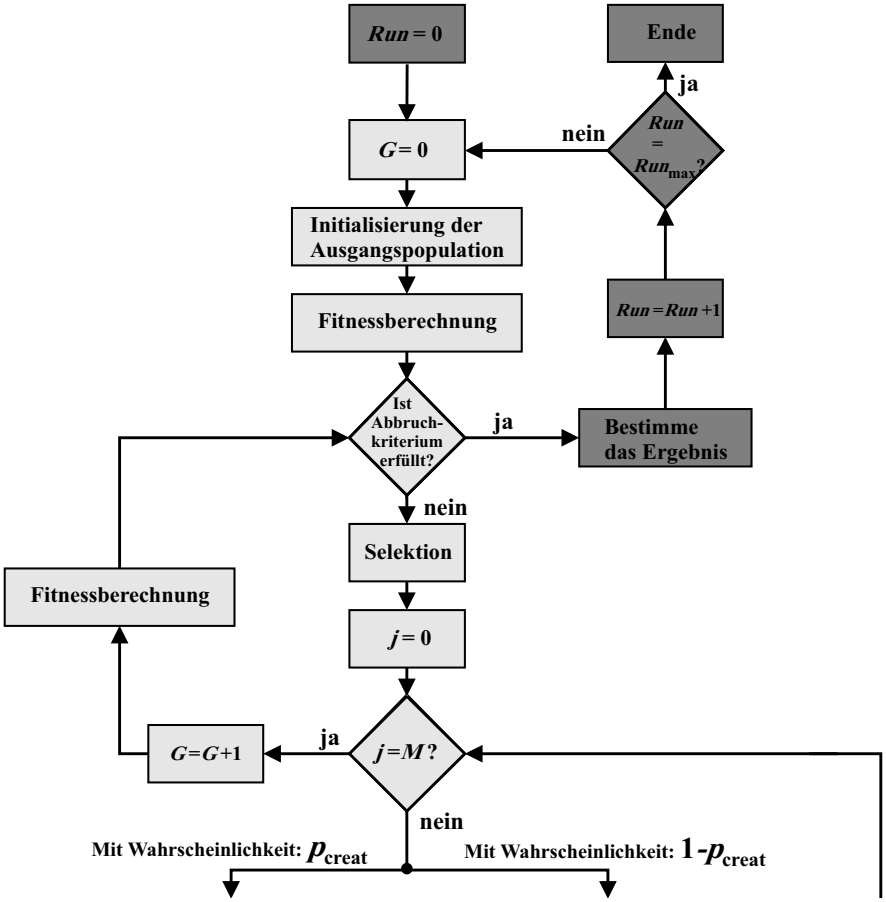


Abbildung 3.5: Ablauf der GP

3.3 Genetische Programmierung mit kontextfreier Grammatik

GP war in [Koza92] ursprünglich zur automatischen Erzeugung von Programmen in LISP entwickelt. Die Sprache LISP hat nur sehr wenige syntaktische Grundregeln. Da höhere Programmiersprachen (C, C++, Java, usw.) durch eine kontextfreie Grammatik beschrieben sind, ist es möglich, die kontextfreie Grammatik als Basis für die automatische Erzeugung von Programmiersprachen einzusetzen.

Eine GP, deren Evolutionsprozess auf einer bestimmten kontextfreien Grammatik basiert, wird hier *genetische Programmierung mit kontextfreier Grammatik (CTF-GP)* genannt.

CTF-GP erscheint in den verschiedensten Veröffentlichungen. In [Hoer96, Gey97] werden Ableitungsbäume einer bestimmten kontextfreien Grammatik als Struktur von Individuen (Programmen) verwendet. Dabei erzeugen der Kreuzungs- und Mutationsoperator immer syntaktisch korrekte Nachkommen. Diese Idee ist ähnlich wie die Arbeit von Whigham in [Whig96, Whig01]. In [Kel96, Free98, One99, One01] sind die Individuen der CTF-GP nicht mehr in Ableitungsbäumen strukturiert, sondern in eine lineare Form umgewandelt. Weitere Veröffentlichungen von CTF-GP findet man in [Wong95, Pat97].

3.3.1 Kontextfreie Grammatik

Eine formale Sprache besteht aus einer Menge von Zeichenketten, die durch eine Grammatik erzeugt werden. Eine Grammatik ist sowohl das mathematische System zur Definition einer formalen Sprache, als auch ein Satz von Regeln zur Feststellung der syntaktischen Gültigkeit eines konkreten Satzes. Im Folgenden werden nur kontextfreie Grammatiken betrachtet, die höhere Programmiersprachen beschreiben. Eine kontextfreie Grammatik ist in der Literatur [Aho72, Aho88] üblicherweise folgendermaßen definiert:

Definition 3.1 *Eine kontextfreie Grammatik ist ein 4-Tupel* $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$*, wobei*

$\mathcal{N}$*: die Menge der Nichtterminalsymbole,*

$\mathcal{T}$*: die Menge der Terminalsymbole, und*

$\mathcal{P}$*: die Relation* $\mathcal{N} \times (\mathcal{N} \cup \mathcal{T})^*$ *ist.*

S: ist ein besonderes Nichtterminalsymbol, auch Startsymbol genannt.

Definition 3.2 *Ein Element $(A, \beta) \in \mathcal{P}$ heißt Ableitung (oder Produktion) und wird abkürzend $A \longrightarrow \beta$ geschrieben.*

Eine Ableitung ist somit eine Ersetzungsregel für ein Nichtterminalsymbol A, wobei dieses Nichtterminalsymbol durch einen String (Zeichenkette) β aus (Nichtterminal- und) Terminalsymbolen ersetzt wird.

Existieren mehrere Ableitungen $(A, \beta_1), (A, \beta_2), ..., (A, \beta_n) \in P$, so schreiben wir hierfür:

$$A \longrightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n$$

Definition 3.3 *Die Menge $\mathcal{P}_{A_0} \subseteq \mathcal{P}$ aller möglichen Ableitungen für ein Nichtterminalsymbol A_0*

$$\mathcal{P}_{A_0} = \{(A, \beta) \in \mathcal{P} \ : \ A = A_0\} \tag{3.10}$$

heißt auch die Menge der Alternativen für A_0. Die Entscheidung für die Anwendung einer konkreten Ableitung $(A_0, \beta_k) \in \mathcal{P}_{A_0}, \ (k = 1, 2, ..., n)$ nennen wir Auswahl.

Beispiel 3.1 (Das boolsche 4-zu-1-Multiplexer Problem) *Alle boolschen Ausdrücke, die aus zwei Adressbits (a_0, a_1) und vier Datenbits (d_0, d_1, d_2, d_3) bestehen, und mit den boolschen Operationen **NOT**, **OR**, **AND** und **IF** verknüpft sind, können durch eine formale Sprache beschrieben werden, die durch die Grammatik $\mathcal{G}_{4:1-\mathrm{Mux}}$ bestimmt ist:*

$$\begin{aligned} \mathcal{G}_{4:1-\mathrm{Mux}} &= (\mathcal{N}, \mathcal{T}, \mathcal{P}, S) \\ \mathcal{N} &= \{F_0, F_1\} \\ \mathcal{T} &= \{a_0, a_1, d_0, d_1, d_2, d_3, \textbf{\textit{NOT}}, \textbf{\textit{OR}}, \textbf{\textit{AND}}, \textbf{\textit{IF}}\} \end{aligned}$$

$\mathcal{P}$ besteht aus den folgenden Ableitungen:

$$\begin{array}{lcl} S & \longrightarrow & F_1 \\ F_1 & \longrightarrow & F_0 \mid \\ & & \textbf{\textit{NOT}} \ F_1 \mid \\ & & \textbf{\textit{OR}} \ F_1 \ F_1 \mid \\ & & \textbf{\textit{AND}} \ F_1 \ F_1 \mid \\ & & \textbf{\textit{IF}} \ F_1 \ F_1 \ F_1 \\ F_0 & \longrightarrow & a_0 \mid a_1 \mid \\ & & d_0 \mid d_1 \mid d_2 \mid d_3 \end{array}$$

Das Programm:

$$\textbf{\textit{IF}}(a_0, (\textbf{\textit{IF}}(a_1, d_3, d_2)), (\textbf{\textit{IF}}(a_1, d_1, d_0)))$$

ist beispielsweise ein gültiger Satz, der einer Lösung des boolschen 4-zu-1-Multiplexer Problems entspricht.

Bei der Erzeugung eines Satzes werden Strings aus Nichtterminal- und ggf. Terminalsymbolen durch Anwendung passender Produktionen auf die Nichtterminalsymbole weiter abgeleitet, bis der String nur noch aus Terminalsymbolen besteht. Im Beispiel besteht der gültige Satz nur noch aus den Terminalsymbolen $a_0, a_1, d_0, d_1, d_2, d_3, \mathbf{NOT}, \mathbf{OR}, \mathbf{AND}, \mathbf{IF}$.

3.3.2 Grundkonzept der CTF-GP

Es gibt verschiedene Konzepte der CTF-GP. In dieser Arbeit wird Bezug auf die Bücher von [Hoer96, Gey97] genommen.

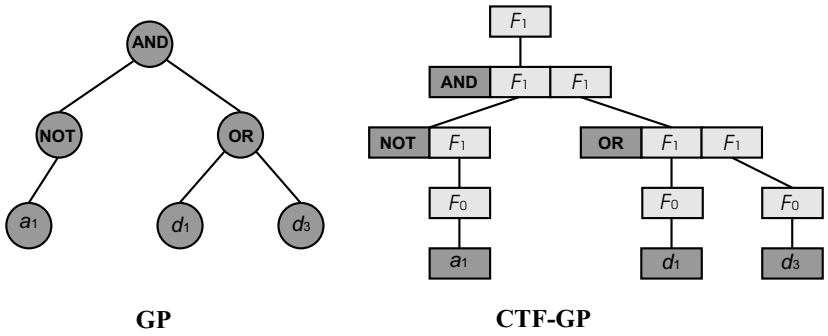


Abbildung 3.6: Baumartige Struktur der GP (*links*) und Ableitungsbaum der CTF-GP (*rechts*)

Die wichtigsten Hauptunterschiede zwischen CTF-GP und GP in [Koza92] sind folgende:

- GP wurde zur automatischen Erzeugung von LISP-Programmen entwickelt, deren Individuen auf einer LISP-baumartigen Struktur basieren. Dagegen besteht ein Individuum in CTF-GP aus Ableitungsbäumen einer bestimmten kontextfreien Grammatik.

  Abbildung 3.6 zeigt deutlich den Unterschied zwischen den beiden Individuen dieser Grammatiken. Offensichtlich ist jede baumartige Struktur der GP ein Ableitungsbaum, jedoch nicht jeder Ableitungsbaum ist passend zur baumartigen Struktur.

- Die genetischen Operatoren basieren in CTF-GP auf der Syntax einer kontextfreien Grammatik. Dagegen verwenden die genetischen Operatoren in GP die Syntax einer solchen kontextfreien Grammatik nicht.

Der Ablauf der CTF-GP soll ähnlich wie der Ablauf der GP aussehen. Aufgrund der Anwendung einer kontextfreien Grammatik müssen aber

- die Wahl der Funktionenmenge und Terminalmenge,

- das Initialisierungsverfahren und
- die genetischen Operatoren, d.h. Kreuzung und Mutation,

angepasst werden. Weitere Prozesse in CTF-GP, wie z.B. Selektion oder Fitnessberechnung, sind identisch mit den entsprechenden Prozessen der GP.

Funktionenmenge und Terminalmenge

Funktionenmenge und Terminalmenge werden durch eine kontextfreie Grammatik ersetzt. Die Wahl einer kontextfreien Grammatik ist besonders wichtig für CTF-GP. Da sie die Struktur des Individuums bestimmt, beeinflusst sie damit auch die Verteilung der Individuen in der Ausgangspopulation, und der Wirkung der genetischen Operatoren, wie Kreuzung und Mutation.

Die kontextfreie Grammatik $\mathcal{G}_{4:1-\mathrm{Mux}}$ im Beispiel 3.1 zeigt eine kontextfreie Grammatik, die der Funktionenmenge:

$$\mathcal{F} = \{\mathbf{NOT}, \mathbf{OR}, \mathbf{AND}, \mathbf{IF}\} \tag{3.11}$$

und der Terminalmenge:

$$\mathcal{T} = \{a_0, a_1, d_0, d_1, d_2, d_3\} \tag{3.12}$$

für das boolsche 4-zu-1-Multiplexer Problem in [Koza92] entspricht.

Initialisierung

Bei der Initialisierung wird eine Ausgangspopulation aus M Zufallsableitungsbäumen erzeugt, wobei M für die Größe dieser Population steht. Es ist nun aber für die Effizienz des Verfahrens von großer Bedeutung, wie diese Bäume erzeugt werden. Dies wird durch das folgende Beispiel offensichtlich.

Wie man aus der kontextfreien Grammatik $\mathcal{G}_{4:1-\mathrm{Mux}}$ im Beispiel 3.1 gezeigten Notation erkennt, gibt es für jedes Nichtterminalsymbol eine gewisse Anzahl möglicher Ableitungen. F_0 kann beispielsweise nach a_0, a_1, d_0, d_1, d_2 oder d_3 abgeleitet werden. Die einfachste Form der Erzeugung eines zufälligen Ableitungsbaumes besteht darin, zunächst mit dem Startsymbol F_1 zu beginnen und dieses um eine der fünf möglichen Ableitungen zu erweitern. Dann wird jedes noch verbleibende Nichtterminalsymbol solange zufällig um eine seiner möglichen Ableitungen erweitert, bis ein vollständiger Ableitungsbaum entstanden ist, oder die maximale Ableitungstiefe erreicht wurde. Beim Erreichen der maximalen Ableitungstiefe muss der unfertige Ableitungsbaum verworfen, und ein neuer Versuch gestartet werden. Die Ableitungstiefe wird hier

durch die Anzahl der Nichtterminalsymbole definiert. Abbildung 3.6 (rechts) zeigt beispielhaft die Zufallserzeugung des Wortes:

$$\mathbf{AND}((\mathbf{NOT}(a_1)), (\mathbf{OR}(d_1, d_3)))$$

In [Gey97] wurde gezeigt, dass bei dieser einfachen Art der Zufallserzeugung kürzere Worte mit weitaus höherer Wahrscheinlichkeit in der Ausgangspopulation vertreten sind, als längere Worte. Auch der naheliegende Gedanke, doppelte Individuen zu eliminieren, bringt keine tatsächliche Verbesserung. Geyer-Schulz schlug in [Gey97] noch weitere Initialisierungsverfahren vor, diese sind jedoch sehr zeitaufwendig.

Kreuzung

Es ist darauf zu achten, dass der durch den Kreuzungsoperator entstandene Ableitungsbaum wieder eine vollständige, korrekte Ableitung ergibt. Dies kann erreicht werden, indem man in den zu kreuzenden Bäumen jeweils stochastisch und unabhängig voneinander zwei gleiche Nichtterminalsymbole heraussucht, und den so gefundenen Teilbaum des ersten Ableitungsbaumes durch den gefundenen Teilbaum des zweiten Ableitungsbaumes ersetzt.

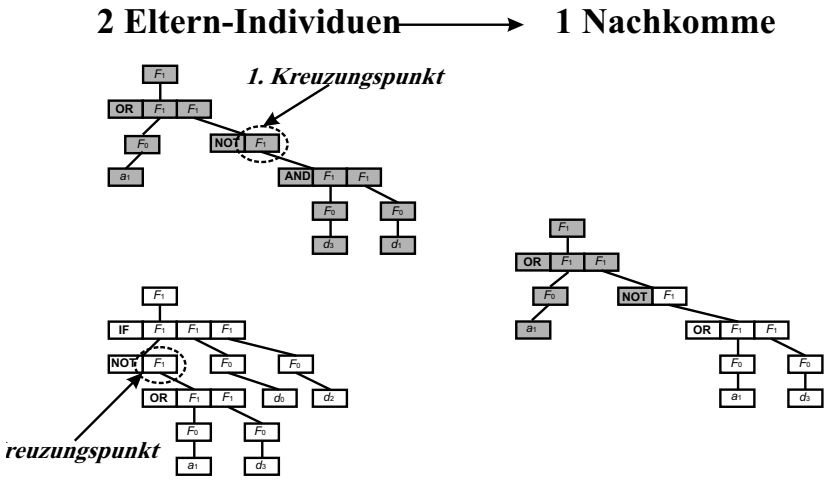


Abbildung 3.7: Kreuzung zwischen zwei Eltern-Individuen in CTF-GP von [Hoer96]

In Abbildung 3.7 wird anhand eines Beispiels demonstriert, wie dies funktionert. In den beiden Ableitungsbäumen des Eltern-Individuums

$$\mathbf{OR}(a_1, \mathbf{NOT}(\mathbf{AND}(d_3, d_1))) \quad \text{und} \quad \mathbf{IF}(\mathbf{NOT}(\mathbf{OR}(a_1, d_3)), d_0, d_2)$$

wird jeweils ein Knoten mit dem Symbol F_1 zufällig herausgesucht. Der erzeugte neue Baum im unteren Teil der Abbildung ist eine Kopie des ersten Baumes, mit dem Unterschied, dass der Teilbaum unterhalb des Kreuzungspunktes durch den Teilbaum des zweiten Ableitungsbaumes ersetzt wurde. Der resultierende Nachkomme $\mathbf{OR}(a_1, \mathbf{NOT}(\mathbf{OR}(a_1, d_3)))$ ist wiederum syntaktisch korrekt.

In CTF-GP von [Hoer96] wird nur ein Nachkomme aus den zwei Eltern-Individuen durch den Kreuzungsoperator erzeugt. Es ist jedoch möglich CTF-GP neu zu implementieren, so dass der Kreuzungsoperator zwei Nachkommen aus zwei Eltern-Individuen erzeugen kann.

Mutation

Ein Teilbaum des zu mutierenden Individuums, beginnend mit einem Nichtterminalsymbol, wird ähnlich dem zuvor erläuterten Kreuzungsoperator durch einen zufälligen Teilbaum, beginnend mit dem gleichen Nichtterminalsymbol, ausgetauscht. Dies stellt sicher, dass der derart modifizierte Baum wiederum eine korrekte Ableitung darstellt.

3.4 Zusammenfassung

GP ist eine automatische Erzeugung von Computerprogrammen, die ursprünglich für die Programmiersprache LISP entwickelt wurde. Ihr Ablauf orientiert sich an der Evolution der Natur. Computer sollen damit in die Lage versetzt werden, sich weitgehend selbst zu programmieren.

Alle möglichen ausführbaren Computerprogramme lassen sich aus einer bestimmten Menge von Funktionen und Terminalen zusammensetzen. Die Funktionenmenge und die Terminalmenge müssen gut genug definiert sein, so dass die gesuchte Lösung (Programm) eines Problems damit ausgedrückt werden kann. Um die syntaktische Korrektheit der durch die genetischen Operatoren neu entstandenen Programme zu gewährleisten, muss jede Funktion neben allen Terminalen auch jedes Ergebnis als Eingabe akzeptieren, das eine andere Funktion zurückliefern kann. Alle Funktionen unterliegen damit der Einschränkung, den gleichen Typ zurückzugeben und ausschließlich Argumente dieses Typs zu akzeptieren.

In der Ausgangspopulation sollen möglichst vielfältige Programmstrukturen enthalten sein. Man benutzt daher bei der Initialisierung häufig ein *half-ramping*-Verfahren.

Die Gestaltung der Fitnessfunktion ist von sehr großer Bedeutung für die Effektivität von GP, da die Fitnessfunktion als Orientierung für die Evolution dient. Die unterschiedliche Qualität verschiedener Individuen muss sich im Fitnesswert möglichst differenziert widerspiegeln.

Bei allen Selektionsverfahren erhält jedes Individuum in GP eine Chance, abhängig von seinem Fitnesswert, Nachkommen zu haben.

Kreuzung und Reproduktion sind primäre genetische Operationen zur Erzeugung von Nachkommen. Dagegen werden weitere genetische Operatoren, wie z.B. Mutation, Editieren, Permutation, Einkapselung mit sehr niedriger Wahrscheinlichkeit oder gar nicht verwendet.

CTF-GP ist eine weitere Form von GP. Der Hauptunterschied zu GP besteht darin, dass die Individuen in der CTF-GP keine baumartige Struktur der Funktionen- bzw. Treminalmenge, sondern Ableitungsbäume einer bestimmten kontextfreien Grammatik, verwenden. Daher müssen die Initialisierung sowie die genetischen Operatoren an die Syntax der kontextfreien Grammatik angepasst werden.

4 Positive Effekte des Mutationsoperators in der genetischen Programmierung

Das Wesen der Kreuzungs- und Mutationsoperatoren wurden bereits in verschiedenen Literatur-Stellen debattiert. Gewöhnlich verwendet die GP den Mutationsoperator sehr selten oder nicht. Der Kreuzungsoperator ist dagegen der vorherrschende Operator beim Erzeugen von Nachkommen. In [Koza92] wird behauptet, dass der Kreuzungsoperator der Hauptoperator ist, der zum Erfolg bei der Entwicklung eines gewünschten Computerprogramms führt.

In diesem Kapitel wird ein Experiment gezeigt, welches GP mit den standardisierten Parameterseinstellungen, d.h. nur mit dem Kreuzungsoperator als Hauptoperator bei der Erzeugung von Nachkommen, nicht lösen kann. Durch den Mutationsoperator wird aber die GP verbessert. Dabei wird in Abschnitt 4.1 eine *Building Block Hypothese* kurz erläutert, die die Wirkung des Kreuzungsoperators in der GP begründen kann. Das Konzept eines linearen Schieberegisters mit Rückkopplung, das als Aufgabe für die GP dient, wird in Abschnitt 4.2 präsentiert. Abschnitt 4.3 stellt die Problemstellung und die Problemanalyse dar. Ergebnisse aus den Experimenten und die Diskussion wird im Abschnitt 4.4 gezeigt.

4.1 Building Block Hypothese

Die dominierende Rolle des Kreuzungsoperators bei der Erzeugung einer Lösung eines Problems wird durch ein Prinzip unterstützt, das *Building Block Hypothese* ([Gold89]) genannt wird. In der *Building Block Hypothese* glaubt man, dass die Kreuzung kleinere *Fit-Building-Blocks* der Eltern kombiniert, dadurch werden größere *Fit-Building-Blocks* in den Nachkommen entstehen.

Dagegen ist der Mutationsoperator ein nur relativ unwichtiger Operator, da er keine *Fit-Building-Blocks* erzeugen kann. Der Mutationsoperator wird mit einer sehr geringen Wahrscheinlichkeit eingesetzt, um die Stagnation der Population während des GP-Laufes zu verhindern, indem sie völlig neue Aspekte, neue Richtungen in den Prozess einbringt.

In der Natur bezeichnet die biologische Kreuzung die Neukombination von Erbmaterial, wie sie bei Organismen mit geschlechtlicher Fortpflanzung auftritt. Die Nachkommen erhalten damit jeweils einen Teil des Erbmaterials des Vaters und der Mutter. Durch diesen Vermischungseffekt können sich vorteilhafte Teile des Erbmaterials in einem Nachkommen vereinen und dort einen Überlebensvorteil bewirken. Die Vermischung von Erbgut bringt daher Vorteile für die Art im Sinne verbesserter Anpassungsmöglichkeiten mit sich.

Jedoch hat die biologische Kreuzung Unterschiede zum Kreuzungsoperator in der GP, z.B. es gibt Arten-, Spezies-Beschränkungen und Geschlechtsbedingungen, außerdem ist das Kreuzen durch die biologische Kreuzung nur an einer bestimmten Position möglich, usw.

In der letzten Zeit präsentieren mehrere GP-Forscher, z.B. die Arbeiten von Banzhaf in [Ban96] und Luke in [Luke97], neue theoretische Argumente und Ergebnisse aus der Erfahrung, dass der Mutationsoperator nützlicher ist, als man in der Vergangenheit angenommen hat. In [Che97] schlägt Chellapila sechs verschiedene Mutationsoperatoren vor, die effektiver als der konventionelle Kreuzungsoperator beim Lösen von gleichen Aufgaben wie in [Koza92] sind. Angeline demonstriert ebenfalls in [Ang97a, Ang97b], dass seine zwei Mutationsoperatoren mehr als der konventionalle Kreuzungsoperator leisten.

4.2 Lineares Schieberegister mit Rückkopplung

Ein *Schieberegister mit Rückkopplung* ([Schn96]) besteht aus zwei Teilen, nämlich einem *Schieberegister* und einer *Rückkopplungsfunktion*. Das Schieberegister enthält eine Bitfolge. Die Länge eines Schieberegisters wird in Bit ausgedrückt; beträgt sie L Bit, so spricht man von einem L-Bit-Schieberegister. Immer wenn ein Bit benötigt wird, werden alle Bits im Schieberegister um ein Bit nach rechts verschoben. Das neue Bit ganz rechts wird abhängig von den anderen Bits im Register berechnet. Das Schieberegister liefert ein Bit Ausgabe, oft das niederwertigste Bit. Die Periode eines Schieberegisters ist die Länge der Ausgabefolge vor der ersten Wiederholung.

Die einfachste Variante eines Schieberegisters mit Rückkopplung ist das *Schieberegister mit linearer Rückkopplung* oder *linear feedback shift register LFSR*. Die Rückkopplung besteht einfach aus einer boolschen Operation **XOR** bestimmter Bits des Registers.

Abbildung 4.1 stellt ein 6-Bit-Schieberegister mit linearer Rückkopplung, das ein Rückkopplungspolynom $f(x) = x^6 + x^5 + x^4 + x + 1$ besitzt, dar.

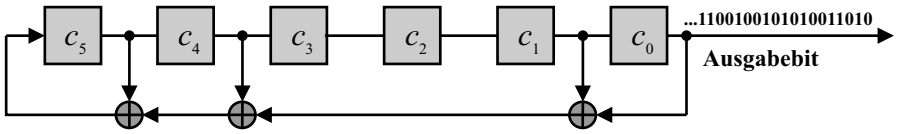


Abbildung 4.1: Ein 6-Bit-Schieberegister mit linearer Rückkopplung mit dem Rückkopplungspolynom $f(x) = x^6 + x^5 + x^4 + x + 1$

Das Ausgabebit a_n kann durch die folgende Rückkopplungsfunktion definiert werden:

$$a_n = a_{n-6} \oplus a_{n-5} \oplus a_{n-2} \oplus a_{n-1} \tag{4.1}$$

wobei $\oplus$ die boolsche Operation **XOR** ist.

Die Rückkopplungsfunktion des L-Bit-LFSRs kann durch die Untersuchung von $2L$ Ausgabebits berechnet werden.

$$\begin{bmatrix} a_6 \\ a_7 \\ a_8 \\ a_9 \\ a_{10} \\ a_{11} \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\ a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ a_4 & a_5 & a_6 & a_7 & a_8 & a_9 \\ a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} \end{bmatrix} \oplus \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} \tag{4.2}$$

Hier: $c_0 = 1$, $c_1 = 1$, $c_2 = 0$, $c_3 = 0$, $c_4 = 1$, $c_5 = 1$

4.3 Problemstellung und Problemeigenschaft

4.3.1 Problemstellung

Für die Problemstellung sind die folgenden 64 Ausgabebits eines 32-Bit-LFSRs $a_0, a_1, a_2, ..., a_{63}$ gegeben:

1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,1,1,0,1,1,0,1,
1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0

Die Rückkopplungsfunktion des 32-Bit-LFSRs ist:

$$a_n = a_{n-25} \oplus a_{n-27} \oplus a_{n-29} \oplus a_{n-30} \oplus a_{n-31} \oplus a_{n-32} \tag{4.3}$$

Diese Rückkopplungsfunktion ist willkürlich ausgewählt, sie kann jedoch eine Pseudozufallssequenz der Länge $2^{32} - 1$ generieren, bevor eine Wiederholung auftritt.

Die Aufgabe des Experiments ist es, aus diesen gegebenen 64 Ausgabebits des 32-Bit-LFSRs die Rückkopplungsfunktion (4.3) mit Hilfe von GP zu gewinnen. Zwei Experimente werden durchgeführt, deren Kreuzungswahrscheinlichkeit p_{c} und Mutationswahrscheinlichkeit p_{m} folgendermaßen eingestellt sind:

- **1. Experiment**: GP mit $p_c = 0,9$ und $p_m = 0,0$
- **2. Experiment**: GP mit $p_c = 0,0$ und $p_m = 0,9$.

Jedes Experiment besteht aus zehn Versuchen, ein Versuch umfasst 50 GP-Läufe.

4.3.2 GP-Einstellung

Beim Experiment wird **die genetische Programmierung mit kontexfreier Grammatik (CTF-GP)** [Hoer96] benutzt. Wichtige Einstellungen des GP-Laufes sind folgende:

Kontextfreie Grammatik

Die folgende kontextfreie Grammatik $\mathcal{G}_{\text{LFSR}}$ wird gewählt.

$$\begin{aligned} \mathcal{G}_{\text{LFSR}} &= (\mathcal{N}, \mathcal{T}, \mathcal{P}, S) \\ \mathcal{N} &= \{F, T\} \\ \mathcal{T} &= \{a_0, a_1, d_0, d_1, d_2, d_3, \mathbf{XOR}\} \end{aligned}$$

Dabei besteht die Relation der kontextfreien Grammatik $\mathcal{P}$ aus den folgenden Ableitungen:

$$\begin{array}{lcl} S & \longrightarrow & F \\ F & \longrightarrow & \mathbf{XOR}\ F\ F \ \mid \\ & & T \\ T & \longrightarrow & a_{n-1} \ \mid a_{n-2} \mid a_{n-3} \mid \ldots \mid a_{n-32} \end{array}$$

Die in der kontextfreien Grammatik $\mathcal{G}_{\text{LFSR}}$ definierten Funktion **XOR** wird die boolsche Operation **XOR** interpretiert.

Fitnessfunktion

Die Rohfitness wird durch die folgende Gleichung bestimmt:

$$f_r = L - d_H \tag{4.4}$$

wobei L die Länge des LSFR repräsentiert, und hier 32 gesetzt wird.

d_H repräsentiert die Summe der Differenz zwischen den tatsächlichen 32 Werten der Folge $a_{32}, a_{33}, a_{34}, ..., a_{63}$ und den 32 Werten der entsprechenden Folge, die durch die bewertete Funktion $f(a_{(n-1)}, ..., a_{(n-32)})$ produziert wird.

$$d_H = \sum_{n=32}^{63} \left(f(a_{(n-1)}, ..., a_{(n-32)}) \oplus a_n \right) \tag{4.5}$$

Darum besitzt die gesuchte Rückkopplungsfunktion die Rohfitness von $f_r = 32$.

Einstellung der GP-Parameter und Abbruchkriterium

Die wichtigsten Kontroll-Parameter des GP-Laufes sind in Tabelle 4.1 dargestellt.

GP-Parameter	Einstellung
Anzahl der Versuche	10
Anzahl der GP-Läufe pro Versuch $RUN_{\max}$	50
Populationsgröße M	1000
Max. Generation $G_{\max}$	50
Kreuzungswahrscheinlichkeit p_{c}	0,9 oder 0,0
Reproduktionswahrscheinlichkeit p_{r}	0,1 oder 1,0
Mutationswahrscheinlichkeit p_{m}	0,0 oder 0,9
Max. Ableitungstiefe der Ausgangspopulation $D_{\mathrm{div}_{\mathrm{initial}}}$	100
Max. Ableitungstiefe während des restl. Laufes $D_{\mathrm{div}_{\mathrm{created}}}$	100
Initialisierungsmethode	Zufällige Initialisierung
Selektion-Verfahren	Fitnessproportional
Elitismus	Ja

Tabelle 4.1: Einstellungen der GP Parameter

Der GP-Lauf wird nur gestoppt, wenn die Rückkopplungsfunktion gefunden ist, oder die Anzahl der maximalen Generationen $G_{\max}$ erreicht ist.

4.3.3 Problemeigenschaft

Angenommen, beinahe alle Teile der Rückkopplungsfunktion sind bereits gefunden, dann ergibt sich z.B. die folgende Funktion:

$$a_n = a_{n-25} \oplus a_{n-27} \oplus a_{n-29} \oplus a_{n-30}$$

Die fehlenden Teile werden noch gesucht. Abbildungen 4.2 stellt die *Fitness-landschaft* (Darstellung der Fitnesswerte in n Dimensionen, hier $n = 3$) von allen möglichen Programmen, d.h. mit den bekannten Teilen $a_{n-25} \oplus a_{n-27} \oplus a_{n-29} \oplus a_{n-30}$ und zwei möglichen Teilen aus der Grammatik $\mathcal{G}_{\mathrm{LFSR}}$, dar.

Obwohl in der bereits gefundenen Rückkopplungsfunktion nur noch $a_{n-(31)} \oplus a_{n-(32)}$ fehlen, besteht die Fitnesslandschaft in Abbildung 4.2 aus mehreren steilen Spitzen (hoher Fitnesswert), deren Umgebung meist die Fitness von ca. 16 besitzen. Ein Problem mit einer solchen Fitnesslandschaft ist für die GP schwierig zu lösen. Schlechte Individuen werden leicht irrtümlich als Eltern ausgewählt, um anschließend Nachkommen zu erzeugen.

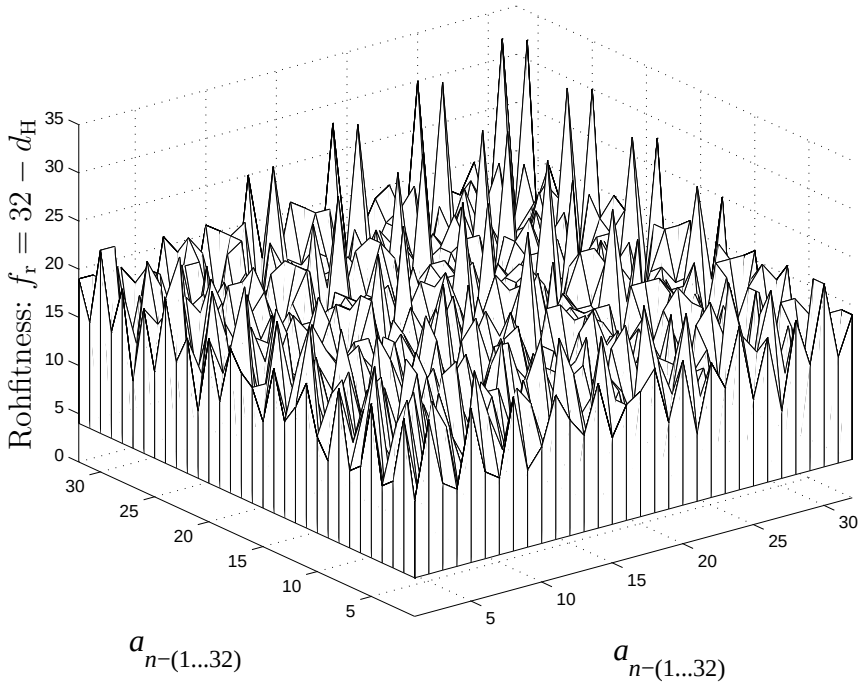


Abbildung 4.2: Fitnesslandschaft der Rückkopplungsfunktion: $a_{n-25} \oplus a_{n-27} \oplus a_{n-29} \oplus a_{n-30} \oplus a_{n-(1...32)} \oplus a_{n-(1...32)}$

Außerdem wird der Kreuzungsoperator keine besseren Nachkommen mehr produzieren. Durch die Kombination von kleineren *Fit-Building-Blocks* können keine größeren *Fit-Building-Blocks* in den Nachkommen entstehen. Darum kann die Evolution in der GP, die nur mit dem Kreuzungsoperator arbeitet, sehr schnell stagnieren.

4.4 Ergebnisse aus den Experimenten und Diskussion

Zur Darstellung und Diskussion der Ergebnisse aus den Experimenten ist es nötig, zuerst den akkumulierten Rechenaufwand R_E (*Computational Effort*) zu erläutern.

4.4.1 Rechenaufwand

Der akkumulierte Rechenaufwand R_E ist bereits in [Koza92] beschrieben und wird in den folgenden Schritten bestimmt:

- Zuerst wird $P(M, G)$ berechnet. $P(M, G)$ ist die kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs mit der Populationsgröße M und der maximalen Generation G, und wird durch die folgende Gleichung definiert:

$$P(M, G) = \frac{\text{Anzahl der erfolgreichen GP-Läufe}}{\text{Anzahl der GP-Läufe}} \tag{4.6}$$

wobei ein erfolgreicher GP-Lauf ein GP-Lauf ist, in dem das gewünschte Individuum (Lösung) gefunden wird. In dem Experiment besitzt das gewünschte Individuum eine maximale Rohfitness von 32.

- Zweitens wird die Erfolgswahrscheinlichkeit z definiert. z ist die Wahrscheinlichkeit, um mindestens in einem GP-Lauf mit der Populationsgröße M ein gewünschtes Individuum innerhalb der Generation G zu bekommen, wenn R GP-Läufe durchgeführt werden, und wird durch die folgende Gleichung definiert:
$$z = 1 - (1 - P(M, G))^R \tag{4.7}$$
In dem Experiment ist die Wahrscheinlichkeit $z = 99\%$.

- Danach wird $R(M, G, z)$ berechnet. $R(M, G, z)$ ist die Anzahl der benötigten unabhängigen GP-Läufe mit der Populationsgröße M und der maximalen Generation G, um mit der Wahrscheinlichkeit z mindestens einen erfolgreichen GP-Lauf zu bekommen, und wird durch die folgende Gleichung definiert:
$$\begin{aligned} R(M, G, z) &= \text{ceil}\left(\frac{\log(1-z)}{\log(1-P(M,G))}\right) \\ &= \text{ceil}\left(\frac{\log(1-0,99)}{\log(1-P(M,G))}\right) \end{aligned} \tag{4.8}$$

- Dann wird $I(M, G, z)$ berechnet. $I(M, G, z)$ ist die minimale Anzahl benötigter Individuen zum Erreichen einer Erfolgswahrscheinlichkeit von z, und wird durch die folgende Gleichung definiert:
$$I(M, G, z) = M \cdot R(M, G, z) \cdot (G + 1) \tag{4.9}$$
Die erste Generation, in der $I(M, G, z)$ minimal ist, wird als beste Generation G^* bezeichnet.

- Zum Schluss wird der akkumulierte Rechenaufwand R_{E} bestimmt. Der Rechenaufwand R_{E} ist folgendermaßen definiert:
$$R_{\text{E}} = I(M, G^*, z) = M \cdot R(M, G^*, z) \cdot (G^* + 1) \tag{4.10}$$

Der akkumulierte Rechenaufwand R_{E} ist eine Schätzgröße. Sie gibt an, wie viele Individuen (Programme) insgesamt mindestens durch die GP generiert werden müssen, um mit der definierten Wahrscheinlichkeit (hier 99%) eine Lösung zu finden, die das Erfolgskriterium erfüllt.

4.4.2 Ergebnisse und Diskussion

Für das 32-Bit-LFSR Problem gibt es insgesamt $2^{32} = 4.294.967.296$ mögliche Rückkopplungsfunktionen. Es ist sehr schwer, die Rückkopplungsfunktionen gemäss Gleichung (4.3) in einer solchen Suchraumgröße durch zufällige Suche zu finden. Außerdem versagt der Kreuzungsoperator. Nur durch den Mutationsoperator wird sich GP

verbessern, da der Mutationsoperator neue Richtungen in die Evolution einbringen kann.

Tabelle 4.2 stellt die Anzahl der erfolgreichen GP-Läufe nach der 50. Generation aus zehn Versuchen der Experimente mit unterschiedlichen Kreuzungs- und Mutationswahrscheinlichkeiten dar.

Einstellung des Experiments	Anzahl der erfolgreichen GP-Läufe nach der 50. Generation im $1., 2., 3., ..., 10.$ Versuch
$p_c = 0,9$ und $p_m = 0,0$	1, 0, 0, 2, 0, 0, 2, 0, 0, 0
$p_c = 0,0$ und $p_m = 0,9$	4, 2, 2, 3, 3, 1, 4, 3, 3, 2

Tabelle 4.2: Anzahl der erfolgreichen GP-Läufe nach der 50. Generation im $1., 2., 3., ..., 10.$ Versuch aus den LFSR-Experimenten, wobei ein Versuch aus 50 GP-Läufen besteht.

GP mit einer höheren Einstellung der Mutationswahrscheinlichkeit ($p_m = 0,9$) ist deutlich besser als GP mit einer Einstellung der Mutationswahrscheinlichkeit von Null ($p_m = 0,0$). Die durchschnittliche Anzahl erfolgreicher GP-Läufe aus zehn Versuchen jeweils mit 50 GP-Läufen pro Versuch beträgt im 1. Experiment ($p_c = 0,9$ und $p_m = 0,0$) $0,5$, im 2. Experiment ($p_c = 0,0$ und $p_m = 0,9$) jedoch $2,7$.

Mit diesen Ergebnissen wird zunächst für jeden Versuch die kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs $P(M,G)$ berechnet. Danach wird die durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs $\bar{P}(M,G)$ aus zehn Versuchen für das 1. und 2. Experiment ermittelt. Die durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs $\bar{P}(M,G)$ wird dann verwendet, um den akkumulierten Rechenaufwand R_E in den beiden Experimenten zu bestimmen.

Abbildung 4.3 stellt die durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs mit der Populationsgröße von 1000 bei den verschiedenen Generationen $\bar{P}(1000,G)$ und die minimale Anzahl benötigter Individuen zum Erreichen einer Erfolgswahrscheinlichkeit von 99% $I(1000,G,99\%)$ aus dem 1. Experiment dar. Die durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs $\bar{P}(1000,G)$ steigt in der 4., 5., 13., 14. und 15. Generation an. Da wurde bei dem 1. Experiment nur in diesen Generationen die Lösung gefunden. Ab der 15. Generation stagniert $\bar{P}(1000,G)$, die maximale $\bar{P}(1000,G)$ beträgt ca. $1,11\%$.

$I(1000,G,99\%)$ kann erst in der 4. Generation berechnet werden. Nur in den Generationen, in denen die Lösung gefunden wird, wird die Anzahl der benötigten unabhängigen GP-Läufe $R(1000,G,99\%)$ reduziert. Dadurch sinkt $I(1000,G,99\%)$ in der 5., 13., 14. und 15. Generation. In anderen Fällen wird $I(1000,G,99\%)$ in Gleichung (4.9) proportional zur Generation ansteigen. Bei den Experimenten wurde ab der 16. Generation keine Lösung gefunden, folglich steigt $I(1000,G,99\%)$ proportional bis zur letzten Generation. Das Minimum von $I(1000,G,99\%)$ liegt in der

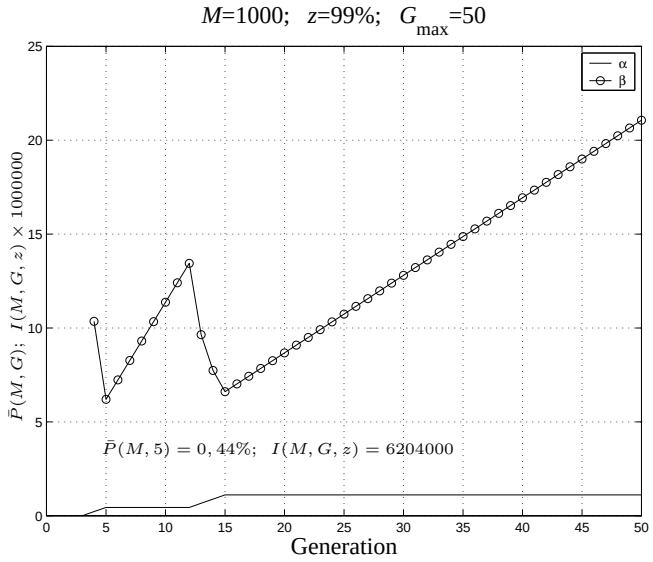


Abbildung 4.3: Ergebnisse aus dem 1. Experiment ($p_c = 0,9$ und $p_m = 0,0$) α: durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs mit der Populationsgröße von 1000 bei den verschiedenen Generationen $\bar{P}(1000, G)$ β: minimale Anzahl benötigter Individuen zum Erreichen einer Erfolgswahrscheinlichkeit von 99% $I(1000, G, 99\%)$

5. Generation. Daraus folgt, dass bei dem 1. Experiment der akkumulierte Rechenaufwand $R_{E_1} = I(1000, 5, 99\%) = 6.204.000$ beträgt. Das Verhältnis zwischen dem akkumulierten Rechenaufwand und der Suchraumgröße beträgt ca. $1,44 \cdot 10^{-3}$.

Abbildung 4.4 stellt die durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs mit der Populationsgröße von 1000 bei den verschiedenen Generationen $\bar{P}(1000, G)$ und die minimale Anzahl benötigter Individuen zum Erreichen einer Erfolgswahrscheinlichkeit von 99% $I(1000, G, 99\%)$ aus dem 2. Experiment dar. In dem 2. Experiment wurde die Lösung in verschiedenen Generationen (5. bis 49. Generation) gefunden. Dadurch steigt die durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs $\bar{P}(1000, G)$ in mehreren Generationen G an, und die maximale $\bar{P}(1000, G)$ beträgt ca. $5,4\%$.

$I(1000, G, 99\%)$ kann erst in der 5. Generation berechnet werden. Nur in der Generation, in der die Lösung gefunden wird, wird $I(1000, G, 99\%)$ sinken. Das Minimum von $I(1000, G, 99\%)$ liegt bei der 14. Generation. Daraus folgt, dass bei dem 2. Experiment der akkumulierte Rechenaufwand $R_{E_2} = I(1000, 14, 99\%) = 2.850.000$ beträgt. Das Verhältnis zwischen dem akkumulierten Rechenaufwand und der Suchraumgröße beträgt ca. $0,66 \cdot 10^{-3}$.

Falls das Minimum von $I(1000, G, 99\%)$ in beiden Experimenten bei der 0. Generation (Initialisierungsphase) läge, würde es bedeuten, dass die gesuchte Rückkopplungsfunktion gemäss Gleichung (4.3) durch zufällige Suche gefunden wäre. Die Ergebnisse

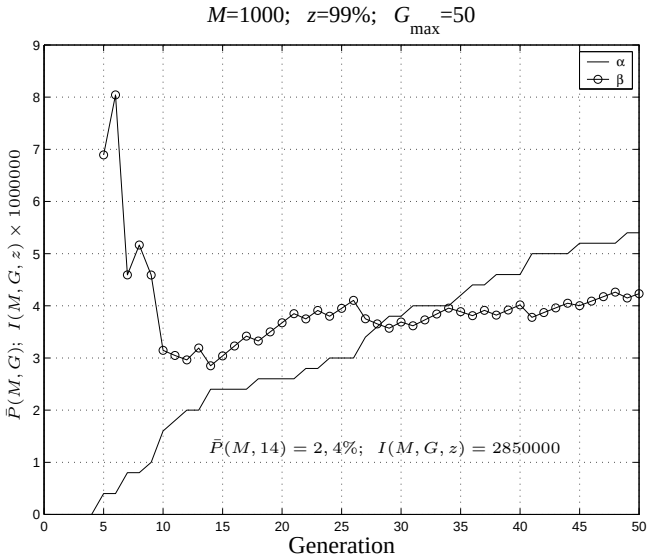


Abbildung 4.4: Ergebnisse aus dem 2. Experiment ($p_c = 0,0$ und $p_m = 0,9$) α: durchschnittliche kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufs mit der Populationsgröße von 1000 bei den verschiedenen Generationen $\bar{P}(1000, G)$ β: minimale Anzahl benötigter Individuen zum Erreichen einer Erfolgswahrscheinlichkeit von 99% $I(1000, G, 99\%)$

zeigen, dass der Mutationsoperator bei der Aufgabe mit diskontinuierlicher Fitnesslandschaft besser als der Kreuzungsoperator funktioniert. Der Mutationsoperator bringt neue Richtungen in die Population ein, er ist außerdem nicht identisch mit zufälliger Suche. Durch den Mutationsoperator soll GP von einer Spitze zur anderen Spitze in der diskontinuierlichen Fitnesslandschaft solange springen, bis GP die richtige Lösung findet. Der Vorteil von GP gegenüber der zufälligen Suche in beiden Experimenten ist aus dem Verhältnis zwischen dem akkumulierten Rechenaufwand und der Suchraumgröße ersichtlich.

4.4.3 GP in der Kryptoanalyse

Die Kryptoanalyse ist die Wissenschaft von der Wiederherstellung des Klartextes einer verschlüsselten Nachricht ohne Zugriff auf den Schlüssel. Die Kryptoanalyse ist erfolgreich, wenn der Klartext oder der Schlüssel ermittelt wird. Sie kann auch Schwachstellen in einem Kyptosystem aufspüren, die schließlich zu den eben genannten Ergebnissen führen.

Das Problem mit LFSR ist ein einfaches Beispiel der Kryptoanalyse. LFSR ist ein Bestandteil in der Kryptologie, indem LFSR als Generator für Pseudozufallsfolgen verwendet wird. Die meisten praktischen Entwürfe für Stromchiffrierungen basieren auf LFSRs. In Abbildung 4.5 ist der prinzipielle Aufbau einer Stromchiffrierung gezeigt. Der Schlüssel bildet den Anfangszustand der LFSRs.

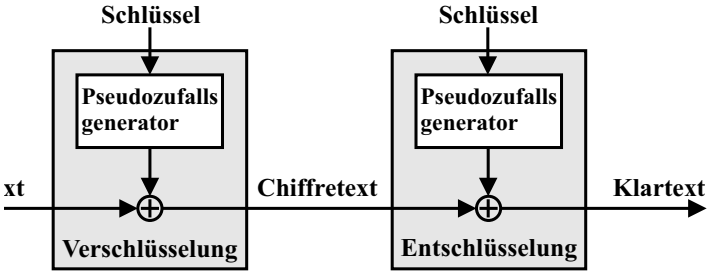


Abbildung 4.5: Aufbau einer Stromchiffrierung

Die Rückkopplungsfunktion des L-Bit-LFSRs kann leicht durch den Berlekamp-Massey-Algorithmus [Schn96] mit nur $2L$ Ausgabebits berechnet werden. Hat man erst einmal das LFSR generiert, ist die Stromchiffrierung geknackt.

Im Bereich der Kryptographie beschäftigt man sich mit der scheinbar zufälligen Welt, damit der kryptographische Algorithmus nicht sehr leicht gebrochen wird. Dadurch besitzt der gute kryptographische Algorithmus immer eine diskontinuierliche Fitnesslandschaft. Außerdem ist der Suchraum der kryptographischen Algorithmen extrem groß. Obwohl GP in den beiden Experimenten die Rückkopplungsfunktion des LFSRs besser als die zufällige Suche finden kann, ist die Anzahl der erfolgreichen GP-Läufe sehr niedrig.

Daraus können wir sehen, dass GP für die Kryptoanalyse nicht besonders gut geeignet ist, in erster Linie wegen der Beschaffenheit der Lösunglandschaft. GP funktioniert am besten bei Problemen, die ein Kontinuum von Lösungen besitzen, wobei einige besser sind als andere. Dies ermöglicht GP zu lernen, indem GP immer bessere Lösungen entwickelt. Das Knacken eines Algorithmus der Kryptographie bietet eigentlich nichts, was mit Lernmöglichkeiten vergleichbar wäre. Entweder hat man den Schlüssel (auch Klartext) entdeckt oder man hat ihn nicht entdeckt. (Dies trifft zumindest auf gute Algorithmen zu.) GP eignet sich gut in strukturierten Umgebungen, in denen es etwas zu lernen gibt. In der scheinbar zufälligen Welt der Kryptographie, die eine hohe Entropie aufweist, ist GP eher fehl am Platze.

4.5 Zusammenfassung

In der GP mit standardisierten Parametereinstellungen ist der Kreuzungsoperator gewöhnlich der vorherrschende Operator beim Erzeugen von Nachkommen. Die bedeutende Rolle des Kreuzungsoperators bei der Erzeugung einer Lösung eines Problems wird durch die *Building Block Hypothese* unterstützt. Der Mutationsoperator wird dagegen mit einer sehr geringen Wahrscheinlichkeit eingesetzt.

Das Problem der Rückkopplungsfunktion eines LFSRs ist ein einfaches Beispiel der Kryptoanalyse. Eine wichtige Eigenschaft der Kryptoanalyse ist, dass sie kein Kontinuum von Lösungen besitzt. Ihre Fitnesslandschaft ist stark diskontinuierlich und

besteht aus mehreren steilen Spitzen. GP kann sich in einer solchen Fitnesslandschaft nicht gut orientieren oder lernen. Schlechte Individuen können irrtümlich leicht als Eltern ausgewählt werden, um Nachkommen zu erzeugen. Außerdem wird der Kreuzungsoperator keine besseren Nachkommen mehr produzieren. Dadurch stagniert die Evolution in der GP, die nur mit Kreuzungsoperationen sehr schnell arbeitet. Nur der Mutationsoperator kann neue Aspekte oder neue Richtungen in der Population einbringen. Dennoch ist die Anzahl der erfolgreichen GP-Läufe sehr niedrig. Auf Grund der geringen Lernmöglichkeiten ist das Problem der Kryptoanalyse für GP nicht geeignet.

5 Genetische Programmierung mit Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem

Dieses Kapitel beschäftigt sich mit der Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem, die eine Variante der globalen Parallelisierung ist. Die Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem beruht auf einer von Siewert[1] betreuten Diplomarbeit ([For00]). Mit dieser Parallelisierung kann der Zeitaufwand der GP reduziert werden.

Im ersten Abschnitt dieses Kapitels beginnen wir mit dem Konzept der globalen Parallelisierung. Im zweiten Abschnitt wird der Aufbau der Parallelisierung der Fitnessberechnung auf verteilte Rechnersysteme erläutert. Der dritte Abschnitt stellt die Experimentergebnisse mit der Parallelisierung dar. Zum Schluss werden Vorteile und Probleme der Arbeit zusammengefasst. Für weiterreichende Darstellungen der Theorie der Parallelisierung bei GP wird auf die Arbeit von M. Tomassini in [Tom99] verwiesen. Weitere Verfahren der Parallelisierung bei GP findet man in [Ous96, Ous97, Wal99, Dev01, Tane01].

5.1 Konzept der globalen Parallelisierung

Wie schon in Kapitel 2 erwähnt, lässt sich die GP leicht parallelisieren. Bei der globalen Parallelisierung werden

- die Fitnessberechnung sowie
- teilweise auch die Ausführung genetischer Operatoren

parallelisiert, ohne inhaltliche Änderungen am Verfahrensablauf vorzunehmen. Es existiert nur eine Population und jedes Individuum der Population kann mit jedem anderen gekreuzt werden.

[1] http://speedy.et.unibw-muenchen.de/persons/gedenk.html#siewert

Die genetischen Operatoren, wie z.B. Mutation und Kreuzung, sind möglich zu parallelisieren. Die Mutation der Individuen erfolgt unabhängig voneinander und kann daher parallel durchgeführt werden. Bei der Kreuzung sind jeweils Paare von Individuen unabhängig voneinander zu betrachten.

Die Fitnessermittlung eines Individuums ist im Normalfall unabhängig von anderen Mitgliedern der Population, daher ist sie leicht zu parallelisieren.

Die globale Parallelisierung zur Fitnessberechnung ist für GP besonders nützlich, da sie den beträchtlichen Zeitaufwand der GP reduzieren kann. Die globale Parallelisierung der Fitnessberechnung kann in verschiedenen Implementierungen durchgeführt werden:

1. Auf einem Multiprozessor-Rechner mit gemeinsamem Speicher kann die Population im gemeinsamen Speicher abgelegt sein. Jeder Prozessor liest die ihm zugewiesenen Individuen und schreibt das Ergebnis der Fitnessbewertung zurück in den gemeinsamen Speicher. Eine Synchronisation ist hier nur zwischen den Generationszyklen nötig.

2. Bei Parallelrechnern mit verteiltem Speicher wird das *Master-Client-Konzept* verwendet. Dies lässt sich folgendermaßen erklären.

   - Die Population wird auf einem *Master*-Prozessor gespeichert.
   - Der Master-Prozessor verteilt die Individuen zur Fitnessberechnung an die *Client*-Prozessoren.
   - Nach der Fitnessberechnung werden die Ergebnisse dem Master-Prozessor zurückgeliefert.
   - Der Master-Prozessor sammelt die Ergebnisse und wendet die genetischen Operatoren zur Erzeugung der nächsten Generation an.

   Hierbei kann allerdings der Master-Prozessor zum Engpass werden. Abbildung 5.1 verdeutlicht das Master-Client-Konzept nochmals im Überblick. Ein Beispiel dafür ist die Arbeit von M. Oussaidène in [Ous96, Ous97].

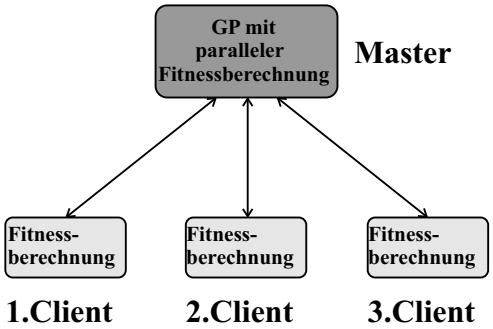


Abbildung 5.1: Einfache globale Parallelisierung zur Fitnessberechnung mit dem *Master-Client-Konzept*

3. In Computernetzwerken ist die Durchführung der globalen Parallelisierung ähnlich wie bei Parallelrechnern mit verteiltem Speicher. Die Population ist auf einem *Master*-Rechner gespeichert, der die Individuen zur Fitnessbewertung an die übrigen *Client*-Rechner verteilt, die Ergebnisse einsammelt und dann die genetischen Operatoren zur Erzeugung der nächsten Generation anwendet. Ein Beispiel dafür ist die Arbeit von C. Forster in [For00].

5.2 Programmpaket zur Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem

C. Forster entwickelte in seiner Diplomarbeit [For00] ein Programmpaket zur Parallelisierung der Fitnessberechnung auf verteilte Rechner. Die Implementierung der Parallelisierung der Fitnessberechnung in einem Computernetzwerk hat folgende Vorteile:

- Es gibt bereits Standard-Software Umgebungen für Computernetzwerke, die die Entwicklung des Programms leichter machen.
- Diese Implementierung ist in jedem normalen PC einsetzbar. Alle im Netz verbundenen Rechner, d.h. sowohl Rechner im lokalen Netzwerk als auch Rechner im Internet, können bei der Parallelisierung der Fitnessberechnung genutzt werden.

Das gewünschte Ziel des Programmpakets ist es, den Zeitaufwand bei der Fitnessberechnung mit n Clients um den Faktor n zu reduzieren.

Um die Ergebnisse der Implementierung deutlich darzustellen, wird ein relativer Geschwindigkeitsfaktor $G_{1:n_c}$ gemessen. Der relative Geschwindigkeitsfaktor $G_{1:n_c}$ ist hier folgendermaßen definiert.

$$G_{1:n_c} = \frac{T_{\text{Single}}}{T_{n_c\text{Clients}}} \tag{5.1}$$

Wobei:

- T_{Single} der benötigte Zeitverbrauch zur Durchführung einer gegebenen Aufgabe mit einem Rechner, und
- $T_{n_c\text{Clients}}$ der benötigte Zeitverbrauch zur Durchführung derselben Aufgabe bei der Parallelisierung mit n Client-Rechnern ist.

Der gewünschte relative Geschwindigkeitsfaktor $G_{1:n_c}$ soll daher bei der Implementierung mit n-Clients n entsprechen. Jedoch ist es normalerweise nicht möglich, eine lineare Steigung des relativen Geschwindigkeitsfaktores $G_{1:n_c}$ beim Erhöhen der Anzahl der Client-Rechner n_c zu erreichen. Weil Zeit für die Kommunikation zwischen dem Master-Rechner und den Cleint-Rechnern, und Zeit für den Transport der zu bearbeiten Pakete und Ergebnisse benötigt wird.

Zum besseren Verständnis der Implementierung werden kurz wichtige Begriffe erläutert.

TCP/IP

Das am häufigsten verwendete Protokoll in lokalen, wie in globalen Netzen ist die TCP/IP Protokollfamilie ([Her99]). TCP steht für *Transmission Control Protocol*, IP für *Internet Protocol*.

TCP/IP liegt derzeit in der Version 4 vor (IPv4). Das vor 20 Jahren entwickelte IPv4 verwendet ein 32-Bit-Adresssystem, das theoretisch an die vier Milliarden IP-Adressen ermöglicht. In der Praxis ist aber ein großer Teil dieser Adressen durch Gruppenbildung und andere Mechanismen nicht nutzbar.

Unix-Domain-Socket

Unix-Domain-Socket ([Her99]) sind keine Netzwerkprotokolle und können nur für Sockets auf einem lokalen Rechner verwendet werden. Die Adressen sind hierbei Dateien (Pfadnamen), die im Filesystem angelegt werden, wenn ein Socket an eine Datei gebunden wird.

Capabilities

Als Capabilities werden Codepakete, die die Clients um neue Fähigkeiten erweitern, bezeichnet. Damit wird die Anwendung variabel gehalten.

Pakete

Drei Pakettypen werden unterschieden.

- Datenpakete beinhalten die vom Client zu bearbeitenden Daten.
- Ergebnispakete enthalten die Ergebnisse der Berechnungen.
- Updatepakete enthalten alle notwendigen Informationen, um einem Client eine neue Capability hinzuzufügen.

Single-Verfahren und Multi-Verfahren

Das *Single-Verfahren* ist in dieser Arbeit als die GP, die in einem Rechner durchgeführt wird, definiert.

Im Gegensatz dazu bedeutet das *Multi-Verfahren* die Durchführung einer GP mit dem Programmpaket zur Parallelisierung der Fitnessberechnung auf die im Computernetzwerk verteilten Rechner ([For00]).

5.2.1 Aufbau der Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem

Das Programmpaket zur Parallelisierung der Fitnessberechnung auf verteilte Rechner besteht aus zwei Teilen, nämlich

1. Anwendung
2. Cluster.

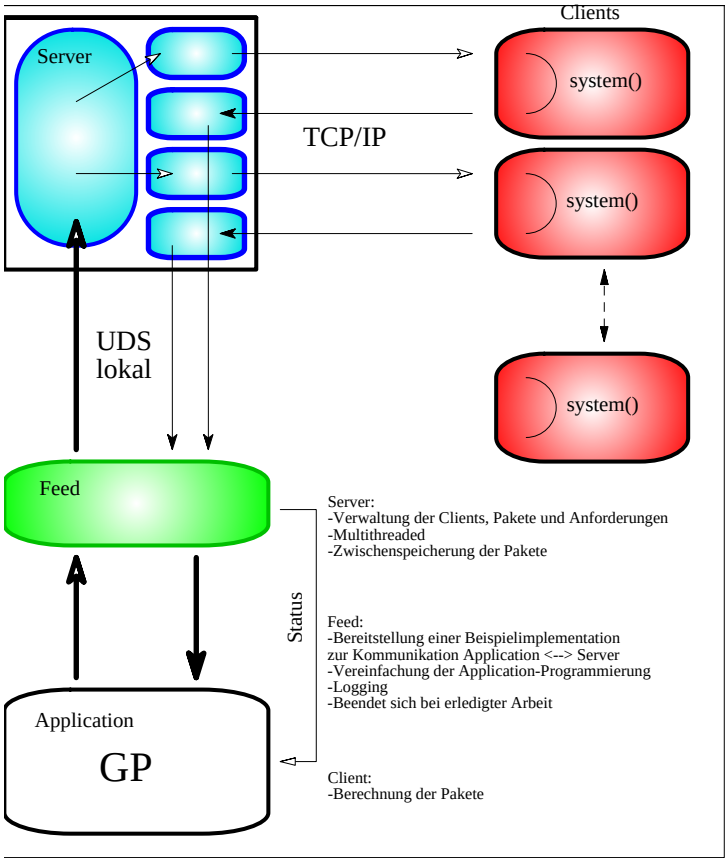


Abbildung 5.2: Funktionsweise des GP-Clusters

Anwendung

Die Anwendung (*Application*) ist ein normales GP-Programm, das vorher umprogrammiert werden muss. In diesem GP-Programm findet keine Fitnessberechnung mehr statt, stattdessen muss das GP-Programm

- Individuen in Datenpakete umwandeln und diese verteilen,
- Fitnesswerte aus dem Ergebnispaket lesen und den entspechenden Individuen zuordnen.

Die Anzahl der Pakete n_{p} wird in der Anwendung eingestellt. Die Anwendung befindet sich im Master-Rechner.

Cluster

Ein Cluster besteht aus drei wichtigen Teilen.

1. **Server**: Der Server ist die zentrale Steuerungseinrichtung des Systems und befindet sich im Master-Rechner. Er verwaltet den Feed sowie die diversen Clients und koordiniert die einzelnen zu bearbeitenden Pakete.

   Die dabei anfallende Notwendigkeit zur Verwaltung von Paketen, Versionsnummern und Daten geschieht über Verzeichnisse und Dateinamen, die nach einem ganz bestimmten Muster vergeben werden.

2. **Clients**: Die Clients machen nichts anderes als erhaltene Datenpakete zu bearbeiten und die Ergebnisse zurückzuschicken. Diese Bearbeitung geschieht über einen einfachen Aufruf des Befehls **system()** ([Her99]), der ein Programm startet, welches die Fitnessberechnung übernimmt. Diese Programme können ebenfalls ähnlich wie Pakete über das Netz geladen und installiert werden.

   Die Verbindung zwischen Clients und Server geschieht über TCP/IP.

3. **Feed**: Der Feed befindet sich im Master-Rechner. Er stellt die Schnittstelle zwischen Anwendung und Server dar. Ihm wird ein Verzeichnis mit den zu bearbeitenden Paketen und eine vorbereitete Datei übergeben. Diese vorbereitete Datei enthält gepackt das für die Clients zum Bearbeiten notwendige Programm samt Installationsroutine. Der Feed befindet sich im Master-Rechner.

   Die Verbindung zwischen Feed und Server geschieht, im Gegensatz zu der von Client und Server, über einen Unix Domain Socket (UDS), der nur lokal, also nicht über ein Netzwerk, angesprochen werden kann.

Abbildung 5.2 veranschaulicht nochmals den Aufbau der Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem.

5.2.2 Ablauf der Parallelisierung der Fitnessberechnung auf einem verteilten Rechnersystem

In Abbildung 5.3 wird der Ablauf der Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem zusammengefasst. Der Ablauf lässt sich in folgenden Schritten erläutern:

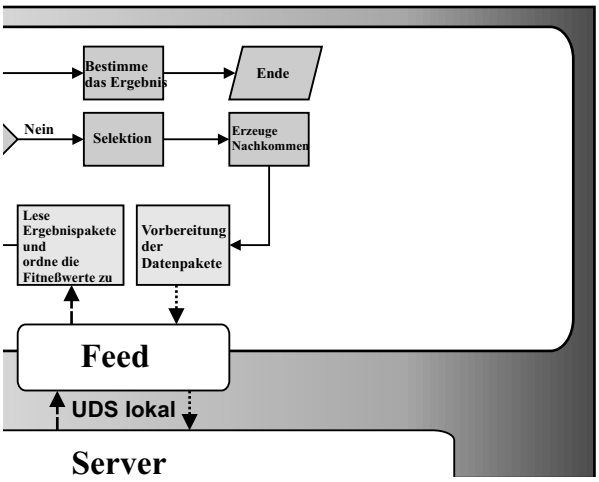


Abbildung 5.3: Einfacher Ablauf des GP-Clusters mit n-Clients

Vorbereitungsschritt

Bevor eine Parallelisierung der Fitnessberechnung auf verteilte Rechnersysteme gestartet werden kann, sind folgende Arbeiten im Master-Rechner notwendig:

- Vorbereiten des GP-Laufes (wie gewöhnlich)
- Festlegen der Paketanzahl n_{p}, und
- Erstellen der benötigten Capabilities

Initialisierung des Clusters

Als erstes wird der Server im Master-Rechner gestartet. Dieser wartet dann auf Verbindungen. Dann folgen die Clients. Diese lesen ihre Konfiguration ein, um die

IP-Adresse des Servers zu erhalten. Dann melden sie sich dort an, übertragen ihre Capabilities und erhalten eine Client-Number (oder ID-Number), die der Unterscheidung der einzelnen Clients dient.

Start von GP und Erzeugen von Datenpaketen

Das GP-Programm arbeitet wie gewöhnlich. Ab der 1. Generation werden die neu erzeugten Nachkommmen aber in Datenpakete, deren Anzahl vor dem Ablauf der GP festgelegt wird, aufgeteilt. GP wartet dann auf Ergebnispakete.

Start des Feed und Aufbau der Feed-Server-Verbindung

Nun wird der Feed gestartet. Die Verbindung zwischen Feed und Server wird über einen Unix Domain Socket (UDS) aufgebaut. Ein normales IP-packet wird an den Server geschickt. Der Inhalt ist der Dateiname des UDS. Daraufhin öffnet der Server diesen Socket und meldet sich als UDS-Client an den Feed an. Der Feed antwortet mit der Übertragung der Capability. Dann folgen die einzelnen Datenpakete. Ist dies beendet, wartet der Feed auf Antwort vom Server.

Verteilen von Datenpaketen durch den Server

Der Server sortiert die Datenpakete, deren Name den UDS Dateinamen enthalten. Besitzt der Server vakante Datenpakete und der Client die passenden Capabilities, werden diese sofort danach übertragen und die Clients beginnen mit der Arbeit.

Bearbeiten der Datenpakete in den Clients

Der Client liest die Datenpakete und berechnet dann Fitnesswerte mit den zuvor vom Server übertragenen Capabilities. Nach der Fitnessberechnung werden die berechneten Fitnesswerte in ein Ergebnispaket geschrieben und zum Server zurückgeschickt.

Ergebnispaket zurück an Server

Kommen die Ergebnispakete beim Server an, analysiert der Server die einzelnen Pakete und öffnet daraufhin jeweils eine Verbindung zu dem im Namen angegebenen UDS, überträgt das Ergebnispaket und schließt die Verbindung wieder.

Ergebnispaket zurück an Feed

Der Feed hingegen speichert das Erhaltene und inkremeniert einen Zähler, bis alle Ergebnispakete wieder angekommen sind. Damit endet der Feed. Wenn aber nach einer bestimmten Wartezeit nicht alle Ergebnispakete angekommen sind, wird der Server die entspechenden fehlenden Datenpakete nochmals zu einem Client schicken.

Lesen von Fitnesswerten und weitere Arbeit in GP

Falls das Ergebnispaket ankommt, werden die Fitnesswerte aus dem angekommenen Ergebnispaket ausgelesen und den ursprünglichen Individuen zugeordnet. Die GP muss warten, bis sie alle Ergebnispakete bekommt, erst dann kann sie wie gewöhnlich weiterarbeiten.

5.2.3 Mathematische Analyse

In diesem Abschnitt wird die mathematische Berechnung zur Analyse des relativen Geschwindigkeitsfaktors $G_{1:n_c}$ durchgeführt. Wir nehmen zuerst an, dass alle Client-Rechner die gleiche Rechnerleistung besitzen. Außerdem ist es nötig, einige Begriffe zu erläutern:

Im Single-Verfahren

- T_S ist der Gesamtzeitaufwand zur Fitnessberechnung im Single-Verfahren.

Im Multi-Verfahren

- T_P ist der Gesamtzeitaufwand der Fitnessberechnung eines Paketes.
- T_H ist die Vorlaufzeit, die ein Paket benötigt (Header).
- T_M ist der Gesamtzeitaufwand der Fitnessberechnung im Multi-Verfahren.
- n_p ist die Anzahl der Pakete.
- n_c ist die Anzahl der Client-Rechner.

Die Vorlaufzeit eines Pakets T_H ist die benötigte Zeit für verschiedene Prozesse innerhalb des Clusters, wie z.B.

1. Bearbeiten von Daten- und Ergebnispaketen in der Anwendung

2. Kommunikation zwischen Server und Feed oder Server und Client
3. Bearbeiten von Daten- und Ergebnispaketen im Client
4. Übertragung des Pakets im Netzwerk,

damit Fitnessberechnungen in Client-Rechnern stattfinden können. Die Vorlaufzeit eines Pakets T_{H} ist daher abhängig von dem Protokoll des Clusters, der Leistung des Rechners und der Geschwindigkeit des Netzes.

Die Lastverteilung in Datenpaketen ist ein wichtiger Faktor für den relativen Geschwindigkeitsfaktor $G_{1:n_{\mathrm{c}}}$. Bei ideal guter Lastverteilung wird T_{P} bei allen Clients gleich lang sein, T_{P} beträgt daher:

$$T_{\mathrm{P}} = \frac{T_{\mathrm{S}}}{n_{\mathrm{p}}} \tag{5.2}$$

Bei schlechter Lastverteilung sind die Paketberechnungszeiten T_{P} deutlich unterschiedlich, und die längsten T_{P} werden den relativen Geschwindigkeitsfaktor $G_{1:n_{\mathrm{c}}}$ stark beeinflussen. Bei der GP mit hohen Populationsgrößen verteilen sich die Individuen mit unterschiedlicher Größe gut in der Population. Bis zu einer bestimmten Anzahl von Paketen kann die Last in jedem Datenpaket noch ideal gleich verteilt sein.

Für den Gesamtzeitaufwand der Fitnessberechnung im Multi-Verfahren T_{M} ergibt sich folgendes:

$$T_{\mathrm{M}} = (\frac{T_{\mathrm{S}}}{n_{\mathrm{p}}} + T_{\mathrm{H}}) \cdot \frac{n_{\mathrm{p}}}{n_{\mathrm{c}}} \quad ;\text{mit} \quad n_{\mathrm{p}} \geq n_{\mathrm{c}} \tag{5.3}$$

Als Funktion für den relativen Geschwindigkeitsfaktor $G_{1:n_{\mathrm{c}}}$ aus Gleichung (5.1) ergibt sich dann:

$$G_{1:n_{\mathrm{c}}} = \frac{T_{\mathrm{Single}}}{T_{n_{\mathrm{c}}\mathrm{Clients}}} = \frac{T_{\mathrm{S}}}{(\frac{T_{\mathrm{S}}}{n_{\mathrm{p}}} + T_{\mathrm{H}}) \cdot \frac{n_{\mathrm{p}}}{n_{\mathrm{c}}}} \quad ;\text{mit} \quad n_{\mathrm{p}} \geq n_{\mathrm{c}}$$

Daraus folgt:

$$G_{1:n_{\mathrm{c}}} = n_{\mathrm{c}} \cdot \frac{T_{\mathrm{S}}}{T_{\mathrm{S}} + n_{\mathrm{p}} \cdot T_{\mathrm{H}}} \quad ;\text{mit} \quad n_{\mathrm{p}} \geq n_{\mathrm{c}} \tag{5.4}$$

Aus Gleichung (5.4) können wir feststellen, dass ein Cluster den gewünschten relativen Geschwindigkeitsfaktor, d.h. $G_{1:n_{\mathrm{c}}} = n_{\mathrm{c}}$, erreicht, wenn

1. die Lastverteilung in den Datenpaketen gleichmäßig ist (siehe Gleichung (5.2)), und
2. die Last der GP sehr groß ist, so dass der Zeitaufwand der Fitnessberechnung im Single-Verfahren T_{S} sehr viel größer als die Vorlaufzeit eines Pakets T_{H} ist.

Übrigens soll die optimale Anzahl der Pakete der Anzahl der Client-Rechner entsprechen, d.h. $n_\mathrm{p} = n_\mathrm{c}$.

Ein Cluster mit geringer Anzahl von Clients und GP-Läufen mit hoher Last bringt deutliche Vorteile. Je größer die Anzahl der Clients n_c ist, desto langsamer steigt der relative Geschwindigkeitsfaktor $G_{1:n_\mathrm{c}}$.

Im Fall von unterschiedlicher Rechnerleistung der Clients ist es erforderlich, die Anzahl der Pakete neu zu optimieren. Clients mit hoher Rechnerleistung können mehr Datenpakete als Clients mit niedriger Rechnerleistung bekommen. Der gewünschte relative Geschwindigkeitsfaktor $G_{1:n_\mathrm{c}}$ ist daher noch schwieriger zu erreichen.

Bei sehr kleinem Zeitaufwand der Fitnessberechnung im Single-Verfahren T_S hat jedoch das Multi-Verfahren keinen Vorteil gegenüber dem Single-Verfahren. Der relative Geschwindigkeitsfaktor $G_{1:n_\mathrm{c}}$ wird unter eins liegen.

5.3 Experiment mit Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem

In diesem Abschnitt wird das Ergebnis aus dem Experiment mit Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem dargestellt. Das Ziel des Experiments ist die Messung des relativen Geschwindigkeitsfaktors $G_{1:n_\mathrm{c}}$ unter optimalen Bedingungen. Die optimalen Bedingungen bedeuten hier:

- Das Cluster befindet sich in einem geschlossenen Netzwerk. Kein weiterer Benutzer kann auf die Rechner zugreifen.
- Alle Client-Rechner haben die gleiche Hardware und besitzen daher die gleiche Rechnerleistung .
- Außerdem entspricht die Anzahl der Pakete n_p der Anzahl der Client-Rechner n_c, d.h. $n_\mathrm{p} = n_\mathrm{c}$.

Im Experiment wird die Anzahl der Client-Rechner n_c von 2, 3, 4, 6 und 8 eingestellt.

5.3.1 Vorbereitung des Experiments

Für das Multi-Verfahren wurden mehrere Rechner (AMD-Prozessor 700 MHz Arbeitspeicher von 256 MByte) zu einem Cluster vereinigt. Jeder Rechner arbeitet mit dem Betriebsystem SuSE Linux 7.0. Die Rechner sind über einen Switch als

sternförmiges Netzwerk verbunden. Das Single-Verfahren fand in einem identischen Rechner statt.

Für den beiden Verfahren (das Single- und Multi-Verfahren mit $n_c = 2, 3, 4, 6$ und 8) werden fünf GP-Läufe durchgeführt. Die Aufgabe der Anwendung (GP) ist die Suche nach einer Prädiktionsfunktion der jährlichen durchschnittlichen Sonnenfleckenrelativzahl [2].

Die genetische Programmierung mit kontextfreier Grammatik (CTF-GP) [Hoer96] wurde dabei verwendet, deren Einstellung in Tabelle 5.1 dargestellt ist.

GP-Parameter	Einstellung
Max. Anzahl der GP-Läufe $RUN_{\max}$	5
Populationsgröße M	1000 und 2000
Max. Generation $G_{\max}$	50
Kreuzungswahrscheinlichkeit p_c	0,9
Reproduktionswahrscheinlichkeit p_r	0,1
Mutationswahrscheinlichkeit p_m	0,2
Max. Ableitungstiefe der Ausgangspopulation $D_{\mathrm{div}_{\mathrm{initial}}}$	30
Max. Ableitungstiefe während des restl. Laufes $D_{\mathrm{div}_{\mathrm{created}}}$	200
Initialisierungsmethode	Zufällige Initialisierung
Selektionsverfahren	Fitnessproportional
Elitismus	Ja

Tabelle 5.1: Einstellungen der GP-Parameter

Die Rohfitness ist die Summe des Quadrats des Prädiktionsfehlers für die Jahre von 1750 bis 1920, die von Gleichung (5.5) deutlich dargestellt wird:

$$f_r = \sum_{t=1750}^{1920} (x_t - \hat{x}_t)^2 \tag{5.5}$$

Dabei ist x_t die tatsächliche jährliche durchschnittliche Sonnenfleckenrelativzahl im Jahr t, und $\hat{x}_t$ die prädiktierte jährliche durchschnittliche Sonnenfleckenrelativzahl im Jahr t.

Die folgende kontextfreie Grammatik $\mathcal{G}_s$ wurde verwendet.

$$\begin{aligned} \mathcal{G}_s &= (\mathcal{N}, \mathcal{T}, \mathcal{P}, S) \\ \mathcal{N} &= \{F_0, F_1, F_t\} \\ \mathcal{T} &= \{a_0, a_1, d_0, d_1, d_2, d_3, \mathbf{NOT}, \mathbf{OR}, \mathbf{AND}, \mathbf{IF}\} \end{aligned}$$

$\mathcal{P}$ besteht aus den folgenden Ableitungen:

[2] Siehe Abschnitt 6.5.3

$$
\begin{array}{lcl}
S & \longrightarrow & F_1 \\
F_1 & \longrightarrow & F_0 \mid \\
 & & \quad \textbf{Add} \quad F_1 \quad F_1 \mid \\
 & & \quad \textbf{Sub} \quad F_1 \quad F_1 \mid \\
 & & \quad \textbf{Mul} \quad F_1 \quad F_1 \mid \\
 & & \quad \textbf{Div} \quad F_1 \quad F_1 \mid \\
 & & \quad \textbf{sin} \quad F_1 \mid \\
 & & \quad \textbf{cos} \quad F_1 \mid \\
 & & \quad \textbf{exp} \quad F_1 \mid \\
 & & \quad \textbf{log} \quad F_1 \\
F_0 & \longrightarrow & F_t \mid x_{t-1} \mid x_{t-2} \mid x_{t-4} \mid x_{t-8} \\
F_t & \longrightarrow & 10.0 \mid 9.5 \mid ... \mid -9.5 \mid -10.0
\end{array}
$$

Die in der kontextfreien Grammatik $\mathcal{G}_\mathrm{s}$ definierten Funktionen, z.B. **Add**, **Sub**, usw. sind als mathematische Funktionen zu interpretieren:

$$
\begin{array}{rcl}
\textbf{Sub}(a,b) & \Longrightarrow & a-b \\
\textbf{Mul}(a,b) & \Longrightarrow & a \times b \\
\textbf{Div}(a,b) & \Longrightarrow & a \div b \\
\textbf{sin}(a) & \Longrightarrow & \sin(a) \\
\textbf{cos}(a) & \Longrightarrow & \cos(a) \\
\textbf{exp}(a) & \Longrightarrow & \exp(a) \\
\textbf{log}(a) & \Longrightarrow & \log(|a|)
\end{array}
$$

In der kontextfreien Grammatik $\mathcal{G}_\mathrm{s}$ bedeutet x_{t-1} die tatsächliche jährliche durchschnittliche Sonnenfleckenrelativzahl im Jahr $t-1$, x_{t-2} die tatsächliche jährliche durchschnittliche Sonnenfleckenrelativzahl im Jahr $t-2$ usw. $-10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0$ sind die einfachen konstanten Werte im Bereich von $-10,0$ bis $10,0$.

Die CTF-GP wurde in fünf unabhängigen GP-Läufen durchgeführt. Bei jedem GP-Lauf des Multi-Verfahrens wurde der relative Geschwindigkeitsfaktor $G_{1:n_\mathrm{c}}$ in jeder Generation gemessen. Danach wurde der mittlere relative Geschwindigkeitsfaktor $\bar{G}_{1:n_\mathrm{c}}$ aus diesen fünf unabhängigen GP-Läufen ermittelt.

5.3.2 Ergebnisse des Experiments und Analyse

Die Ergebnisse des Experiments und ihre Analyse werden in zwei Teile aufgeteilt. Zuerst wird gezeigt, wie hoch der Zeitaufwand der Fitnessberechnung im Vergleich zum Gesamtzeitaufwand aller Vorgänge innerhalb einer Generation (z.B. Selektion, Erzeugung von Nachkommen durch Reproduktion, Kreuzung und Mutation, sowie Fitnessberechnung) beim Single-Verfahren ist. Danach wird der Vergleich zwischen dem Single-Verfahren und dem Multi-Verfahren dargestellt.

Zeitaufwand der Fitnessberechnung im Single-Verfahren

Die Fitnessberechnung ist der zeitaufwendigste Vorgang in der GP bzw. in der CTF-GP. Dies wird in Abbildung 5.4 deutlich dargestellt. Das obere Bild der Abbildung 5.4 stellt die durchschnittliche mittlere Ableitungstiefe innerhalb einer Generation bei dem Single-Verfahren aus fünf GP-Läufen dar. Dabei bedeutet die Ableitungstiefe eines Individuums D_{div} in CTF-GP die Anzahl von Nichtterminalsymbolen. Die mittlere Ableitungstiefe $\bar{D}_{\text{div}}$ ist damit die mittlere Ableitungstiefe von allen Individuen der Population in einer bestimmter Generation eines GP-Laufes.

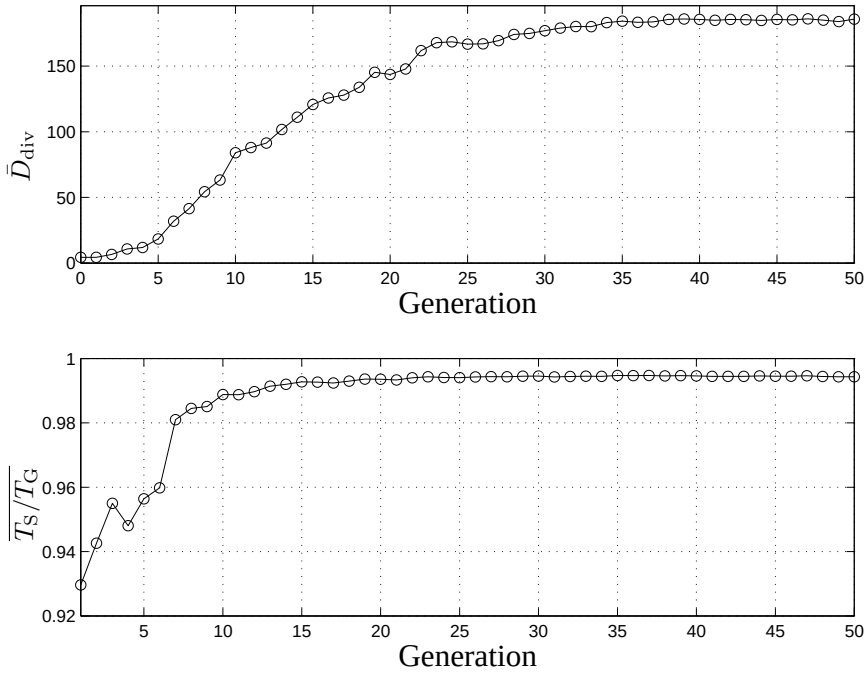


Abbildung 5.4: *oben:* durchschnittliche mittlere Ableitungstiefe $\bar{D}_{\text{div}}$ innerhalb einer Generation beim GP-Single-Verfahren aus fünf GP-Läufen, *unten:* durchschnittliches Verhältnis zwischen dem Zeitaufwand der Fitnessberechnung beim Single-Verfahren T_{S} und dem gesamten Zeitaufwand aller Vorgänge innerhalb einer Generation bei dem Single-Verfahren T_{G} aus fünf GP-Läufen

Am Anfang ist die durchschnittliche mittlere Ableitungstiefe $\bar{D}_{\text{div}}$ aus fünf GP-Läufen meist klein, wächst aber im Laufe der Evolution, bis die meisten Individuen die maximale Ableitungstiefe besitzen. Dann stagniert die durchschnittliche mittlere Ableitungstiefe $\bar{D}_{\text{div}}$ (ab ca. 40. Generation).

Das untere Bild der Abbildung 5.4 stellt das durchschnittliche Verhältnis zwischen dem Zeitaufwand der Fitnessberechnung T_{S} und dem gesamten Zeitaufwand aller Vorgänge innerhalb derselben Generation aus fünf GP-Läufen dar. Unter T_{G} versteht man den Zeitaufwand zur Durchführung aller Vorgänge innerhalb einer Generation z.B. Selektion, Erzeugung von Nachkommen durch Reproduktion, Kreuzung und

Mutation, sowie Fitnessberechnung. Der Zeitaufwand der Fitnessberechnung T_{S} aus dem Experiment beträgt durchschnittlich über 90% vom gesamten Zeitaufwand aller Vorgänge innerhalb einer Generation T_{G}.

Der mittlere Zeitaufwand der Fitnessberechnung ist proportional zur mittleren Ableitungstiefe des Induvidiums. Je länger die mittlere Ableitungstiefe $\bar{D}_{\mathrm{div}}$ ist, desto größer ist der Zeitaufwand der Fitnessberechnung T_{S}. Dagegen variiert die Zeit zur Selektion und Erzeugung von Nachkommen nur wenig und ist im Vergleich zur Fitnessberechnungzeit sehr klein. Daher wächst ebenfalls der Zeitaufwand der Fitnessberechnungzeit im Lauf der Evolution. Das mittlere Verhältnis zwischen dem Zeitaufwand der Fitnessberechnungszeit T_{S} und dem gesamten Zeitaufwand aller Vorgänge innerhalb einer Generation T_{G} stagniert ab der 40. Generation.

Vergleich zwischen dem Single-Verfahren und dem Multi-Verfahren

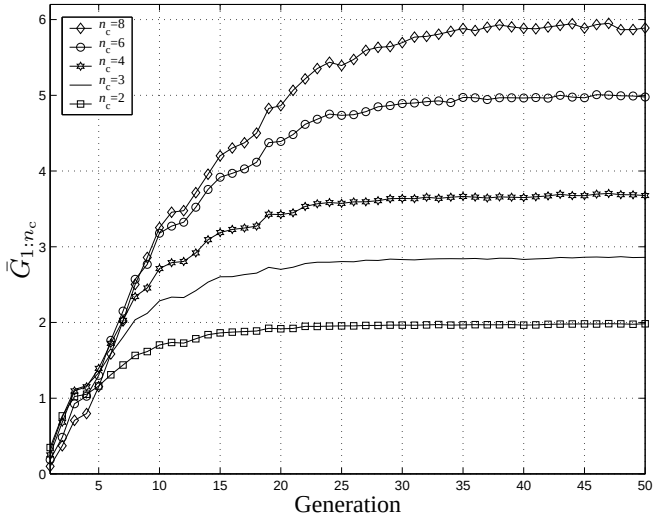


Abbildung 5.5: mittlerer relativer Geschwindigkeitsfaktor $\bar{G}_{1:n_{\mathrm{c}}}$ des Multi-Verfahrens mit $2, 3, 4, 6$ und 8 Clients

Nach dem Experiment mit den Multi-Verfahren wurde der mittlere relative Geschwindigkeitsfaktor $\bar{G}_{1:n_{\mathrm{c}}}$ (mit $n_{\mathrm{c}} = 2, 3, 4, 6$ und 8) aus fünf GP-Läufen berechnet. Abbildung 5.5 zeigt die Ergebnisse des Experiments, dessen Populationsgröße auf $M = 1000$ eingestellt ist. Alle fünf mittleren Geschwindigkeitsfaktoren $\bar{G}_{1:2}$, $\bar{G}_{1:3}$, $\bar{G}_{1:4}$, $\bar{G}_{1:6}$ und $\bar{G}_{1:8}$ aus fünf GP-Läufen zeigen einen sehr ähnlichen Verlauf.

In den frühen Generationen (bis zur 3. Generation) sind die meisten Individuen sehr klein. Dadurch können die Fitnessberechnungen sehr schnell durchgeführt werden. Infolgedessen liegen alle mittleren relativen Geschwindigkeitsfaktoren $\bar{G}_{1:n_{\mathrm{c}}}$ unter 1, und der mittlere relative Geschwindigkeitsfaktor $\bar{G}_{1:n_{\mathrm{c}}}$ bei einer kleinen Anzahl von Client-Rechnern n_{c} höher als der mittlere relative Geschwindigkeitsfaktor $\bar{G}_{1:n_{\mathrm{c}}}$ bei einer größeren Anzahl von Client-Rechnern n_{c}. Das Multi-Verfahren hat daher keinen Vorteil gegenüber dem Single-Verfahren.

Außerdem kommt es bei der Simulation von Multi-Verfahren mit 4 oder mehr Clients in den frühen Generationen (bis zur 3. Generation) sehr häufig vor, dass die Clients mit größeren Client-Nummern an der Fitnessberechnung nicht beteiligt sind. Die Fitnessberechnungen finden nur in den Clients mit kleinen Client-Nummern statt. Zum Beispiel bei der Simulation von Multi-Verfahren mit 4 Clients: Der 1. Client beendet die Fitnessberechnung des 1. Datenpakets, und schickt das Ergebnispaket zum Server zurück. Anschließend meldet der 1. Client, dass er frei, d.h. bereit zur Bearbeitung des weiteren Datenpakets, ist. Dies geschieht, bevor das 3. und 4. Datenpaket gesendet wird. In dieser Zeit bearbeitet der 2. Client noch das 2. Datenpaket. Daraus folgt, dass der Server das 3. Datenpaket zum 1. Client sendet. Wenn sich der 2. Client vor der Sendung des 4. Datenpaket frei meldet, wird der Server das 4. Datenpaket zum 2. Client senden. Die Clients mit größeren Client-Nummern, z.B. 3. Client oder 4. Client, werden daher an der Fitnessberechnung nicht beteiligt.

Im Lauf der Generation werden die Ableitungstiefen von den meisten Individuen größer, folglich vergrößert sich der Zeitaufwand der Fitnessberechnung. Wie in Abbildung 5.5 gezeigt wurde, hat das Multi-Verfahren ab der 5. Generation Vorteile gegenüber dem Single-Verfahren. Die mittleren relativen Geschwindigkeitsfaktoren $\bar{G}_{1:n_c}$ steigen weiter, bis ihr maximaler Wert erreicht wird. Die maximalen mittleren relativen Geschwindigkeitsfaktoren $\max \bar{G}_{1:n_c}$ bei einer bestimmten Anzahl n_c von Client-Rechnern sind in Tabelle 5.2 dargestellt. Die Ergebnisse sind ähnlich wie die besten Ergebnisse aus der Arbeit von Oussaidène in [Ous96, Ous97]. Die maximalen mittleren Geschwindigkeitsfaktoren $\max \bar{G}_{1:n_c}$ aus dem Experiment können leicht durch Gleichung (5.4) erklärt werden.

- Wenn man die Anzahl der Client-Rechner n_c erhöht, muss man auch die Anzahl der Pakete n_p erhöhen. Jedes Paket bekommt darum weniger Daten als vorher. Die Vorlaufzeit eines Pakets T_H wird nur leicht abnehmen, jedoch steigt der Anteil $n_p \cdot T_H$ in Gleichung (5.4) an. Aus diesem Grund nimmt der maximale mittlere relative Geschwindigkeitsfaktor $\max \bar{G}_{1:n_c}$ beim Erhöhen der Anzahl der Clients n_c nur langsam zu.

- Wenn man die Populationsgröße erhöht, wird die Last des Multi-Verfahrens vergrößert. Damit nimmt der Zeitaufwand der Fitnessberechnung im Single-Verfahren T_S in Gleichung 5.4 deutlich stärker als die Vorlaufzeit eines Pakets T_H zu. Aus diesem Grund steigt der maximale mittlere relative Geschwindigkeitsfaktor $\max \bar{G}_{1:n_c}$ beim Erhöhen der Populationsgröße an.

Die relativen Geschwindigkeitsfaktoren beim Multi-Verfahren mit 2 oder 3 Clients sind in manchen GP-Läufen besonders interessant, da sie größer sind als die Werte, die wir erwarten. Zum Beispiel betragen die größten relativen Geschwindigkeitsfaktoren mit 2 Clients 2,09 bei GP mit $M = 1000$ und 2,04 bei GP mit $M = 2000$.

Beim Multi-Verfahren mit 2 Clients bekommt jeder Client 500 Individuen zu verarbeiten, anstatt 1000 Individuen wie im Single-Verfahren. Außerdem kann das Betriebsystem Linux kleine Daten (z.B. 500 Individuen) besser als große Daten (z.B.

Anzahl der Clients n_c	$\max \bar{G}_{1:n_c}$ bei der Populationsgröße $M = 1000$	$\max \bar{G}_{1:n_c}$ bei der Populationsgröße $M = 2000$
2	1,98 (2,09)	1,99 (2,04)
3	2,87 (3,04)	-
4	3,70 (3,96)	3,80 (3,95)
6	5,01 (5,75)	5,35 (5,65)
8	5,95 (7,37)	6,55 (7,13)

Tabelle 5.2: Die maximalen mittleren relativen Geschwindigkeitsfaktoren $\max \bar{G}_{1:n_c}$ des Multi-Verfahrens mit 2, 3, 4, 6 und 8 Clients aus fünf GP-Läufen bei der Populationsgröße $M = 1000$ und $M = 2000$. Die in runde Klammern gesetzten Werte sind die größten relativen Geschwindigkeitsfaktoren aus fünf GP-Läufen.

1000 Individuen) im Rechner verwalten. Aus diesem Grund ist der größte relative Geschwindigkeitsfaktor $\max G_{1:n_c}$ im Multi-Verfahren mit 2 Clients bei manchen Generationen im GP-Lauf größer als 2. Bei Erhöhung der Anzahl der Clients (z.B. 3, 4, 6 und 8) wird sich der Anteil $n_p \cdot T_H$ in Gleichung (5.4) vergrößern, demnach wird dieser Vorteil des Betriebsystems Linux unterdrückt.

Dass der größte relative Geschwindigkeitsfaktor bei der Populationsgröße $M = 1000$ größer als der größte relative Geschwindigkeitsfaktor bei der Populationsgrößer $M = 2000$ ist, ist rein zufällig, da die Individuen in beiden Fällen verschieden sind.

5.4 Vorteile und Probleme des Programmpakets zur Parallelisierung der Fitnessberechnung auf ein verteiltes Rechnersystem

Aus dem Ergebnis des Experiments werden die Vorteile und Probleme des Programmpakets [For00] festgestellt, und wie folgt zusammengefasst:

Vorteile

- Das Programmpaket [For00] kann den Zeitaufwand der Fitnessberechnung reduzieren. Besonders bei Clustern mit zwei oder drei Client-Rechnern bekommt man den gewünschten relativen Geschwindigkeitsfaktor $G_{1:n_c}$. Dieser Vorteil ist jedoch von mehreren Faktoren, wie z.B.
  - der richtigen Einstellung der Anzahl der Pakete,
  - der Lastgröße von Anwendung oder GP, und

– der Übertragungsgeschwindigkeit des Netzes,

abhängig.

- Das Cluster könnte nicht nur bei GP-Anwendungen, sondern auch in anderen Anwendungen eingesetzt werden. Man muss die Anwendungen so verändern, dass sich ihre Prozesse in Datenpakete zerteilen lassen, und die Ergebnispakete lesbar und weiter verarbeitbar sind. Im Übrigen müssen alle Clients die entsprechenden Capabilities besitzen.

Probleme

- Die Anzahl der Pakete n_{p} ist zur Zeit konstant und wird immer vor dem GP-Lauf festgelegt. Diese Festlegung ist manchmal sehr schwierig, da man die Last des GP-Laufes nicht richtig einschätzen kann. Bei einer großen Anzahl von Paketen n_{p} bei kleiner Last des GP-Laufes bringt das Cluster keinen Vorteil mit sich. Außerdem könnte das Cluster, besonders bei kleiner Last des GP-Laufes, zu längerem Zeitaufwand als beim Single-Verfahren führen. Zur Erhöhung des effizienten Einsatzes des Clusters soll die Anzahl der Pakete n_{p} automatisch und dynamisch nach der Last des GP-Laufes in jeder Generation eingestellt werden. Die Last des GP-Laufes und die Geschwindigkeitsfaktoren $G_{1:n_{\mathrm{c}}}$ in der jetzigen n. Generation können durch ihre Werte in der vorherigen $(n-1)$. Generation geschätzt werden.

- Bei Experimenten mit mehr als acht Clients entsteht sehr häufig ein Problem, da Ergebnispakete nach ihrer Ankunft im Master-Rechner verloren gehen. Die Ursache wird noch untersucht und beseitigt.

5.5 Zusammenfassung

Das Programmpaket zur Parallelisierung der Fitnessberechnung auf verteilte Rechnersysteme ([For00]) ist eine Variante der globalen Parallelisierung. Es besteht aus zwei Teilen, Anwendung und Cluster.

Die Anwendung ist ein GP-Programm, in dem keine Fitnessberechnung stattfindet. Sie verteilt Individuen in Datenpakete und liest anschließend die Fitnesswerte aus den Ergebnispaketen.

Das Cluster besteht aus Server, Feed und Client. Die Anwendung, Server und Feed befinden sich in demselben Rechner, der Master-Rechner genannt wird. Server ist die zentrale Steuerungseinrichtung des Systems. Er verwaltet den Feed sowie die diversen Clients und koordiniert die einzelnen zu bearbeitenden Pakete. Feed stellt

die Schnittstelle zwischen Anwendung und Server dar. Die Clients bearbeiten erhaltene Datenpakete mit den entsprechenden Capabilities, also Fitnessberechnung, und schicken Ergebnisse in Ergebnispaketen dem Server zurück.

Das Programmpaket kann den Zeitaufwand bei der Fitnessberechnung reduzieren. Je größer die Last der GP ist, desto mehr steigt der relative Geschwindigkeitsfaktor $G_{1:n_c}$ nahezu linear an. Die Anzahl der Clients n_c ist theoretisch unbegrenzt. Sie muss jedoch an die Geschwindigkeit der Rechner und dem Netzwerk angepasst sein.

6 Einführung in die Chaostheorie

Dieses Kapitel beschäftigt sich mit der Einführung in die Chaostheorie. Wir beginnen im Abschnitt 6.1 und 6.2 mit dem Begriff der dynamischen Systeme und Attraktoren. Das deterministische Chaos wird im Abschnitt 6.3 definiert. Im Abschnitt 6.4 werden zwei bekannte Methoden zur Charakterisierung chaotischen Verhaltens erläutert. Dies sind die Lyapunov-Exponenten und die Dimensionen. Zum Schluss werden einige Beispiele des dynamischen Chaos demonstriert. Für weiterreichende Darstellungen der Theorie wird auf das Buch von H. Nagashima und Y. Baba [Nag92] und A. A. Tsonis [Tso92], sowie das deutschsprachige Buch von R. Leven, B. Koch und B. Pompe [Lev89] und O. Loistl und I. Betz [Loi93] verwiesen.

6.1 Dynamisches System

Chaotisches Verhalten kennt man vor allem aus der Theorie dynamischer Systeme. Zur Beschreibung eines derartigen Systems muss man seinen *Zustand* (die wesentliche Information über ein System) und die *Dynamik* (eine Vorschrift, welche die zeitliche Änderung des Zustandes angibt) kennen. Die zeitliche Entwicklung kann man sich im *Phasen-* bzw. *Zustandsraum* veranschaulichen, dessen Koordinaten die Komponenten des Zustands sind.

Definition 6.1 *[Bro93] Phasenraum oder Zustandsraum wird ein Koordinatensystem genannt, in dem die Zustandsvariablen eines Systems als Koordinaten dienen. Der Phasenraum enthält die Menge aller möglichen Zustände eines Systems und kann endlich– oder unendlichdimensional sein. Die im Phasenraum ablaufende Zeit kann stetig oder diskret sein.*

Der Phasenraum eignet sich besonders zur Darstellung der Systemdynamik.

Definition 6.2 *[Moe92]*

- *Ein System heißt eindimensional, wenn seine Zustandsvariable eine Komponente aufweist.*
- *Ein System heißt mehrdimensional, wenn seine Zustandsvariable mehrere Komponenten aufweist.*

- *Ein System heißt endlichdimensional, wenn seine Zustandsvariable endlich viele Komponenten aufweist.*

Definition 6.3 *[Loi93] Ein dynamisches System sei nachfolgend durch ein System von gewöhnlichen Differenzengleichungen:*

$$\boldsymbol{x}(t+1) = \boldsymbol{f}(\boldsymbol{x}(t)) \tag{6.1}$$

oder durch ein System von gewöhnlichen Differentialgleichungen in der Form:

$$\frac{d\boldsymbol{x}}{dt} \equiv \dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x}) \tag{6.2}$$

gegeben.

Dabei bezeichnen $\boldsymbol{x} \equiv (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ *und* $\boldsymbol{F} \equiv (F_1, F_2, ..., F_n)$ *und* $\boldsymbol{f} \equiv (f_1, f_2, ..., f_n)$ *reellwertige Funktionsvektoren mit* $\mathbb{R}^n \to \mathbb{R}^n$.

$\mathbb{R}^n$ *bezeichnet einen n-dimensionalen euklidischen Vektorraum mit dem Skalarprodukt* $(\boldsymbol{x}, \boldsymbol{y}) = \sum_i x_i \cdot y_i$ *und der Norm* $|\boldsymbol{x}| \equiv (\boldsymbol{x}, \boldsymbol{x})^{\frac{1}{2}}$.

$\boldsymbol{x}(t)$ *stellt den Zustand des Systems zur Zeit t dar, der im Falle von Differentialgleichungen durch* $t \in \mathbb{R}$ *und im Falle von Differenzengleichungen durch* $t \in \mathbb{Z}$ *definiert ist.*

Mit der Differenzengleichung (6.1) wird ein *zeitdiskretes System*, mit der Differentialgleichung (6.2) ein *zeitkontinuierliches System* beschrieben.

Definition 6.4 *[Bro93] Eine Abbildung* $f : A \to B$ *ist eine Vorschrift, die jedem Element a aus einer Menge A genau ein Element* $b = f(a)$ *aus einer Menge B zuordnet:*

$$a \in A, \quad a \to b = f(a) \in B. \tag{6.3}$$

Man nennt $b = f(a)$ *ein Bild von a und a das Urbild von b.*

Durch das System (6.1) und (6.2) werden spezielle Abbildungen beschrieben, die von den *Zustandsvariablen* x_i $(i = 1, 2, ..., n)$ und Parametern abhängen. Oft geht dabei wie in (6.1) die zeitliche Entwicklung in diskreten Schritten, *Iterationen* genannt, vor sich. Man spricht dann von *iterierter Abbildung*.

Im zeitkontinuierlichen Fall sei angenommen, dass die Funktionen $F_i (i = 1, 2, ..., n)$ im $\mathbb{R}^n$ hinsichtlich aller Argumente x_i stetig differenzierbar seien. Dieses ist eine hinreichende Bedingung für die Existenz eindeutiger Lösungen zu einem beliebigen Anfangszustand. Bei gegebenem Anfangszustand folgt daraus eindeutig jeder zukünftige als auch vergangene Zustand eines dynamischen Systems. Man spricht infolgedessen auch von der *Determiniertheit* des Systems.

Ist im zeitdiskreten Fall $\boldsymbol{f}$ nicht umkehrbar, so nennt man das dynamische System *halbdeterminiert*.

Für die Zeit t folgt daraus $t \in \mathbb{Z}$. Hängen die Funktion $\boldsymbol{F}$ bzw. $\boldsymbol{f}$ nicht explizit von der Zeit ab, so bezeichnet man das dynamische System als *autonom*, andernfalls als *nichtautonom*. Formal kann ein nichtautonomes System durch Einführung der Zeit t jederzeit in ein autonomes System überführt werden.

Ein zeitdiskretes System bezeichnet man als lösbar, wenn es in die Form $\boldsymbol{x}(t) = \boldsymbol{f}^t(\boldsymbol{x}_0) \equiv \boldsymbol{f}^t\boldsymbol{x}_0$ überführt werden kann ($\boldsymbol{f}^t \equiv \boldsymbol{f} \circ \boldsymbol{f}^{t-1}, \boldsymbol{f}^0 \equiv 1$) und somit zu jedem Anfangszustand $\boldsymbol{x}_0$ unmittelbar jeder zukünftige Zustand angegeben werden kann. Eine spezielle Lösung ordnet einem Anfangszustand $\boldsymbol{x}_0$ bei festem t einen eindeutigen Zustand $\boldsymbol{x}(\boldsymbol{x}_0, t)$ zu. Die gesamte Lösungsmenge ordnet demzufolge allen Anfangszuständen aus $\mathbb{R}^n$ nach der Zeit t neue Zustände zu. Man nennt eine solche Abbildung *Phasenfluss* auf dem Phasenraum $\mathbb{R}^n$. Einzelne Zustände im $\mathbb{R}^n$ erhalten die Bezeichnung *Phasenpunkte*.

Variiert man unter Festhalten einer Anfangsbedingung $\boldsymbol{x}_0$ die Zeit t, so erhält man eine Lösungskurve des dynamischen Systems, die als *Trajektorie* oder *Orbit* des Flusses $\boldsymbol{f}^t$ zur Anfangsbedingung $\boldsymbol{x}_0$ bezeichnet wird.

6.1.1 Konservative Systeme

Für ein autonomes konservatives System gilt, dass das Volumen eines Volumenelementes im Phasenraum erhalten bleibt, was einer Energieerhaltung entspricht. Formalisiert bedeutet dieses, dass die Divergenz eines durch $\boldsymbol{F}$ im Phasenraum $\mathbb{R}^n$ aufgespannten Vektorfeldes gleich 0 ist [Eck81].

$$\operatorname{div} \boldsymbol{F} \equiv \sum_{i=1}^{n} \frac{\partial F_i(\boldsymbol{x})}{\partial x_i} = 0 \tag{6.4}$$

Ein Volumenelement wird in konservativen Systemen unter der Wirkung eines Flusses $\boldsymbol{f}^t$ also höchstens deformiert, während dabei auf jeden Fall das Volumen als solches erhalten bleibt.

6.1.2 Dissipative Systeme

Ursprünglich wurden mechanische Systeme mit Dämpfung oder Reibung, d.h. in denen die Energie keine Erhaltungsgröße ist, dissipativ genannt.

Eine der wichtigsten Eigenschaften dissipativer Systeme ist, dass sich ein Volumenelement des Phasenraums $\mathbb{R}^n$ unter der Wirkung des Flusses $\boldsymbol{f}^t$ zusammenzieht und

schließlich zum Volumeninhalt Null wird (vorausgesetzt, dass dem dynamischen System keine neue Energie zugeführt wird). Nach einem solchen transienten Übergang befindet sich der Systemzustand auf einem Attraktor, dessen Dimension kleiner ist als die des ursprünglichen Phasenraums.

Dissipation impliziert jedoch nicht, dass das Phasenraumvolumen für t gegen unendlich gegen Null strebt und sich der Systemzustand letztendlich immer auf einem Fixpunkt einfindet.

Wird als Raum für die Anfangsbedingungen der gesamte Phasenraum $\mathbb{R}^n$ zugelassen, so zieht sich der systemzugängliche Phasenraum im Mittel der Zeit auf einen niedriger dimensionalen Unterraum des $\mathbb{R}^n$ zusammen, d.h. dass sich die Zahl der zur Beschreibung der Dynamik notwendigen Zustandsvariablen reduziert.

6.2 Attraktoren

Die nun zuvor beschriebene Volumenkontraktion in einem dissipativen System hat zur Folge, dass sich ein Volumen $V \subset \mathbb{R}^n$ unter der Wirkung eines Flusses im Regelfall auf einen Attraktor $\mathcal{A}$ niedriger Dimension zusammenzieht. Trajektorien $\boldsymbol{x}(\boldsymbol{x}_0, t)$ mit $\boldsymbol{x}_0 \in V \setminus \mathcal{A}$ bezeichnet man als *Transiente*. Transiente beschreiben also das Übergangsverhalten im Einzugsbereich des Attraktors.

6.2.1 Definition von Attraktoren

Definition 6.5 *[Loi93] Ein Attraktor eines Flusses $\boldsymbol{f}^t$ ist eine kompakte Menge $\mathcal{A}$ mit den folgenden Eigenschaften:*

1. *$\mathcal{A}$ ist invariant unter dem Fluss $\boldsymbol{f}^t$, d.h. $\boldsymbol{f}^t(\mathcal{A}) = \mathcal{A} \ \ \forall \ \ t$.*
2. *$\mathcal{A}$ besitzt eine offene Umgebung U, die sich unter dem Fluss $\boldsymbol{f}^t$ auf $\mathcal{A}$ zusammenzieht, d.h. $\lim\limits_{t\to\infty} \boldsymbol{f}^t(U) = \mathcal{A}$.*
3. *Der Fluss von $\mathcal{A}$ ist wiederkehrend, d.h. dass $\mathcal{A}$ keine Untermenge besitzt, die transient ist.*
4. *$\mathcal{A}$ kann nicht in nichttriviale, kompakte, invariante Mengen zerlegt werden.*

Das Einzugsgebiet oder Bassin von $\mathcal{A}$ lässt sich durch die offene Menge aller zugelassenen Anfangsbedingungen $\boldsymbol{x}_0$ definieren, für die gilt:

$$\lim_{t\to\infty} \boldsymbol{f}^t(\boldsymbol{x}) \in \mathcal{A} \tag{6.5}$$

Der Attraktor ist also die Menge, gegen die schließlich alle von einer Umgebung U des Attraktors ausgehenden Trajektorien konvergieren.

6.2.2 Fixpunktattraktoren

Die einfachste Form eines Attraktors ist ein *asymptotisch stabiler Fixpunkt* (Abbildung 6.1). Das dynamische System bewegt sich unabhängig vom Anfangspunkt und landet mit der Zeit zu einem Fixpunkt im Phasenraum.

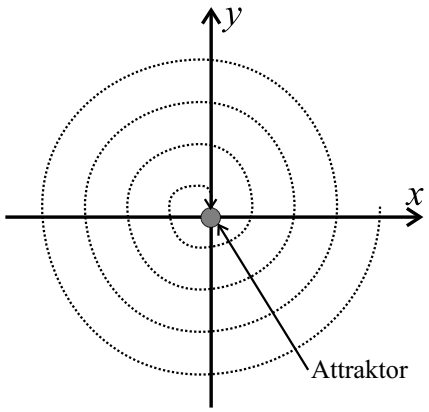


Abbildung 6.1: Fixpunktattraktor

Definition 6.6 *[Bro93] Fixpunkt nennt man einen Punkt* $\boldsymbol{x}^* = (x_1^*, ..., x_n^*)$*, wenn er unter der Wirkung der Abbildung in sich selbst übergeht:*

$$\boldsymbol{x}^* = \boldsymbol{f}^t(\boldsymbol{x}^*) \tag{6.6}$$

für alle $t > 0$*.*

Definition 6.7 *[Arr94] Ein Fixpunkt* $\boldsymbol{x}^*$ *heißt stabil, wenn für alle Umgebungen* U *von* $\boldsymbol{x}^*$ *eine Umgebung* $U' \subseteq U$ *von* $\boldsymbol{x}^*$ *existiert, so dass für* $\boldsymbol{x} \in U'$ *gilt* $\boldsymbol{f}^t(\boldsymbol{x}) \in U$ *für alle* $t > 0$*.*

Definition 6.7 sagt also aus, dass Punkte in der Nähe eines stabilen Fixpunktes bei Iteration in der Nähe bleiben für positive t. Ist ein Fixpunkt $\boldsymbol{x}^*$ stabil und $\lim_{t\to\infty} \boldsymbol{f}^t(\boldsymbol{x}) = \boldsymbol{x}^*$ für alle $\boldsymbol{x}$ in einer Umgebung von $\boldsymbol{x}^*$, dann nennt man den Fixpunkt asymptotisch stabil. Trajektorien von Punkten in der Nähe von asymptotisch stabilen Punkten bewegen sich auf ihn hin für steigendes t.

6.2.3 Grenzzyklische Attraktoren

Die nächstkompliziertere Form eines Attraktors ist ein *Grenzzyklus* (Abbildung 6.2).

Definition 6.8 *[Loi93] Eine Trajektorie* $\boldsymbol{x}(t)$ *eines dynamischen Systems heißt geschlossene Bahn, wenn ein* $t \neq 0$ *existiert, so dass* $\boldsymbol{f}^t(\boldsymbol{x}) = \boldsymbol{x}$ *und* $\boldsymbol{x}(t)$ *kein Gleichgewichtspunkt ist.*

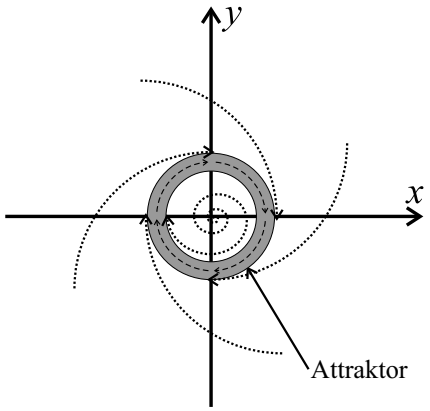


Abbildung 6.2: Grenzzyklischer Attraktor

Handelt es sich bei dieser Bahn um einen Attraktor so spricht man von einem Grenzzyklus. Präzisiert lautet die Definition:

Definition 6.9 *[Loi93] Eine geschlossene Bahn $\boldsymbol{x}(t)$ ist ein Grenzzyklus, wenn alle Lösungen des Systems, die durch Punkte einer Umgebung $U(\boldsymbol{x})$ verlaufen, gegen $\boldsymbol{x}(t)$ konvergieren, d.h.*

$$\lim_{t\to\infty} d(\boldsymbol{f}^t(\boldsymbol{x})) = 0 \tag{6.7}$$

mit d als der Distanz zwischen der Trajektorie und dem Grenzzyklus.

6.2.4 Torus-Attraktoren

In höherdimensionalen Systemen mit $n \geq 3$ sind wesentlich kompliziertere Attraktorstrukturen möglich. Zusammengesetzte Schwingungen oder quasiperiodisches Verhalten gehören zum *Torus-Attraktor* (Abbildung 6.3).

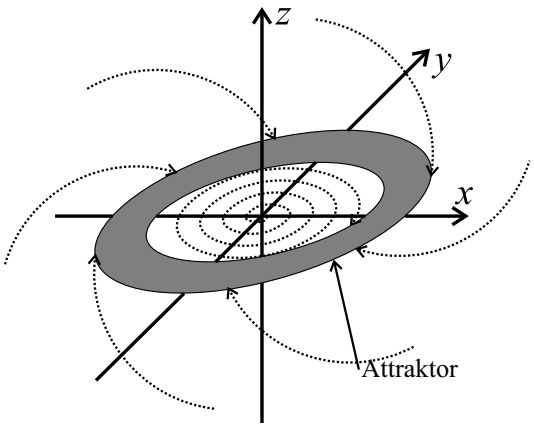


Abbildung 6.3: Torus-Attraktor

Es ist jedoch hervorzuheben, dass bei auf Tori verlaufenden Lösungen zwei beliebig nahe beieinander startende Trajektorien sich im Zeitverlauf nicht voneinander entfernen, sie divergieren nicht. Tori, Grenzzyklen und Fixpunkten ist diese Eigenschaft gemeinsam, die sie als Klasse der nicht chaotischen Attraktoren charakterisieren.

6.2.5 Seltsame Attraktoren

Neben den vorstehend beschriebenen Attraktortypen existieren aber auch solche, auf denen nicht-periodische Bewegungen mit *sensitiver Abhägigkeit von den Anfangsbedingungen* stattfinden. Die Klasse der Attraktoren, die durch diese Eigenschaft gekennzeichnet sind, werden als *seltsame* oder auch chaotische Attraktoren (Abbildung 6.4) bezeichnet. Grundlegendes Charakteristikum der seltsamen Attraktoren besteht also in der exponentiellen Divergenz benachbarter Trejektorien auf dem Attraktor.

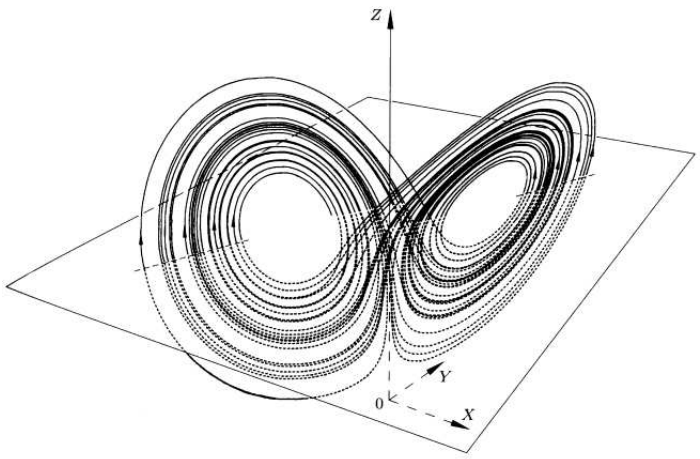


Abbildung 6.4: Seltsamer Attraktor

Definition 6.10 *[Loi93] Ein chaotischer oder seltsamer Attraktor ist ein Attraktor mit einer sensitiven Abhängigkeit von den Anfangsbedingungen.*

Sensitive Abhägigkeit lässt sich durch das Vorhandensein von mindestens einem positiven Lyapunov-Exponenten nachweisen. In den seltsamen Attraktoren ist daher mindestens ein Lyapunov-Exponent positiv, mindestens einer Null, und mindestens einer negativ[1].

6.3 Deterministisches Chaos

In dieser Arbeit soll der Begriff *Chaos* als *deterministisches Chaos* verstanden werden.

[1]Siehe Abschnitt 6.4.1

Definition 6.11 *[Hao90] Ein physikalsches dynamisches System, das irreguläre oder scheinbar zufällige Bewegungen ausführt, wird* deterministisch chaotisch *genannt, wenn es folgende Eigenschaften besitzt:*

1. *Die das System beschreibenden Gleichungen sind deterministisch.*
2. *Kein äußeres Rauschen wird (dem System) zugeführt.*
3. *Die zeitliche Entwicklung des Systems hängt sensitiv von kleinen Änderungen des aktuellen Zustandes der Bewegung ab (sensitive Abhängigkeit vom Anfangszustand).*
4. *Es existiert eine Reihe globaler Größen, die nicht vom Anfangszustand abhängen. Diese invarianten Größen werden durch Mittelung über lange Zeiten oder vieler Bewegungszyklen gewonnen.*
5. *Der irreguläre Bewegungszustand wird durch das Durchstimmen eines Systemparameters über eine Reihe von Ereignissen, das sog. Szenario, z.B. das der Periodenverdopplung, erreicht.*

Deterministisch chaotische Bewegungen finden sich in einer großen Reihe natürlicher und technischer Systeme und zwar sowohl in solchen mit sehr vielen als auch in solchen mit nur wenigen Freiheitsgraden. Die Analyse solcher Systeme hat überraschenderweise ergeben, dass eine Reihe chaotischer Systeme mit ursprünglich vielen Freiheitsgraden durch Gleichungen mit nur wenigen Variablen beschrieben werden können. Ein Beispiel dafür ist die logistische Abbildung.

Es scheint paradox, dass Chaos deterministisch ist, erzeugt nach festen Regeln ohne stochastische Elemente. Prinzipiell ist die Zukunft durch die Vergangenheit vollständig bestimmt, aber praktisch werden kleine Fehler verstärkt - das Verhalten ist deshalb zwar kurzfristig vorhersagbar, langfristig aber unvorhersagbar.

6.4 Methoden zur Charakterisierung chaotischen Verhaltens

Die Dynamik eines Systems wird von der Geometrie des Phasenraumes und seines Attraktors beschrieben. Diese Geometrie lässt sich durch eine Serie von Dimensionen und Lyapunov-Exponenten quantifizieren.

6.4.1 Lyapunov-Exponenten

Der Lyapunov-Exponent stellt ein Maß zur Bestimmung des Grades der sensitiven Abhägigkeiten von den Anfangsbedingungen eines Systems dar. Er, (bzw. im höherdimensionalen Fall) sie, eignen sich somit insbesondere als Indikator für Stabilität oder Chaos in der Analyse eines zu untersuchenden dynamischen Prozesses.

Lyapunov-Exponenten λ_i sind reele Zahlen, die eine exponentielle Konvergenz ($\lambda_i < 0$), Neutralität ($\lambda_i = 0$) oder Divergenz ($\lambda_i > 0$) zweier eng benachbarter Trajektorien eines dynamischen Systems im zeitlichen Mittel beschreiben.

Für ein ein-dimensionales zeitdiskretes System lässt sich der Lyapunov-Exponent berechnen gemäß:

$$\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N-1} \ln |f'(x_i)| \tag{6.8}$$

Wobei $f'(x_i)$ die Ableitung der Funktion $f(x_i)$ darstellt.

Trajektorien, die am Anfang durch den Abstand ε voneinander getrennt sind haben nach N Iterationen den Abstand $\varepsilon \cdot e^{\lambda}$, wobei e^{λ} *Dehnungskoeffizient* genannt wird.

- Für $\lambda < 0$ wird die Dehnung $\varepsilon \cdot e^{\lambda} < 1$, d.h. es findet eine Annäherung der Trajektorien statt.
- Für $\lambda > 0$ wird die Dehnung $\varepsilon \cdot e^{\lambda} > 1$, so dass die Trajektorien exponentiell auseinanderlaufen.

Die Anzahl der Lyapunov-Exponenten richtet sich nach der Dimension des Phasenraumes. d.h. der Dimension des untersuchten dynamischen Systems. Ein n-dimensionales System besitzt n Lyapunov-Exponenten, d.h., dass sich die durch den dynamischen Prozess erzeugten Trajektorien in Richtung jeder Dimensionsachse des Systems im Phasenraum entwickeln können. Die sensitive Abhängigkeit lässt sich durch das Vorhandensein von mindestens einem positiven Lyapunov-Exponenten nachweisen. Der größte Lyapunov-Exponent erhält den Index 1.

Die Summe aller Lyapunov-Exponenten lässt sich als Maß zur Klassifizierung des untersuchten dynamischen Prozesses in

1. konservativ (das Volumen des Phasenraumes bleibt gleich), falls:

$$\sum_{i=1}^{n} \lambda_i = 0$$

2. dissipativ (das Volumen des Phasenraumes nimmt über die Zeit ab), falls:

$$\sum_{i=1}^{n} \lambda_i < 0$$

heranziehen.

Außerdem können verschiedene dynamische Systemzustände anhand des Lyapunov-Spektrums klassifiziert werden. Die Klassifikation des Verhaltens des dynamischen Systems und den Attraktortypen ist folgendermaßen (in den runden Klammern wird das Vorzeichen der Lyapunov-Exponenten angegeben):

- Spektrum der Lyapunov-Exponenten: $(-, -, ..., -)$
  - Verhalten: Stabilität
  - Attraktortyp: Fixpunkt
- Spektrum der Lyapunov-Exponenten: $(0, -, -, ..., -)$
  - Verhalten: Periodizität
  - Attraktortyp: Grenzzyklus
- Spektrum der Lyapunov-Exponenten: $(\underbrace{0, 0, ..., 0}_{n}, -, -, ..., -)$
  - Verhalten: Quasiperiodizität
  - Attraktortyp: stabiler n-Torus
- Spektrum der Lyapunov-Exponenten: $(+, 0, -, -, ..., -)$
  - Verhalten: Chaotisches Verhalten
  - Attraktortyp: Seltsamer Attraktor

6.4.2 Dimensionen

Zur quantitativen Charakterisierung von Attraktoren wurde eine Vielzahl von Dimensionsbegriffen eingeführt. Ihnen ist gemeinsam, dass sie für nicht-chaotische Attraktoren wie Fixpunkte, Grenzzyklen und n-Torus ganzzahlige Werte $0, 1, ..., n$ annehmen, während sie seltsame Attraktoren durch eine gebrochenzahlige Dimension charakterisieren. Gebrochenzahlige Dimensionen wurden eingeführt, um die auf beliebig kleinen Skalen bestehenden diffizilen Strukturen die so genannten *Fraktale*[2] charakterisieren zu können. Es zeigte sich, dass für eine quantitative Charakterisierung von natürlichen Objekten wie Küstenlinien, Wolken und Niederschlagsgebieten fraktale Dimensionen besser geeignet sind als der herkömmliche euklidische Dimensionsbegriff. Dieser ordnet jedem geometrischen Objekt eine *ganze* Zahl zu. So besitzt im euklidischen Sinne ein Punkt die Dimension 0, eine Linie die Dimension 1, eine Fläche die Dimension 2 usw.

[2] Ein Fraktal ist ein Objekt, das aus Teilen besteht, die ihrerseits in gewissem Sinne dem gesamten Objekt ähneln.

Die fraktale Dimension

Die fraktale Dimension gibt nun, vereinfacht ausgedrückt, die Anzahl der paarweise unabhängigen Zustandsgrößen an, die die Bewegung auf dem Attraktor charakterisieren.

Zerlegt man nun den Phasenraum in m-dimensionale Kuben der Kantenlänge ε, so ist es möglich abzuzählen, wieviele dieser Boxen notwendig sind, um den Attraktor damit vollständig zu überdecken. Sei nun $N(\varepsilon)$ die minimale Anzahl der dazu benötigten Boxen, so ist durch die Beziehung

$$D_F = -\lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log \varepsilon} \tag{6.9}$$

die fraktale Dimension D_F definiert. Die fraktale Dimension einer Menge $A \subset \mathbb{R}^n$ gibt also an, wie sich die Anzahl $N(\varepsilon)$ der zu ihrer Überdeckung benötigten n-dimensionalen Elementarkuben ändert, wenn die Kantenlänge ε immer kleiner wird. Beispielhaft sei eine Kochsche Schneeflocke betrachtet.

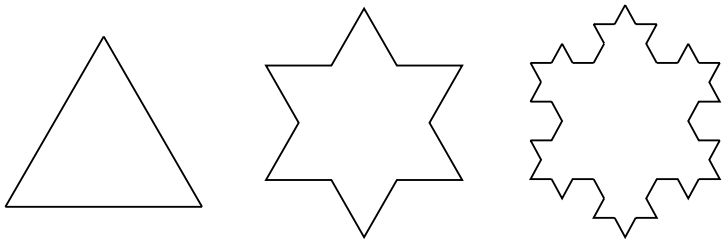


Abbildung 6.5: Kochsche Schneeflocke

Beispiel 6.1 (Kochsche Schneeflocke) *Die Seiten eines gleichseitigen Dreiecks werden in drei gleiche Teile geteilt. Auf jedem mittleren Drittel wird nach außen hin jeweils ein gleichseitiges Dreieck aufgesetzt und so fort (Siehe Abbildung 6.5). So entsteht als äußere Begrenzung eine gebrochene Linie, die* Kochsche Kurve*, die immer wieder von neuem gleichseitige Dreiecke einbezieht. Die Grenzmenge heißt Kochsche Schneeflocke. Sie stellt eine geschlossene, nirgends differenzierbare Linie unendlicher Länge dar, die aber eine endliche Fläche umschließt.*

Die minimale Anzahl von notwendigen Quadraten der Seitenlänge ε, die zur vollständigen Überdeckung des so erhaltenen geometrischen Objekts benötigt werden, entwickelt sich offensichtlich folgendermaßen:

$$\begin{array}{lllll}
\text{Ausgangssituation} & \varepsilon = 1 & \Longrightarrow & N(\varepsilon) = 1 \\
\text{1. iterierte} & \varepsilon = 1/3 & \Longrightarrow & N(\varepsilon) = 4 \\
\text{2. iterierte} & \varepsilon = 1/9 & \Longrightarrow & N(\varepsilon) = 16 \\
\quad \vdots & & & \\
n\text{. iterierte} & \varepsilon = (1/3)^n & \Longrightarrow & N(\varepsilon) = 4^n
\end{array}$$

Die fraktale Dimension der Kochschen Schneeflocke ist:

$$D_F = \lim_{n\to\infty} \lim_{\varepsilon\to 0} \frac{\log 4^n}{\log \frac{1}{(\frac{1}{3})^n}} = \lim_{\varepsilon\to 0} \frac{\log 4}{\log 3} \approx 1,2618... \tag{6.10}$$

Weitere fraktale Dimension

Viele Fraktale werden nicht nur durch die fraktale Dimension charakterisiert, sondern mit einem ganzen Satz anderer Dimensionen, z.B. die Informationsdimension, die Renyi-Dimension usw. Um den Umfang dieser Einführung nicht unnötig zu vergrößern, wird das Thema an dieser Stelle nicht weiter vertieft.

6.5 Beispiele des dynamischen Chaos

6.5.1 Logistische Abbildung

Hierbei handelt es sich um eine ein-dimensionale diskrete Abbildung im Intervall $[0, 1]$ auf sich selbst [May76]:

$$x_{t+1} = f(x_t) = r \cdot x_t \cdot (1 - x_t) \quad \text{mit} \quad 0 \leq r \leq 4\ ,\ t \in \mathbb{Z} \tag{6.11}$$

Während der Kontrollparameter r konstant gehalten wird, wird die Variable x entsprechend der obigen Vorschrift iteriert und ergibt so eine Zeitreihe $\{x_t\}$. Funktionale Zusammenhänge dieser Art sind in viele Fachdisziplinen nachweisbar. So z.B. in der Biologie, wo Insekten existieren, die zu Beginn der kalten Jahreszeit sterben und deren nächste Generation im folgenden Frühjahr aus den Eiern schlüpft. x_{t+1} ist dann beispielsweise die Individuendichte pro Flächeneinheit der $(t + 1)$-ten Generation in Abhängigkeit der Individuendichte x_t, d.h. der Dichte der Vorgängergeneration. Für kleine Populationsdichten folgt aus dem funktionalen Zusammenhang der logistischen Abbildung ein nahezu exponentielles Wachstum. Unbegrenztes Wachstum wird jedoch durch den quadratischen Term verhindert.

Externe Einflüsse auf das System, z.B. Umwelteinflüsse, sind im Parameter r zusammengefasst. Der einzige die Systemdynamik beeinflussende Parameter ist dann offenbar r. Somit kommt der Untersuchung des dynamischen Verhaltens der logistischen Abbildung in Abhängigkeit von r eine zentrale Bedeutung zu.

Der (x_t-x_{t+1})-Graph kann dazu dienen, die Dynamik der logistischen Abbildung zu verdeutlichen (siehe Abbildung 6.7). Dazu beginnt man an einem gegebenen Punkt x_t auf der waagrechten Achse, geht dann senkrecht nach oben bis zur Kurve und von dort nach links, wo man den darauffolgenden Wert x_{t+1} findet. Von dort geht man wieder waagrecht zurück bis zur eingezeichneten Diagonalen und dann senkrecht nach unten, bis man die waagrechte Achse wieder erreicht hat, aber nun beim Punkt

x_{t+1}, und beginnt von vorne. Änderungen von r können dazu führen, dass sich das qualitative Verhalten der Zeitreihe bzw. des Orbits grundlegend ändert. Dieses wird in Abbildung 6.6 und 6.7 veranschaulicht.

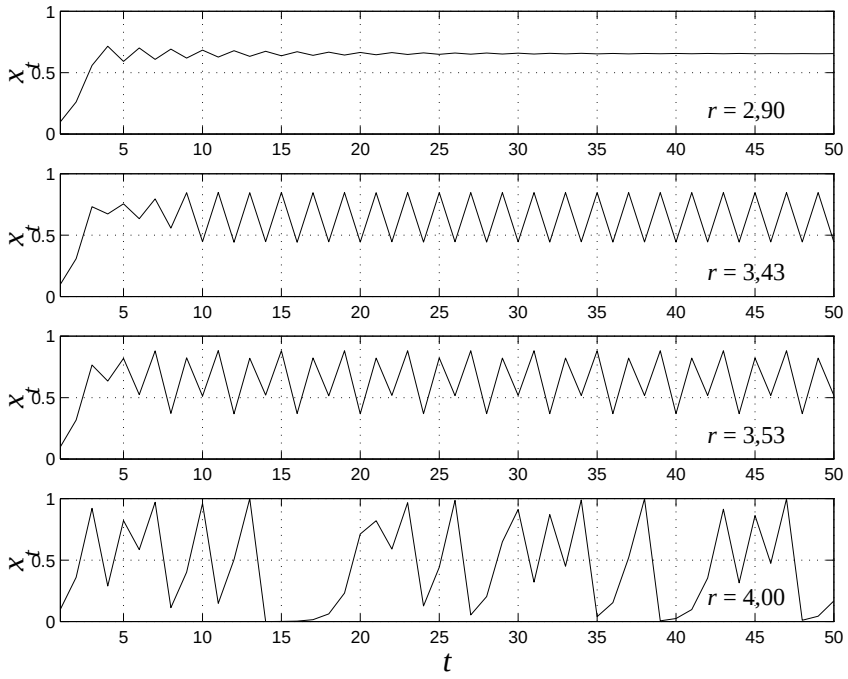


Abbildung 6.6: Zeitreihe $\{x_t\}$ der logistischen Abbildung beim gleichen Startwert mit vier verschiedenen Werten des Parameters ($r = 2,90$; $r = 3,43$; $r = 3,53$ und $r = 4,00$)

Man kann folgende Fälle unterscheiden:

- Anziehender Fixpunkt als Attraktor, wenn $r < 3$. Das System läuft von den meisten Anfangsbedingungen aus auf einen Fixpunkt zu.

- Bei größeren Werten des Parameters r, der die Form der Parabel bestimmt, z.B. $3 \leq r < r_2$ (wobei $r_2 \approx 3,44949...$), tritt ein Grenzzyklus der Periode 2 auf. Das System springt zwischen zwei Werten hin und her.

- Macht man die Steilheit der Parabel noch größer, z.B. $r_2 \leq r < r_3$ (wobei $r_3 \approx 3,54409...$), so tritt ein Grenzzyklus mit der Periode 4 auf, d.h. Periodenverdopplung. Nach vier Schritten erreicht das System wieder den gleichen Zustand wie beim ersten.

- Für noch größere r findet man Grenzzyklen mit immer längerer Periode $(8, 16, 32, ...)$.

- Ab einem bestimmten kritischen Wert $r \geq r_\infty \approx 3,569945...$ wird die Periode unendlich, das System also aperiodisch und chaotisch.

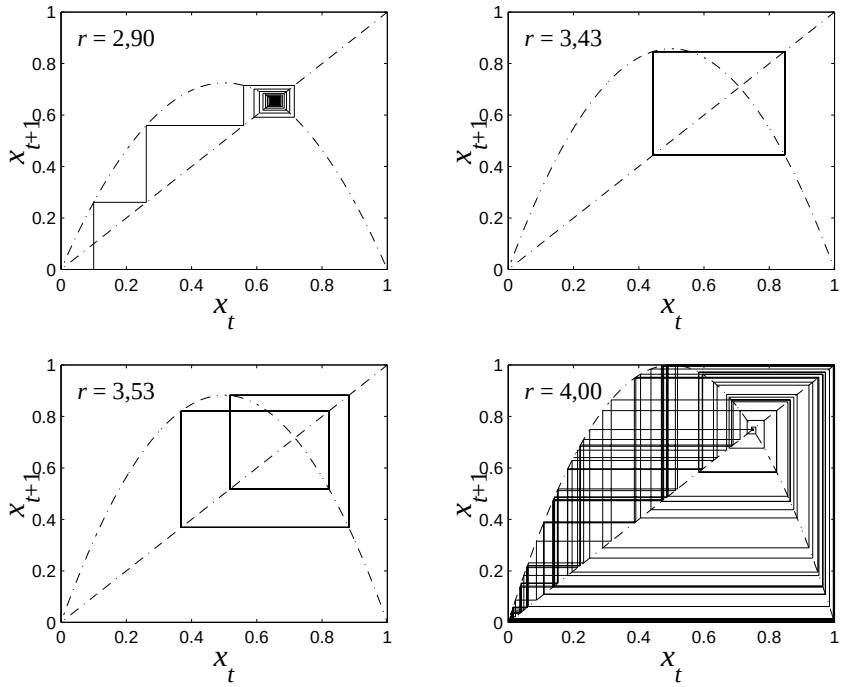


Abbildung 6.7: Iterationsschritte der logistischen Abbildung bei vier verschiedenen Werten des Parameters ($r = 2,90$; $r = 3,43$; $r = 3,53$ und $r = 4,00$), der die Steilheit der Parabel bestimmt

6.5.2 Mackey-Glass-Gleichung

Die Mackey-Glass-Gleichung (M-G) in [Mac77] ist ein Modell der Produktion von der weißen Blut-Zelle. Sie ist durch eine Differentialgleichung in der Form:

$$\frac{dx_t}{dt} = \frac{b \cdot x_{t-\Delta}}{1 + (x_{t-\Delta})^c} - a \cdot x_t \tag{6.12}$$

oder eine Abbildung in der Form:

$$x_{t+1} = x_t + \frac{b \cdot x_{t-\Delta}}{1 + (x_{t-\Delta})^c} - a \cdot x_t \tag{6.13}$$

gegeben, wobei die Konstanten $a = 0,1$ $b = 0,2$ und $c = 10$ oft eingesetzt sind. Der Verschiebungsparameter Δ bestimmt die charakteristische zeitliche Entwicklung des Modells.

$x(t)$ stellt die Konzentration der weißen Blutkörperchen in der Zeit t dar. $x_{t-\Delta}$ ist die Konzentration zu dem Zeitpunkt, zu dem der Wunsch nach mehr Blut auftritt. Bei der Person mit der Leukämie kann das Δ übermäßig groß sein und die Konzentration des Blutes wird oszillieren. Falls Δ sehr groß ist, verhält sich die Konzentration des Blutes chaotisch.

Abbildung 6.8 zeigt die Zeitreihen $\{x_t\}$ und die Trajektorie der M-G Gleichung bei $a = 0,1$ $b = 0,2$ $c = 10$ und $\Delta = 30$. Die Trajektorie ist in 3-Dimensionen, als (x_{t+1}-x_t-x_{t-30})-Graph, konstruiert. Die Charakteristik der M-G-Gleichung als Funktion

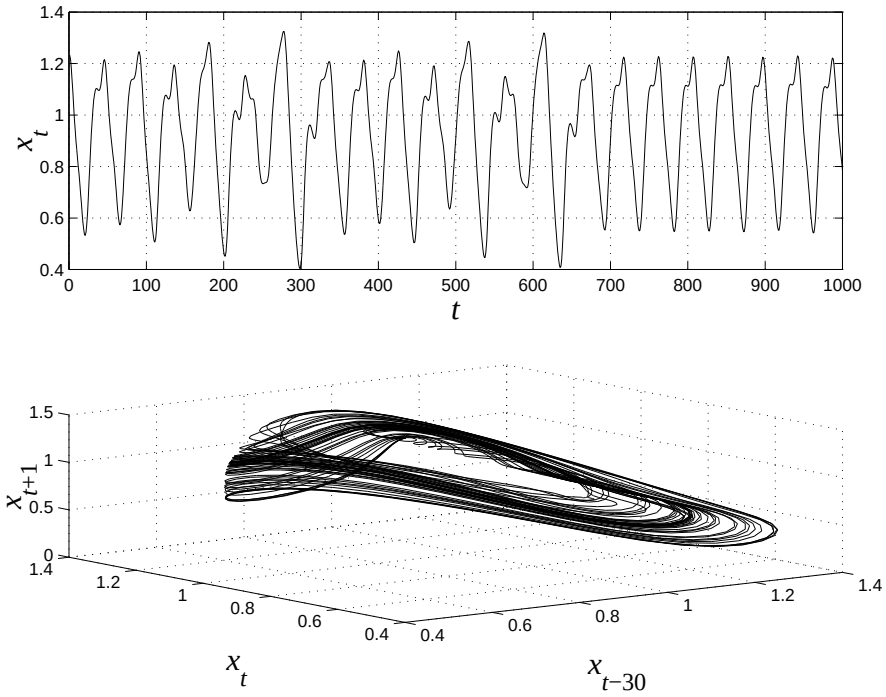


Abbildung 6.8: Zeitreihe $\{x_t\}$ (*oben*) und Trajektorie (*unten*) der M-G Gleichung bei $a = 0,1 \;\; b = 0,2 \;\; c = 10$ und $\Delta = 30$

von Δ mit $a = 0,1 \;\; b = 0,2$ und $c = 10$ wurde von J. D. Farmer untersucht, und in [Farm82] veröffentlicht.

- Für $\Delta < 4,53$ entsteht ein Fixpunktattraktor.
- Für $4.53 \leq \Delta < 13,3$, verhält sich die Gleichung wie ein instabiler Grenzzyklus-Attraktor.
- Wenn $\Delta = 13,3$ ist, verdoppelt sich die Periode des Grenzzyklus. Diese Periodenverdopplung setzt sich bis $\Delta = 16,8$ fort.
- Ab $\Delta > 16,8$ hat die Gleichung einen seltsamen Attraktor, deren Charakteristik von Δ abhängig ist.

Falls $\Delta = 30$ ist, hat der Attractor der M-G-Gleichung die fraktale Dimension $D_F > 2,94$ und zwei positive Lyaponov-Exponenten. Die Summe aller positiven Lyaponov-Exponenten beträgt ca. 0.01.

6.5.3 Sonnenflecken

Sonnenflecken sind die dunkel erscheinenden Flecken auf der Sonnenoberfläche, weil sie kühler sind als die sie umgebende Photosphäre. Sie besteht aus einem dunklen

Kern (*Umbra*), welcher von einem etwas helleren Hof (*Penumbra*) umgeben wird. Die Sonnenflecken entwickeln sich aus kleinen Einzelflecken (*Poren*) und bilden bei ihrer Ausbreitung Gruppen. Ihre Lebensdauer kann von wenigen Stunden bis zu mehreren Monaten reichen.

Normalerweise schützt das eigene Magnetfeld die Erde vor den meisten der von der Sonne ausgehenden Emissionen. Aber während Zeiten intensiver Sonnenfleckenaktivität können geomagnetische Stürme auftreten, eine verstärkte und höchst eindrucksvolle Erscheinung ist das Nordpolarlicht und das Südpolarlicht. Geomagnetische Stürme können auch Funkverbindungen unterbrechen und Stromversorgungsnetze stören. Energiereiches elektromagnetisches Bombardement kann die Übertragung von Radiowellen beeinflussen und auch den Stromfluss in elektrischen Leitungen beeinträchtigen. Funker kennen die Wirkungen der Sonnenfleckenaktivität und müssen sich auf eine Zunahme atmosphärischer Störungen auf die Funkwellen einstellen. Durch genau die gleichen Bombardements können auch Stromversorgungsnetze überlastet werden.

Die *Sonnenfleckenrelativzahl* R_s [Rod68] wurde zur Bewertung der Sonnenfleckenaktivität im letzten Jahrhundert vom Direktor des Züricher Observatoriums Rudolf Wolf eingeführt. Sie wird zu seinen Ehren auch als *Wolf-Zahl* bezeichnet. Sie ist wie folgt definiert:

$$R_\mathrm{s} = k_\mathrm{s} \cdot (10 \cdot g_\mathrm{s} + f_\mathrm{s}) \tag{6.14}$$

wobei:

- R_s die Sonnenfleckenrelativzahl,
- k_s der Korrekturfaktor, abhängig von atmosphärischen Bedingungen, Instrument, Beobachter, Höhe der Sonnenaktivität,
- 10 der Gewichtsfaktor für die Gruppenzahl,
- g_s die Gruppenzahl, die nach ihrer Komplexität mit Großbuchstaben A bis F und H klassifiziert wird
- f_s die Einzelfleckenzahl ist.

Dabei bedeutet Klasse A ein kleiner einzelner unipolarer Sonnenfleck oder eine sehr kleine Sonnenfleckengruppe ohne Halbschatten. F steht für eine ausgedehnte bipolare Sonnenfleckengruppe mit Halbschatten an beiden Enden mit mindestens 15 Winkelgraden umfassender longitudinaler Ausdehnung.

Bei der Beobachtung der Sonnenflecken sind weltweit viele Observatorien beteiligt, deshalb wird das erhaltene Ergebnis mit einem Korrekturfaktor k_s, der aber etwa 1 ist, multipliziert. Praktisch heißt das: Ist kein Fleck zu sehen, so ist R_s gleich Null. Ist ein Fleck zu sehen, so ist R_s gleich 11 (der Fleck ist zugleich eine Gruppe).

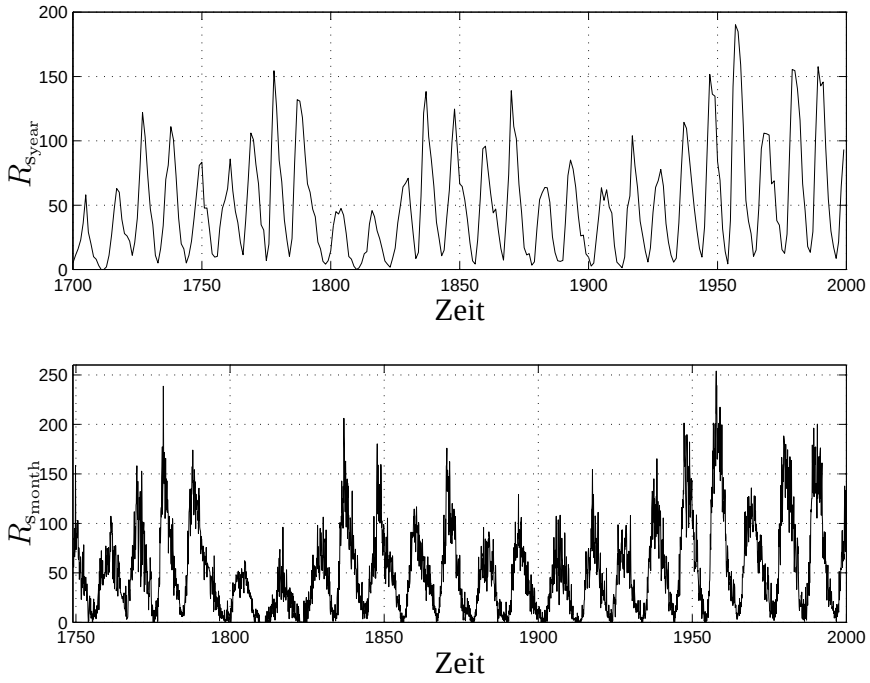


Abbildung 6.9: Jährliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ von 1700-1999 (*Oben*), monatliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ vom Oktober 1749 bis Dezember 2000 (*Unten*)

Die Abbildung 6.9 (oben) zeigt die jährliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ von 1700 bis 1999[3]. Die Sonnenfleckenrelativzahl steigt und fällt durchschnittlich alle 11,1 Jahre. Der Zyklus ist jedoch nicht symmetrisch. Die Sonnenfleckenrelativzahl steigt im Durchschnitt in ca. 4,8 Jahren von einem Minimum bis zu einem Maximum und fällt 6,2 Jahre zu einem anderen Minimum wieder ab. Die größte jährliche durchschnittliche Sonnenfleckenrelativzahl betrug 190,2 und wurde im Jahre 1957 beobachtet.

Die durchschnittliche monatliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ verläuft sehr ähnlich wie die durchschnittliche jährliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$, die jedoch von dem Rauschen überlagert wird. Die Abbildung 6.9 (unten) zeigt die durchschnittliche monatliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ von Oktober 1749 bis Dezember 2000. In [Mun91] wurde die monatliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ durch einen Tiefpass gefiltert, und anschließend die Lyaponov-Exponenten und die fraktale Dimension berechnet. Der eingesetzte Tiefpass ist ein digitales Butterworth-Filter zweiter Ordnung, dessen Grenzfrequenz $f_{\mathrm{c}} = 1/6$ Jahr^{-1} ist.

Nach der Berechnung beträgt einer der Lyaponov-Exponenten der gefilterten monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ ca. $0,02$ Monat^{-1}. Mit nur einem positiven Lyaponov-Exponent bedeutet es dann, dass diese monatliche

[3]Die Daten können in der Web-Site: ftp://ftp.ngdc.noaa.gov/STP/SOLAR_DATA/SUNSPOT_NUMBERS gefunden werden.

durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s}_{\mathrm{month}}}$ eine chaotische Zeitreihe ist. Die fraktale Dimension D_{F} wird dabei ca. $2,3$ geschätzt.

7 Einführung in die Prädiktion von deterministischen chaotischen Zeitreihen

Dieses Kapitel gibt eine Einführung in die Prädiktion der deterministischen chaotischen Zeitreihen. Im ersten Abschitt werden Begriffe wie Modellieren, Identifikation und Prädiktion definiert. Danach werden die Arten der Prädiktion erläutert. Zum Schluss stellen wir die verschiedenen Arten der Prädiktionverfahren vor.

7.1 Definition der Prädiktion

In diesem Abschnitt werden die Begriffe *Modellieren*, *Identifikation* und *Prädiktion* definiert. Diese drei Begriffe sind bereits in der Literatur [Pham99] erwähnt, nämlich:

Definition 7.1 *Das Modellieren (modelling) ist das Bilden eines mathematischen Modells eines Systems durch die Anwendung der physikalischen Gesetze, die das Verhalten des Systems bestimmen.*

Definition 7.2 *Identifikation (identification) ist das Bilden eines mathematischen Modells von einem System, indem man*

- *Eingangs- und Ausgangsignale eines bestimmten Systems beobachtet,*
- *und anschließend ein Modell sucht, das das System am besten beschreibt.*

Definition 7.3 *Prädiktion (prediction) ist eine Verquickung der vorherigen verfügbaren Daten bis zum Zeitpunkt t zur Vorhersage von zukünftigen Werten zu verschiedenen Zeitpunkten $t+1, t+2, t+3, \ldots$.*

Obwohl die Begriffe des Modellierens, der Identifikation und der Prädiktion unterschiedlich sind, kann es sein, dass ihre mathematischen Modelle identisch sind.

Unterscheidet man das Modellieren, die Identifikation und die Prädiktion aus dem Vorwissen, hat man beim Modellieren die meiste Information über das System. Bei der Identifikation kennt man Eingangs- und Ausgangsignale. Der schwierigste Fall ist die Prädiktion, da man nur vorherige Daten kennt. Zur guten Langzeitprädiktion werden aber noch mehr Informationen benötigt. Dabei ist die mathematische Funktion, die sich aus dem Modellieren ergibt, für die Langszeitprädiktion sehr gut geeignet. Das Modellieren kann folglich bei der Prädiktion umgesetzt werden.

7.2 Arten der Prädiktion

Die Prädiktion lässt sich in die folgenden Arten gliedern ([Weig91]).

Einschrittprädiktion

Einschrittprädiktion (*single-step prediction*) ist die Prädiktion eines zukünftigen Wertes $\hat{x}_{t+1}$ mit Hilfe von vorhergegangenen Werten $x_t, x_{t-1}, x_{t-2}, ..., x_{t-m}$, wie in der folgenden Gleichung:

$$\hat{x}_{t+1} = \hat{f}(x_t, x_{t-1}, x_{t-2}, ..., x_{t-m}) \quad ; m < t \tag{7.1}$$

Wobei $\hat{f}$ die Prädiktionsfunktion zur Prädiktion des nächsten zukünftigen Wertes ist.

Da die Einschrittprädiktion nur einen nächsten Wert prädiktiert, kann man sie auch als Kurzzeitprädiktion bezeichnen.

Mehrschrittprädiktion

Bei der Mehrschrittprädiktion (*multi-step prediction*) werden mehrere zukünftige Werte $\hat{x}_{t+1}, \hat{x}_{t+2}, \hat{x}_{t+3}, ...$ prädiktiert. Deshalb kann man die Mehrschrittprädiktion auch als Langzeitprädiktion bezeichnen. Es gibt zwei Arten der Mehrschrittprädiktion:

1. Mehrschrittprädiktion durch die **iterative Einschrittprädiktion (*iterated single-step*):** Der durch die Einschrittprädiktion prädiktierte Wert wird als Anfangswert für den nächsten Prädiktionsschritt verwendet.

$$\begin{aligned}
\hat{x}_{t+1} &= \hat{f}(x_t, x_{t-1}, ..., x_{t-m}) \quad ; m < t \\
\hat{x}_{t+2} &= \hat{f}(\hat{x}_{t+1}, x_t, ..., x_{t-m+1}) \quad ; m < t \\
\hat{x}_{t+3} &= \hat{f}(\hat{x}_{t+2}, \hat{x}_{t+1}, ..., x_{t-m+2}) \quad ; m < t \\
&\vdots \\
\hat{x}_{t+k} &= \hat{f}(\hat{x}_{t+k-1}, \hat{x}_{t+k-2}, ..., x_{t-m+k-1}) \quad ; m < t
\end{aligned} \tag{7.2}$$

2. **Direkte Mehrschrittprädiktion (*direct multi-step prediction*):** Der Wert vorwärts wird direkt prädiktiert. Dies geschieht wie es die folgende Gleichung zeigt:

$$\hat{x}_{t+k} = \hat{f}_k(x_t, x_{t-1}, ..., x_{t-m}) \quad ; m < t \quad \text{und} \quad k > 1 \tag{7.3}$$

   Wobei $\hat{f}_k$ die Prädiktionsfunktion zur Prädiktion des k. zukünftigen Wertes ist.

Wenn der erste Anfangswert der Prädiktionsfunktion bei der iterativen Einschrittprädiktion x_{t-m} ist, werden nach m-maliger iterativer Einschrittprädiktion keine guten prädiktierten Werte mehr erwartet. Die prädiktierten Werte werden möglicherweise periodisch sein. Dieses Problem könnte aber die direkte Mehrschrittprädiktion beheben. Durch die Kombination der iterativen Einschrittprädiktion und der direkten Mehrschrittprädiktion lässt sich die Langzeitprädiktion verbessern.

Weitere Arten der Prädiktion

In [Smith94] werden zwei weitere Arten der Prädiktion definiert, nämlich *Runaway* und *Update Extension*.

- ***Runaway*:** Bei *Runaway* werden zuerst Anfangswerte in der Prädiktionsfunktion vorgegeben. Man versucht danach mit dieser Prädiktionsfunktion so weit wie möglich iterativ zu prädiktieren, indem prädiktierte Werte als Anfangswerte für den nächsten Prädiktionsschritt verwendet werden. Die iterative Einschrittprädiktion ist daher eine Art von *Runaway*.
- ***Update Extension*:** Bei *Update Extension* werden dagegen in jedem Prädiktionsschritt die tatsächlichen Werte verwendet. Die Einschrittprädiktion und die direkte Mehrschrittprädiktion entsprechen daher einer Art von *Update Extension*.

7.3 Prädiktionsverfahren

Zur Prädiktion von deterministischen chaotischen Zeitreihen wird eine nichtlineare Funktion benötigt. Für die Langzeitprädiktion ist die mathematische Funktion aus dem Modellieren sehr geeignet. Einführungen in die Verfahren zum Modellieren oder zur Prädiktion von deterministischen chaotischen Zeitreihen sind bereits in der Literatur veröffentlicht, z.B. [Cas89, Cas92, Tso92, Ger93, Kan97]. Um den Umfang der Arbeit nicht unnötig zu vergrößern, werden in diesem Abschnitt nur einige wichtige Verfahren zur Entwicklung der Prädiktionsfunktion kurz erläutert. Weitere Prädiktionsverfahren findet man z.B. in [Nak00, Lan01, Mat01, Nak01].

7.3.1 FIR- und IIR-System

FIR-System (Finite Impulse Response) Die allgemeine Differenzengleichung eines FIR-Systems lautet:

$$x_t = \sum_{n=0}^{N} b_n e_{t-n} = b_0 e_t + b_1 e_{t-1} + ... + b_N e_{t-N} \tag{7.4}$$

Wobei:

- x_t das Ausgangssignal des Systems zur Zeit t,
- e_t das Eingangssignal des Systems zur Zeit t ist, und
- $b_0, b_1, ..., b_N$ die zu optimierenden Koeffizienten des FIR-Systems sind.

IIR-System (Infinite Impulse Response) Die allgemeine Differenzengleichung eines IIR-Systems lautet:

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + \sum_{n=0}^{N} b_n e_{t-n} \tag{7.5}$$

Wobei:

- x_t das Ausgangssignal des Systems zur Zeit t,
- e_t das Eingangssignal des Systems zur Zeit t ist, und
- a_m und b_n die zu optimierenden Koeffizienten des IIR-Systems sind.

Die FIR- und IIR-Systeme beherrschen den Bereich der Zeitreihenanalyse und digitalen Verarbeitung seit mehr als einem halben Jahrhundert. Sie sind besonders bei der Prädiktion eines linearen Systems sehr gut geeignet, versagen jedoch bei der Prädiktion eines deterministischen Chaos, da das deterministische Chaos ein nichtlineares System ist.

7.3.2 Globale und lokale Annäherung

Im Allgemeinen lässt sich das globale und lokale Annäherungsverfahren in zwei Schritten gliedern:

1. Rekonstruktion eines Phasenraums
2. Annäherung einer nichtlinearen Funktion

Abbildung 7.1 zeigt die Funktionsweise des Verfahrens. Dies kann folgendermaßen erläutert werden:

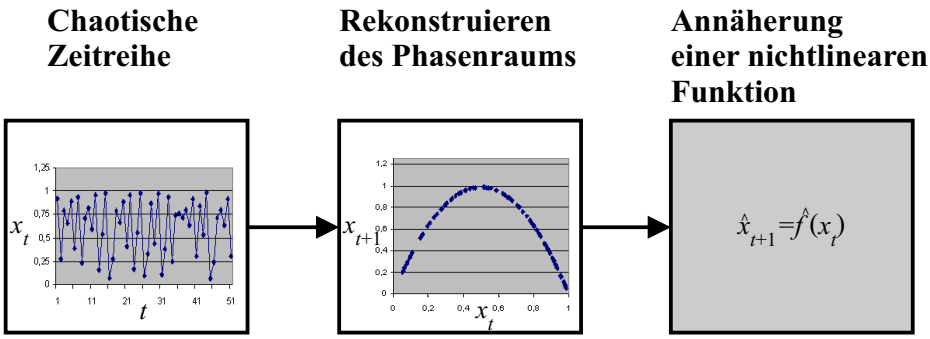


Abbildung 7.1: Funktionsweise der globalen und lokalen Annäherung

Rekonstruktion eines Phasenraums

Die Aufgabe dieses Schrittes ist das Rekonstruieren eines Phasenraums aus den gesammelten Daten einer Zeitreihe. Zuerst wird die Einbettungsdimension D_{E} (*embedding dimension*) und der Abstand zwischen den Komponenten der Hyperpunkte in der Originalzeitreihe τ geschätzt.

Unter Einbettungsdimension D_{E} versteht man die kleinste Anzahl von unabhängigen geometrischen Koordinaten im Phasenraum, die für die Bestimmung des Verhaltens des Attraktors benötigt ist. Die Einbettungsdimension D_{E} muss größer sein als die Attraktordimension, um den Attraktor überhaupt rekonstruieren zu können. Im normalen Fall ist D_{E} gleich oder größer als $2d_{\mathrm{A}} + 1$, wobei d_{A} die Dimension des Attraktors einer Zeitreihe ist.

Um einen günstigen Phasenraum zu bekommen, soll τ nicht zu groß oder zu klein sein. Bei einem sehr kleinen τ werden die abgetasteten Werte $x_t, x_{t-\tau}, x_{t-2\tau}, ...$ etwa den gleichen Wert besitzen und somit dazu führen, dass ein Attraktor entlang der Diagonalen gestreckt wird. Dagegen werden die abgetasteten Werte der Zeitreihe bei einem sehr großen τ stark nichtkontinuierlich, so dass die Trajektorie des rekonstruierten Phasenraumes extrem komplex ist.

Mit gewähltem τ und D_{E} wird ein Delay-Vektor $\boldsymbol{x}_t$ mit der Dimension D_{E} konstruiert. Der Delay-Vektor $\boldsymbol{x}_t$ hat dann die Form:

$$\boldsymbol{x}_t = \begin{bmatrix} x_t \\ x_{t-\tau} \\ x_{t-2\tau} \\ \vdots \\ x_{t-(D_{\mathrm{E}}-1)\tau} \end{bmatrix} \tag{7.6}$$

Damit kann man den Phasenraum rekonstruieren. Wenn es möglich ist, soll der Störungsanteil der Zeitreihe so klein wie möglich werden.

Annäherung einer nichtlinearen Funktion

Im diesem Schritt wird eine nichtlineare Funktion $\hat{f}$ gesucht, die zukünftige Werte $x_{t+\tau}$ aus $x_t, x_{t-\tau}, x_{t-2\tau}, ..., x_{t-(D_E-1)\tau}$ gemäß der folgenden Gleichung abbilden kann.

$$\hat{x}_{t+\tau} = \hat{f}(\boldsymbol{x}_t) = \hat{f}(x_t, x_{t-\tau}, x_{t-2\tau}, ..., x_{t-(D_\mathrm{E}-1)\tau}) \tag{7.7}$$

Eine Annäherung der nichtlinearen Funktion $\hat{f}$ kann gefunden werden, indem

- man zunächst eine nichtlineare Funktion $\hat{f}$ wählt, und
- dann die Koeffizienten der gewählten nichtlinearen Funktion $\hat{f}$ mit den Daten im Phasenraum optimiert.

Man kann die Annäherung der nichtlinearen Funktion in zwei Arten unterteilen:

Globale Funktionsannäherung: Bei der globalen Funktionsannäherung wird eine nichtlineare Funktion herausgesucht, die den gesamten Verlauf im Phasenraum beschreiben kann. Die folgenden nichtlinearen Funktionen können angewendet werden:

- **Ganze rationale Funktion** oder **Polynome (*Polynomials*):** Wenn wir annehmen, dass die Einbettungsdimension $D_\mathrm{E} = 3$ und die gesuchte nichtlineare Funktion Polynome 2. Grades sind, kann die gesuchte nichtlineare Funktion folgendermaßen dargestellt werden:

$$\begin{aligned} \hat{x}_{t+\tau} \;=\; & a_0 + a_1 \cdot x_t + a_2 \cdot {x_t}^2 + \\ & a_3 \cdot x_{t-\tau} + a_4 \cdot {x_{t-\tau}}^2 + \\ & a_5 \cdot x_{t-2\tau} + a_6 \cdot {x_{t-2\tau}}^2 + \\ & a_7 \cdot x_t \cdot x_{t-\tau} + a_8 \cdot x_t \cdot x_{t-2\tau} + \\ & a_9 \cdot x_{t-\tau} \cdot x_{t-2\tau} \end{aligned} \tag{7.8}$$

  Die Koeffizienten a_0, a_1, a_2, ..., a_9 können durch das Least-squares-Verfahren optimiert werden, indem die Summe des Quadrates der Differenz zwischen dem tatsächlichen Wert $x_{t+\tau}$ und dem prädiktierten Wert $\hat{x}_{t+\tau}$ minimiert wird.

  Das Polynom ist ungeeignet für die Prädiktion von Daten, die außerhalb der Trainingsdaten liegen, und somit auch für die iterative Einschrittprädiktion. Außerdem wird die Anzahl der zu optimierenden Koeffizienten bei Erhöhung von D_E und dem Grad des Polynoms explodieren.

- **Gebrochene rationale Funktion (*Rational Polynomials*):** Sie ermöglicht eine weitere Verbesserung des Polynoms durch Anwendung von zwei Polynomen mit dem gleichen Grad.

- **Radiale Basisfunktion (*Radial Basis Function*):** Sie ist in der folgenden Form dargestellt:

$$\hat{x}_{t+\tau} = \hat{f}(\boldsymbol{x}_t) = \sum_{\acute{t}=1}^{N} a_{\acute{t}} \cdot \phi(\|\boldsymbol{x}_t - \boldsymbol{x}_{\acute{t}}\|) \tag{7.9}$$

  Wobei:

  - $\boldsymbol{x}_{\acute{t}}$, mit $\acute{t} < t$, repräsentiert einen Zentralpunkt im Phasenraum. Zentralpunkte $\boldsymbol{x}_{\acute{t}}$ sollen gut im Phasenraum verteilt sein.
  - $\|\boldsymbol{x}_t - \boldsymbol{x}_{\acute{t}}\|$ ist der euklidische Abstand zwischen $\boldsymbol{x}_t$ und $\boldsymbol{x}_{\acute{t}}$.
  - $\phi(r)$ ist eine radiale Basisfunktion, z.B. die gauß'sche Funktion. Die typische radiale Basisfunktion ist immer glockenförmig, ihr maximaler Wert liegt bei $r = 0$.

  Die Koeffizienten a_0, a_1, a_2, usw. können gut durch das Least-squares-Verfahren optimiert werden. Die radiale Basisfunktion ist nur vom euklidischen Abstand zwischen den verschiedenen Punkten abhängig.

Die Globale Funktionsannäherung ist geeignet, wenn die deterministische chaotische Zeitreihe nicht sehr komplex und die Einbettungsdimension D_{E} nicht sehr hoch ist. Andere Möglichkeiten der globalen Funktionsannäherung sind z.B. künstliche neuronale Netze oder die GP.

Lokale Funktionsannäherung: Im Gegensatz zu der globalen Funktionsannäherung wird bei der lokalen Funktionsannäherung eine nichtlineare Funktion, die den Verlauf innerhalb eines bestimmten Bereichs im Phasenraum beschreiben kann, herausgesucht.

Eine lokale Funktionsannäherung ist z.B. das *Local Fitting*. Dies funktioniert folgendermaßen:

- Wenn man einen zukünftigen Wert $x_{t+\tau}$ prädiktieren will, werden zuerst p nächste Nachbarn von x_t im Phasenraum bestimmt. Der nächste Nachbar von x_t sei der Wert $x_{\acute{t}}$, der innerhalb eines bestimmten Abstands von x_t liegt, wobei $\acute{t} < t$ ist.

- Mit den p nächsten Nachbarn $x_{\acute{t}}$ wird eine nichtlineare Funktion bestimmt, die die zukünftigen Werte der p nächsten Nachbarn $x_{\acute{t}+\tau}$ prädiktieren kann. Hier kann man alle nichtlinearen Funktionen in der globalen Funktionsannäherung verwenden.

- Mit Hilfe der gefundenen nichtlinearen Funktion wird $x_{t+\tau}$ prädiktiert.

Die Anzahl der nächsten Nachbarn soll nicht zu groß gewählt werden. Bei einer zu großen Anzahl der nächsten Nachbarn können ungeeignete Werte in der nichtlinearen Funktion berücksichtigt werden, daher wird der Prädiktionsfehler außerhalb der Trainingsdaten vergrößert.

Es ist ersichtlich, dass die nichtlineare Funktion aus der lokalen Funktionsannäherung kleinere Einschrittprädiktionsfehler besitzt als die aus der globalen Funktionsannäherung. Jedoch ist die nichtlineare Funktion $\hat{f}$ aus der lokalen Funktionsannäherung meistens nichtkontinuierlich, sie ist daher für die iterative Einschrittprädiktion ungeeignet.

7.3.3 Künstliche neuronale Netze

Künstliche Neuronale Netze (KNN) sind biologisch motivierte, im weitesten Sinne naturanaloge Systeme. Sie modellieren, auf stark vereinfachte Weise, Organisationsprinzipien und Abläufe in natürlichen Neuronalen Netzen. Jedes KNN besteht aus einer Anzahl künstlicher Neuronen, die als elementare Informationsverarbeitungseinheiten in bestimmter Weise angeordnet und untereinander verbunden sind (Netzarchitektur). Die Verbindungen dienen zum Austauschen von Nachrichten und sind gewichtet, was die Intensität des Informationsflusses entlang dieser Verbindung zum Ausdruck bringen soll. Über Lernregeln können die Verbindungsgewichte variiert werden. Dabei besteht das Ziel darin, in der Trainingsphase, anhand von Beispielen und mit Hilfe einer Lernregel, Abbildungen von Eingaben auf erwünschte Ausgaben des Netzes zu lernen. Das so Gelernte soll dann im praktischen Einsatz auf zuvor nicht gesehenes ähnliches Datenmaterial generalisiert werden können.

Die Neuronen eines KNN verarbeiten Eingabesignale, aus denen sie, unter Berücksichtigung ihres aktuellen Zustandes (ihrer Aktivierung, gegeben durch eine reelle Zahl), ihren neuen Zustand berechnen und eine Ausgabe generieren. Die Neuronen verarbeiten dabei ihre Eingaben vollständig parallel und unabhängig voneinander. Dabei dienen die Eingabe- bzw. Ausgabeneuronen zur Kommunikation mit der Umwelt, während Zustand und Ausgabe der inneren (oder verdeckten) Neuronen von außen nicht zu erkennen sind.

Eine beliebte Klasse von KNN sind die *Mehrschichten-Perceptrons*. Sie bestehen aus einer Eingabe- und einer Ausgabeschicht sowie einer oder mehreren verdeckten Schichten. Nur die Einheiten zweier aufeinanderfolgender Schichten sind miteinander verbunden. Es bestehen keine Verbindungen zwischen Neuronen innerhalb derselben Schicht oder über mehrere Schichten hinweg. Die Verbindungen durch das Netz laufen gerichtet von der Eingabe- zur Ausgabeschicht. Mehrschichten-Perceptrons gehören damit zur Klasse der vorwärtsgekoppelten (*feedforward*) KNN.

Als Lernregel verwendet man für Mehrschichten-Perceptrons im allgemein das sogenannte *(Error)-Backpropagation-Verfahren*. Hierbei wird dem Netzwerk während

der Trainingsphase außer den Eingaben zu jedem Beispiel auch der gesuchte Ausgabevektor geboten. Die Summe der quadrierten Abweichungen zwischen der tatsächlichen und der gewünschten Ausgabe des KNN bildet ein Fehlermaß, welches nun rückwärts von der Ausgabeschicht in Richtung der Eingabeschicht durch das Netz propagiert wird. Es bildet die Grundlage für die Aktualisierung der Verbindungsgewichte im Netz.

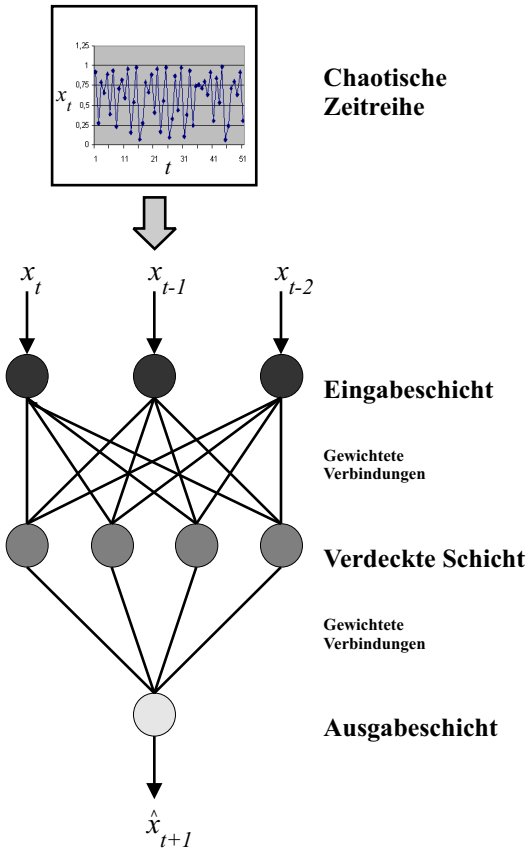


Abbildung 7.2: Aus dem aktuellen und zwei zurückliegenden Werten x_t, x_{t-1}, x_{t-2} soll der zukünftige Wert x_{t+1} mit Hilfe von KNN prädiktiert werden.

Ein allgemeines Problem von Mehrschichten-Perceptrons besteht in der richtigen Wahl von Anzahl und Größe (gemessen in der Zahl der Neuronen) der verdeckten Schichten. Wählt man eine zu geringe Anzahl innerer Neuronen, so ist das Netz nicht in der Lage, die Lernaufgabe zu lösen. Bei zu vielen inneren Neuronen kann das Overfittingproblem auftreten. Das Netz lernt dann zwar die Trainingsbeispiele, erzeugt bei neuen Eingabemustern aber unerwünschte Ausgaben.

In Abbildung 7.2 ist schematisch die Prädiktion einer Zeitreihe mit Hilfe eines vorwärtsgekoppelten KNN dargestellt. Die vergangenen Werte werden dem Netz von außen zur Verfügung gestellt. KNN ist flexibler als andere vorher erwähnte Verfahren. Ein Nachteil von KNN ist der höhere Zeitaufwand während des Trainings als bei anderen bereits erwähnten Prädiktionsverfahren.

Beispiele von KNN bei der Prädiktion einer Zeitreihe sind z.B. in [Goh00, Wak00, Wak01, Wan01] veröffentlicht.

7.3.4 Genetische Programmierung

Zur Entwicklung der Prädiktionsfunktion kann auch die GP eingesetzt werden. Durch das Training mit den Trainingsdaten wird die GP eine Prädiktionsfunktion entwickeln. GP ist ein interessantes Verfahren zur Funktionsannäherung. Da muss man keine nichtlineare Funktion selbst wählen. GP übernimmt diese Aufgabe. Man benötigt daher keine hohen mathematischen Kenntnisse. Vorkenntnis ist jedoch sehr hilfreich. Außerdem kann die durch GP entwickelte Prädiktionsfunktion wichtige Informationen über die chaotische Zeitreihe beinhalten. Die GP ist sehr flexibel. Ein Nachteil von GP ist der große Zeitaufwand während des Trainings.

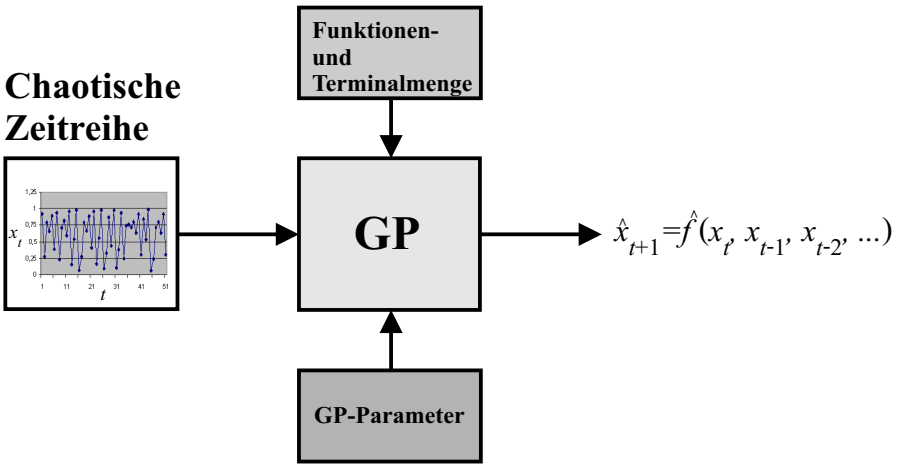


Abbildung 7.3: Einfaches Prädiktionsverfahren mit GP

Die Anwendung von GP im Bereich der Prädiktion, Modellieren oder Identifikation ist bereits in verschiedener Literatur veröffentlicht. In [Koza92] wurde versucht, die logistische Abbildung zu modellieren. Abbildung 7.3 zeigt das einfache Prädiktionsverfahren mit GP. Weitere Beispiele sind in [Iba94, Oak94, Kim96, Mul96, Mau98, She01, Vaz01] veröffentlicht.

Zur Verbesserung der Effizienz des Verfahrens wird einerseits GP mit anderen Konstantenoptimierungsmethoden, z.B. Least-squares-Verfahren, Simulated Annealing, GA usw., kombiniert. Diese Kombination funktioniert folgendermaßen:

- GP entwickelt zunächst Strukturen der gesuchten Lösung.
- Danach entwickelt die Konstantenoptimierungsmethode die numerischen Parameter der gesuchten Lösung.

Diese Idee ist z.B. in [Shar95, Gray96, Gray97, Rod97, Rod99, Yos99, Cao00, Yos00, Whig01] zu finden.

Da die Kombinationsverfahren zeitaufwendiger als eine einfache GP sind, wurde in [South96] eine neue Fitnessfunktion vorgeschlagen, die *Correlation Fitness Function* genannt wird. Damit müssen Individuen nicht vor jeder Fitnessberechnung während des GP-Laufes optimiert werden. Diese Fitnessfunktion ist jedoch für das deterministische Chaos ungeeignet.

Andererseits verwendet man die GP, um z.B. Strukturen der Schichten bei KNN oder Polynome des komplexen Systems zu entwickeln. Diese Idee ist z.B. in [Ang98, Zha99, Nik00, Nik01, Tan01] zu finden.

In [Lee99, Lee01] wurde GP rekursiv eingesetzt, um die richtige Prädiktion anzunähern, indem GP Fehler der vorher gefundenen Prädiktion zu minimieren versucht.

In [Kab01] wurde GP [Koza92] mit KNN bei der Einschrittprädiktion des monatlichen Rohölpreises von Januar 1993 bis Dezember 1999 in der vereinigten Staaten von Amerika verglichen. Die in dem Experiment verwendete KNN ist die Mehrschichten-Perceptrons mit einer verdeckten Schicht. Das Ergebnis deutet an, dass GP sowohl die Daten innerhalb als die Daten außerhalb der Trainingsdaten besser als KNN prädiktiert.

In dieser Arbeit wird ein Prädiktionsverfahren vorgeschlagen, das auf der Kombination GP mit anderen Konstantenoptimierungsmethoden basiert. Das Prädiktionsverfahren arbeitet rekursiv. Dadurch bekommt man in jeder Rekursionsstufe eine Sub-Prädiktionsfunktion, die einen Teil der richtigen Prädiktionsfunktion darstellt. Am Ende des Prädiktionsprozesses bilden alle Sub-Prädiktionsfunktionen gemeinsam die Prädiktionsfunktion. Im Kapitel 8 wird unser Prädiktionsverfahren detailliert erläutert.

7.4 Zusammenfassung

Die Begriffe des Modellierens, der Identifikation und der Prädiktion sind unterschiedlich definiert. Jedoch können die mathematischen Modelle identisch sein. Die Prädiktion lässt sich in zwei Arten unterscheiden, nämlich Einschritt- und Mehrschrittprädiktion.

Zur Prädiktion der determistischen chaotischen Zeitreihen stehen mehrere Verfahren zur Verfügung. Bei der globalen und lokalen Annährung wird zunächst ein Phasenraum rekonstruiert. Danach sucht man eine nichtlineare Funktion, die den Verlauf im Phasenraum beschreiben kann. Der Unterschied zwischen der globalen und lokalen Annährung besteht darin, dass die nichtlineare Funktion der globalen Annährung den gesamten Verlauf im Phasenraum beschreibt, dagegen die nichtlineare Funktion der lokalen Annährung nur einen bestimmten Verlauf im Phasenraum.

KNN und GP sind die Natur nachahmende Verfahren, die sehr flexibel aber sehr zeitaufwendig sind. Durch das Lernen von Trainingdaten werden die KNN und die GP die gesuchte Lösung entwickeln. Bei KNN muss vorher die Struktur des Netzes ausgewählt werden. Bei GP hat man noch mehr Freiheiten, da die GP die Struktur der Lösung selbst sucht. Außerdem kann die durch GP entwickelte Prädiktionsfunktion wichtige Informationen über die chaotische Zeitreihe beinhalten. Durch die Kombination mit anderen Optimierungsverfahen wird sich die Effizienz der GP verbessern.

8 Prädiktion der deterministischen chaotischen Zeitreihe mit genetischer Programmierung

Dieses Kapitel, der Hauptabschnitt dieser Arbeit, behandelt Prädiktionsverfahren der deterministischen chaotischen Zeitreihe mit GP. Im ersten Abschnitt werden zwei Prädiktionsverfahren vorgestellt, die auf GP basieren, nämlich ein *einfaches Prädiktionsverfahren* und ein *rekursives Prädiktionsverfahren*. Danach wird ein Problem bei der Prädiktion erläutert, das sogenannte *Overfitting*. Im letzten Abschnitt beschäftigen wir uns mit der Funktion zur Berechnung und dem Verlauf des Prädiktionsfehlers.

8.1 Prädiktionsverfahren

Um das Prädiktionsverfahren leicht zu verstehen, werden zuerst das einfache Prädiktionsverfahren und anschließend das rekursive Prädiktionsverfahren erläutert.

Im normalen Fall entwickeln die beide Prädiktionsverfahren eine Prädiktionsfunktion für die Einschrittprädiktion durch das Lernen mit der chaotischen Zeitreihe. Zur Prädiktion von mehreren zukünftigen Werten (Mehrschrittprädiktion) wird diese Prädiktionsfunktion mit der Prädiktionsart der iterative Einschrittprädiktion verwendet.

Es ist möglich, durch beide Prädiktionsverfahren eine Prädiktionsfunktion für die direkte Mehrschrittprädiktion zu entwickeln. Dies benötigt jedoch beträchtlich mehr Zeit bei der Entwicklung, da die Prädiktionsfunktion für die direkte Mehrschrittprädiktion komplexer als die Prädiktionsfunktion für die Einschrittprädiktion ist.

8.1.1 Einfaches Prädiktionsverfahren

Das *einfache Prädiktionsverfahren* (oder das *einfache Verfahren*) besteht aus der Kombination von GP und dem Konstantenoptimierungsverfahren.

Die Grundidee dabei ist:

- Durch das Lernen mit den Trainingsdaten aus einer deterministischen chaotischen Zeitreihe entwickelt oder sucht GP Strukturen der nichtlinearen Funktion, die die Zeitreihe prädiktieren kann.

- Anschließend werden alle Konstanten und Koeffizienten der durch GP entwickelten oder gefundenen nichtlinearen Funktion optimiert, um ihre Effizienz zu erhöhen.

Wenn man bei jedem Individuum vor der Fitnessberechnung die Konstantenoptimierung durchführt, wird dieses Verfahren extrem zeitaufwendig. Zur Reduzierung des Zeitaufwandes setzt man den Vereinfachungszwang ein, d.h. die Teife des Individuums (Programmlänge) wird in dem Fitnesswert berücksichtigt, damit die Anzahl der zu optimierenden Konstanten und Koeffizienten reduziert wird.

Der Vorteil des Vereinfachungszwangs kann jedoch heute noch nicht völlständig zusammengefasst werden. Bei dem *Introns* ([Ban98]) hat der Vereinfachungszwang Nachteile. Unter Intron versteht man ein Merkmal der Erbinformation (*Genotyp*), das durch den Evolutionsprozess von der variablen Strukturlänge entsteht. Das Intron beeinflusst nicht direkt die Überlebensfähigkeit des Individuums in GP. Das Intron schützt jedoch die wichtigen Teile der guten Eltern vor der Zerstörung durch genetische Operatoren, wie z.B. Mutation oder Kreuzung, damit Nachkommen diese wichtigen Teile erben und bessere Fitnesswerte als ihre Eltern besitzen können. Mit dem Vereinfachungszwang wird das Intron reduziert, folglich kann GP gute Nachkommen nur schwer produzieren. Der GP-Lauf stagniert also schnell.

Eine andere Möglichkeit zur Reduzierung des Zeitaufwands ist, die Konstantenoptimierung nur am Ende des GP-Laufs durchzuführen, d.h. man optimiert nur das beste Individuum aus dem GP-Lauf. Für einfache Aufgaben ist dies völlig ausreichend. Diese Grundidee wird daher in dieser Arbeit verwendet.

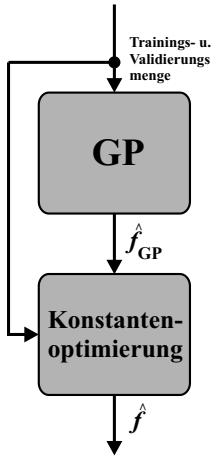


Abbildung 8.1: Das einfache Prädiktionsverfahren mit GP

Funktionsweise des einfachen Prädiktionsverfahrens

Abbildung 8.1 zeigt den Ablauf des einfachen Prädiktionsverfahrens mit GP. Das einfache Prädiktionsverfahren ist in zwei Teile gegliedert, und kann folgendermaßen erläutert werden.

1. Teil: Suche der Prädiktionsfunktion durch GP In diesem Teil lernt die GP mit den Trainingsdaten, um möglichst die beste Prädiktionsfunktion $\hat{f}_{\mathrm{GP}}$ zu entwickeln oder zu finden. Die GP-Parameter werden wie gewöhnlich eingestellt.

Die Funktionenmenge $\mathcal{F}$ kann z.B. die folgenden mathematischen Funktionen enthalten:

sin-Funktion	cos-Funktion
log-Funktion	exp-Funktion
log-Funktion	exp-Funktion
$+$ (Addition)	$-$ (Subtrahieren)
$\times$ (Multiplikation)	$\div$ (Dividieren)

Elemente der Terminalmenge $\mathcal{T}$ sind z.B.:

- Variablen in verschiedenen Zeitpunkten, z.B. $x_t, x_{t-1}, x_{t-2}, ...$ und
- einfache konstante Werte, z.B. im Bereich von -10 bis 10.

2. Teil: Konstantenoptimierung Nachdem die Prädiktionsfunktion durch die GP gefunden ist, werden Koeffizienten in die Prädiktionsfunktion $\hat{f}_{\mathrm{GP}}$ eingefügt. Anschließend werden die Koeffizienten und einfache konstante Werte optimiert.

Die Koeffizienten können in zwei verschiedenen Möglichkeiten eingefügt werden.

- Bei mathematischen Funktionen

Mathematische Funktionen	Erste Möglichkeit	Zweite Möglichkeit
sin-Funktion: $\sin(x)$	$a \cdot \sin(x)$	$a \cdot \sin(x) + b$
cos-Funktion: $\cos(x)$	$a \cdot \cos(x)$	$a \cdot \cos(x) + b$
log-Funktion: $\log(x)$	$a \cdot \log(x)$	$a \cdot \log(x) + b$
exp-Funktion: $\exp(x)$	$a \cdot \exp(x)$	$a \cdot \exp(x) + b$

- Bei Variablen

Variable	Erste Möglichkeit	Zweite Möglichkeit
x_t	$a \cdot x_t$	$a \cdot x_t + b$
x_{t-1}	$a \cdot x_{t-1}$	$a \cdot x_{t-1} + b$
x_{t-2}	$a \cdot x_{t-2}$	$a \cdot x_{t-2} + b$
$\vdots$	$\vdots$	$\vdots$
x_{t-m}	$a \cdot x_{t-m}$	$a \cdot x_{t-m} + b$

Wobei a und b die eingefügten Koeffizienten sind. Mit der zweiten Möglichkeit der Einfügung von Koeffizienten wird die Anzahl der zu optimierenden Koeffizienten doppelt so groß wie bei der ersten.

Falls die durch GP entwickelte Prädiktionsfunktion lang und komplex ist, soll man die erste Möglichkeit der Einfügung von Koeffizienten verwenden. Ansonsten wird die Anzahl der zu optimierenden Konstanten und Koeffizienten sehr hoch, so dass die Optimierung keine gute Werte finden kann.

Es existieren derzeit mehrere Optimierungsmethoden. Die bekanntesten Methoden sind z.B. Least-squares-Verfahren, Simulated Annealing ([Kinne94]), Treshold Accepting ([Kinne94]), die Sintflut-Methode ([Kinne94]), Evolutionsstrategien und Genetische Algorithmen. In dieser Arbeit werden jedoch nur zwei Optimierungsmethoden verwendet:

- **Least-squares-Verfahren:** Eine einfache Optimierungsmethode ist *das Least-squares-Verfahren*. Das Least-squares-Verfahren ist ein gradientenorientiertes Opitmierungsverfahren. Es findet die optimalen Konstanten, indem die Summe des Quadrates der Differenz zwischen dem tatsächlichen Wert und dem prädiktierten Wert minimiert wird. Das Least-squares-Verfahren arbeitet sehr schnell, jedoch kann es das lokale Optimum nicht überspringen.

  Das Least-squares-Verfahren wird verwendet, wenn

  - die Funktionenmenge $\mathcal{F}$ so gewählt ist, dass die durch GP erzeugte Funktion linear ist, oder
  - es wenig Zeit zur Verfügung gibt.

- **Genetischer Algorithmus:** Der genetische Algorithmus (GA) ist eine Optimierungsmethode mit einem hohen Zeitaufwand, jedoch kann GA das lokale Optimum überspringen. GA wird verwendet, wenn der Zeitfaktor eine untergeordnete Rolle spielt und ein sehr gutes Ergebnis erwartet wird.

8.1.2 Rekursives Prädiktionsverfahren

Alternativ zum obigen Verfahren gibt es das *rekursive Prädiktionsverfahren* (oder das *Rekursionsverfahren*). Dieses Verfahren wird in mehrere Stufen aufgeteilt. Jede Stufe besteht aus einem einfachen Prädiktionsverfahren und hat eigene Trainingsdaten, die die Prädiktionsfehler der vorherigen Stufe sind. Die GP-Parameter, die Funktionenmenge $\mathcal{F}$ und die Terminalmenge $\mathcal{T}$ von jeder Stufe sind gleich und ähnlich, wie sie im einfachen Prädiktionsverfahren vorgestellt wurden.

Funktionsweise des rekursiven Prädiktionsverfahrens

Die Abbildung 8.2 zeigt den Ablauf des rekursiven Prädiktionsverfahrens.

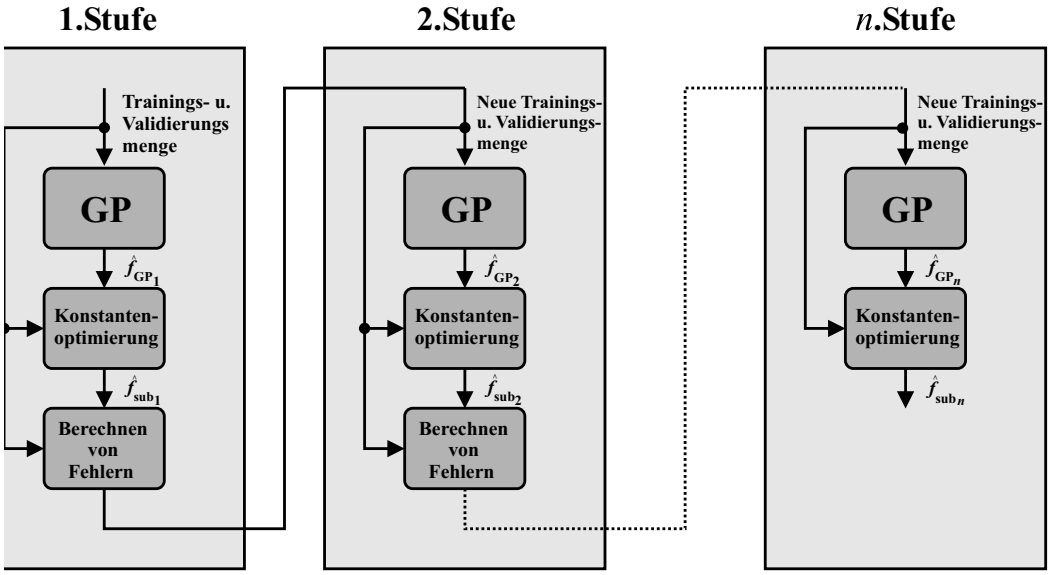


Abbildung 8.2: Ablauf des rekursiven Prädiktionsverfahrens

Das rekursive Prädiktionsverfahren lässt sich folgendermaßen darstellen:

1. Stufe In der 1. Stufe wird zuerst wie beim einfachen Prädiktionsverfahren vorgegangen. Danach werden die Prädiktionsfehler, die sich beim Durchlaufen der Trainingsdaten durch die erste Stufe ergeben, bestimmt, wenn man die Sub-Prädiktionsfunktion $\hat{f}_{\text{sub}_1}$ anwendet. Diese Prädiktionsfehler werden nun als die neuen Trainingsdaten für die 2. Stufe verwendet.

2. Stufe Die 2. Stufe funktioniert zunächst ähnlich wie das einfache Prädiktionsverfahren, indem die 2. Sub-Prädiktionsfunktion $\hat{f}_{\text{sub}_2}$ zu den neuen Trainingsdaten gesucht wird. Dabei wird der Prädiktionsfehler der 1. Stufe modelliert.

Daher entspricht jetzt die gesuchte Prädiktionsfunktion der Summe der 1. Sub-Prädiktionsfunktion $\hat{f}_{\text{sub}_1}$ und der 2. Sub-Prädiktionsfunktion $\hat{f}_{\text{sub}_2}$. Nun werden die Koeffizienten a_0, a_1 und a_2 in beide Sub-Prädiktionsfunktionen eingeführt, und anschließend optimiert.

$$\hat{f}_{\text{sum}_2} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) \tag{8.1}$$

Die Prädiktionsfunktion $\hat{f}_{\text{sum}_2}$ in der Gleichung (8.1) ist jetzt die aktuelle Prädiktionsfunktion. Mit $\hat{f}_{\text{sum}_2}$ werden die weiteren Prädiktionsfehler berechnet, die als die neuen Trainingsdaten in die 3. Stufe eingehen.

3. Stufe Die 3. Stufe funktioniert wie die vorherige Stufe. Nach der Erzeugung der 3. Sub-Prädiktionsfunktion $\hat{f}_{\text{sub}_3}$ wird die Prädiktionsfunktion $\hat{f}_{\text{sum}_3}$ erstellt. Die Koeffizienten a_0, a_1, a_2 und a_3 werden in die Prädiktionsfunktion $\hat{f}_{\text{sum}_3}$ eingefügt, und wieder aktualisiert.

$$\hat{f}_{\text{sum}_3} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) + (a_3 \cdot \hat{f}_{\text{sub}_3}) \tag{8.2}$$

Die Prädiktionsfunktion $\hat{f}_{\text{sum}_3}$ in Gleichung (8.2) ist jetzt die aktuelle Prädiktionsfunktion. Mit $\hat{f}_{\text{sum}_3}$ werden die weiteren Prädiktionsfehler berechnet, die als die neuen Trainingsdaten in die 4. Stufe eingehen.

Der Ablauf des rekursiven Prädiktionsverfahrens wiederholt sich, bis

- die maximale eingegebene Anzahl der Rekursionsstufen überschritten wird, oder
- keine weitere beste Sub-Prädiktionsfunktion gefunden wird, oder
- Overfittig[1] auftritt.

Am Ende des rekursiven Prädiktionsverfahrens bekommt man eine Prädiktionsfunktion f_{sum_n} wie in der folgenden Gleichung (8.3) dargestellt:

$$\hat{f}_{\text{sum}_n} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) + ... + (a_n \cdot \hat{f}_{\text{sub}_n}) \tag{8.3}$$

Es ist zu beachten, dass die Koeffizienten $a_0, a_1, a_2, ..., a_n$ nach dem Ende der jeweiligen Stufe immer aktualisiert werden. Die Koeffizienten $a_1, a_2, ..., a_n$ sind dabei der Gewichtsfaktor der Sub-Prädiktionsfunktion. Sie stellen die Wichtigkeit der Sub-Prädiktionsfunktion dar.

Außerdem ist es möglich, dass mehrere GP-Läufe in einer Rekursionsstufe durchgeführt werden. Jedoch wird nur die beste durch GP entwickelte Prädiktionsfunktion ausgewählt und anschließend zur Konstantenoptimierung benutzt. Diese Prädiktionsfunktion gilt daher als die Sub-Prädiktionsfunktion der Rekursionsstufe .

[1] Siehe Abschnitt 8.2

8.1.3 Vergleich von einfachem Prädiktionsverfahren und rekursivem Prädiktionsverfahren

C. Kraikhaw vergleicht die beiden Verfahren in seiner Diplomarbeit [Krai01]. Dabei kann man feststellen, dass das rekursive Prädiktionsverfahren effizienter als das einfache Prädiktionsverfahren ist.

Die Funktionsweise des einfachen Prädiktionsverfahrens und des rekursiven Prädiktionsverfahrens lässt sich folgendermaßen unterscheiden:

- Beim einfachen Prädiktionsverfahren versucht die GP durch die genetischen Operatoren eine bessere Prädiktionsfunktion in jeder Generation zu finden oder zu entwickeln. Wenn eine bessere Funktion bei der nächsten Generation gefunden wird, geht die beste Funktion der vorherigen Generation verloren.

  Am Ende entsteht nur ein Teil der Originalfunktion, der durch GP nach der letzten Generation gefunden wird. Die Funktionen, die bereits vorher gefunden wurden, gehen im Laufe der GP verloren.

- Im Unterschied zum einfachen Prädiktionsverfahren funktioniert das rekursive Prädiktionsverfahren stufenweise, d.h. nach der 1. Stufe wird eine Sub-Prädiktionsfunktion $\hat{f}_{\text{sub}_1}$ gefunden, die auch ein Teil der Originalfunktion sein könnte. Diese Sub-Prädiktionsfunktion $\hat{f}_{\text{sub}_1}$ wird immer in der gesamten Prädiktionsfunktion beibehalten. In der 2. und den weiteren Stufen könnten weitere Teile der Originalfunktion gefunden werden. Am Ende dieses Verfahrens bilden die Sub-Prädiktionsfunktionen aller Stufen ($\hat{f}_{\text{sub}_1}$, $\hat{f}_{\text{sub}_2}$, $\hat{f}_{\text{sub}_3}$, ..., $\hat{f}_{\text{sub}_n}$) eine gemeinsame gesamte Prädiktionsfunktion $\hat{f}_{\text{sum}_n}$.

Bei der Verwendung der beiden Verfahren kann man nicht garantieren, dass man immer die Originalfunktion findet. Die beiden Verfahren sind aber in der Lage, eine Annäherungsfunktion zu finden. Obwohl diese Annäherungsfunktion nicht identisch mit der Originalfunktion ist, verhält sie sich ähnlich wie die Originalfunktion.

8.2 Overfittingproblem und Gegenmaßnahmen

Ein häufiges Problem bei der Prädiktion ist, dass die gefundene Prädiktionsfunktion innerhalb der Trainingsdaten gut prädiktieren kann. Jedoch ist diese Prädiktionsfunktion für die Prädiktion der Daten außerhalb der Trainingsdaten ungeeignet. Dieses Problem nennt man *Overfitting* oder *Überanpassung*. Das Overfitting-Problem erscheint deutlich, wenn die Trainingsdaten verrauscht sind. Da die GP und die weiteren Optimierungsverfahren (z.B. Least-Squares oder GA) versuchen, während des Trainings sich dem Rauschanteil anzupassen.

Bei KNN, z.B. in [Sar01], gibt es verschiedene Methoden, mit diesem Problem umzugehen. Eine einfache und sehr bekannte Methode ist das *frühe Stoppen*. Dies kann sowohl für GP als auch für die Konstantenoptimierung sehr nützlich sein.

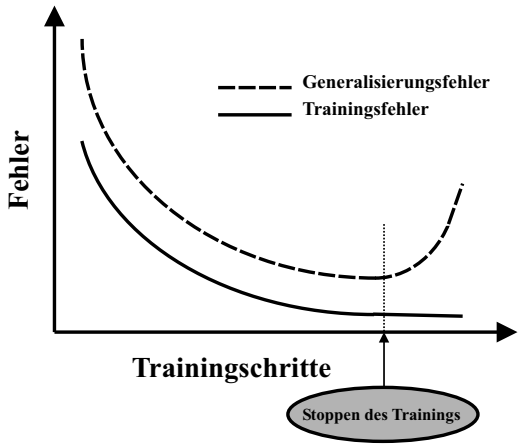


Abbildung 8.3: Entwicklung des Tranings- und Generalisierungsfehlers über die Trainingschritte

Der Gedanke beim frühen Stoppen besteht darin, die Anpassung des Trainings an die Trainingsdaten beim Beginn des Overfittings vorzeitig abzubrechen (siehe Abbildung 8.3). Der Eintritt von Overfitting beim Training kann durch ein Wiederansteigen des Generalisierungsfehlers nach dem Erreichen den minimalen Wert festlegen. Dabei versteht man unter Generalisierung einen Prozess, die Beschreibung des Ganzen aus einzelnen Teilen abzuleiten, vom konkreten auf den allgemeinen Fall zu schließen oder aus einer bzw. mehrerer Instanzen eine Klasse von Objekten zu definieren.

Die algorithmische Vorschrift für das frühe Stoppen kann daher etwa folgendermaßen zusammengefasst werden:

- Das Training wird mit Hilfe der Trainingsmenge durchgeführt. Dabei überwacht man während des Trainings den Generalisierungsfehler anhand einer separaten Menge als Maß zur Feststellung, wann das Overfitting auftritt.

- Tritt beim Training Overfitting ein, d.h. der Generalisierungsfehler steigt nach dem Erreichen seines minimalen Wertes an, wird das Training gestoppt.

- Das beste gefundene Ergebnis ist das Ergebnis mit dem niedrigsten Generalisierungsfehler.

Das frühe Stoppen wird bei GP und der Konstantenoptimierung eingesetzt, um Overfitting zu vermeiden.

8.2.1 Schätzung des Generalisierungsfehlers

Der Generalisierungsfehler kann nicht direkt gemessen werden, sondern wird mit Hilfe einer separaten Menge aus den Trainingsdaten geschätzt. Im Allgemeinen wird diese separate Menge an Trainingsdaten *Validierungsmenge* genannt.

Die Daten außerhalb der Trainingsdaten wird *Testmenge* genannt. Hat man das Training beendet, wird mit Hilfe der Testmenge der tatsächliche Fehler bestimmt, der sich bei der Testmenge ergibt.

Es gibt drei Möglichkeiten zur Schätzung des Generalisierungsfehlers:

Teilung der Trainingsdaten

Zur Messung des Generalisierungsfehlers werden die Trainingsdaten in

- eine *Trainingsmenge* und
- eine *Validierungsmenge*

unterteilt ([Pre98, Sar01]). Die beiden Mengen müssen unabhängig voneinander sein. Die Teilung der Trainingsdaten in Trainings- und Validierungsmenge kann *zufällig* oder *deterministisch* sein.

Die Fehler aus der Trainings- und Validierungsmenge werden als Trainings- und Validierungsfehler bezeichnet. Der Validierungsfehler entspricht nun dem gesuchten Generalisierungsfehler.

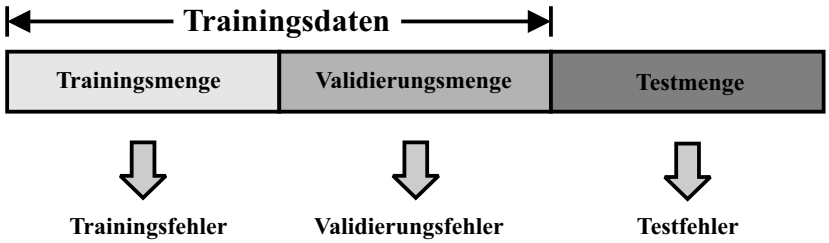


Abbildung 8.4: Teilung der Trainingsdaten in *Trainingsmenge* und *Validierungsmenge*. Die Daten außerhalb der Trainingsdaten wird *Testmenge* genannt.

Da der Generalisierungsfehler nur mit Hilfe einer Validierungsmenge geschätzt wird, kann es passieren, dass das frühe Stoppen in die Irre führt. Der gewählte Stoppzeitpunkt ist nicht wirklich günstig, sondern sieht nur aufgrund einer zufälligen Eigenschaft der endlichen Validierungsmenge günstig aus. Dieser Effekt ist umso stärker, je kleiner die benutzte Validierungsmenge ist. Man kann sie aber nicht beliebig groß machen, da sie von den wertvollen Trainingsdaten abgezweigt werden muss. Man kann dieses Problem verkleinern, indem man eine sogenannte n-fache Kreuzvalidierung ausführt.

n-Fache Kreuzvalidierung

Bei n-fachen Kreuzvalidierung ([Neal96, Sche97, Sar01]) werden die Trainingsdaten in n Teilmengen unterteilt (Abbildung 8.5).

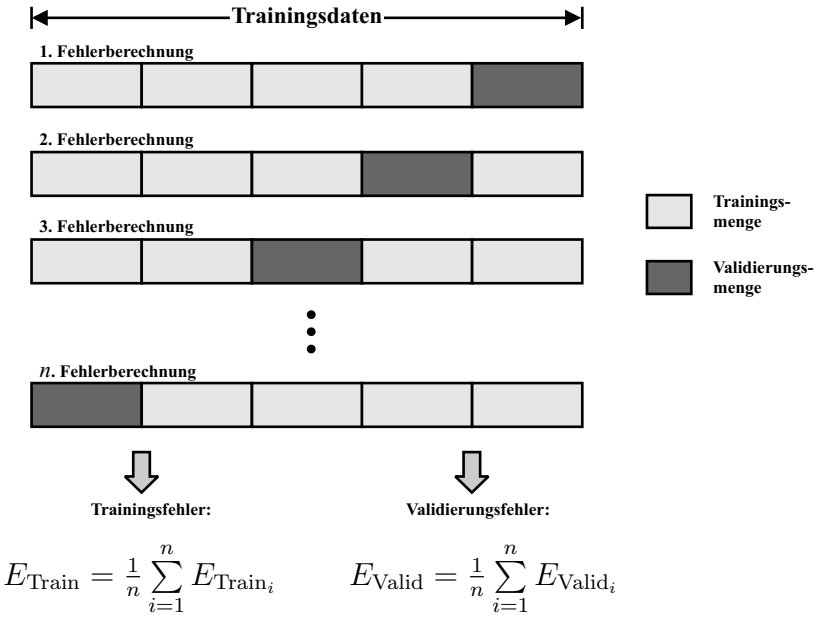


Abbildung 8.5: Beispiel: n-Fache Kreuzvalidierung

In diesem Verfahren wird die Fehlerberechnung n-mals durchgeführt. Der Trainingsfehler und Validierungsfehler wird folgendermaßen berechnet:

- Bei der 1. Fehlerberechnung wird die n. Menge der Trainingsdaten als Validierungsmenge, und die anderen Mengen der Trainingsdaten als Trainingsmenge verwendet.
- Bei der 2. Fehlerberechnung wird die $(n-1)$. Menge der Trainingsdaten als Validierungsmenge, und die anderen Mengen der Trainingsdaten als Trainingsmenge verwendet.
- Die Fehlerberechnung wird bis zur n. Fehlerberechnung durchgeführt.
- Bei der n. Fehlerberechnung wird die 1. Menge der Trainingsdaten als Validierungsmenge, und die anderen Mengen der Trainingsdaten als Trainingsmenge verwendet.
- Der mittlere Wert aller n Fehler aus der Trainingsmenge und der mittlere Wert aller n Fehler aus der Validierungsmenge entsprechen folglich dem Trainings- und dem Validierungsfehler.

Der Validierungsfehler wird auch hier als Maß des Generalisierungsfehlers verwendet. Die n-fache Kreuzvalidierung benötigt einen sehr hohen Zeitaufwand. Dies ist der große Nachteil des Verfahrens.

Bootstrap

Das Bootstrap ([Sar01]) funktioniert zuerst ähnlich wie die n-fache Kreuzvalidierung. Die Trainingsdaten werden in n Mengen aufgeteilt. Jedoch wird die Fehlerberechung nicht n-mals durchgeführt. Nur einige Fehlerberechnungen von n Fehlerberechnungen der n-fachen Kreuzvalidierung werden zufällig ausgewählt und durchgeführt. Der mittlere Wert von Fehlern aus Trainings- und Validierungsmenge von den ausgewählten Fehlerberechnungen entsprechen daher dem Trainings- und Validierungsfehler.

Dieses Verfahren kann den hohen Zeitaufwand der n-fachen Kreuzvalidierung reduzieren. Jedoch ist es nicht einfach festzulegen, wie oft man das Training durchführen muss.

Obwohl es drei Möglichkeiten zur Schätzung des Generalisierungsfehlers gibt, wird in dieser Arbeit nur die Schätzung mit der Teilung der Trainingsdaten verwendet, da sie wenigsten Zeitaufwand benötigt.

8.2.2 Stoppkriterien

Die ursprüngliche Annahme war, dass die Kurve des Generalisierungsfehlers über der Zeit eine konvexe Form hat, wie in Abbildung 8.3 gezeigt. Es hat sich aber gezeigt, dass das wirkliche zeitliche Verhalten des Generalisierungsfehlers bei weitem komplizierter ist. Der Verlauf des Generalisierungsfehlers hat mehrere lokale Minima bevor die endgültige Überanpassung einsetzt.

Aus dem Vorhandensein mehrerer lokaler Minima in der Generalisierungsfehlerkurve folgt, dass ein *optimales* Kriterium für das frühe Stoppen nicht so einfach sein kann. Auch wenn der Generalisierungsfehler ansteigt, kann man nicht sicher sein, dass er nicht später einmal auf einen noch kleineren Wert als bisher abfallen wird.

Bevor wir uns mit den Stoppkriterien beschäftigen, müssen zunächst einige Begriffe erläutert werden.

Sei E die Fehlerfunktion des Lernverfahrens, zum Beispiel der quadratische Fehler. Ferner sei $E_{\text{Train}}(t_{\text{T}})$ der mittlere Fehler pro Beispiel über alle Beispiele der Trainingsmenge, gemessen im Trainingsschritt t_{T}. $E_{\text{Valid}}(t_{\text{T}})$ ist analog dazu der mittlere Fehler bei der Validierungsmenge im Trainingsschritt t_{T} und $E_{\text{Test}}(t_{\text{T}})$ ist der Fehler bei der Testmenge im Trainingsschritt t_{T}. Dabei entspricht der Trainingsschritt bei GP und GA der Generation G.

1. Stoppkriterium

Zuerst wird der *Generalisierungsverlust GL* in Trainingsschritt t_{T} definiert, als die relative Mehrgröße (in Prozent) des aktuellen Validierungsfehlers, verglichen mit

seinem bisherigen Minimum:

$$GL(t_{\mathrm{T}}) = 100 \cdot \left(\frac{E_{\mathrm{Valid}}(t_{\mathrm{T}})}{\min\limits_{\acute{t} \leq t} E_{\mathrm{Valid}}(\acute{t}_{\mathrm{T}})} - 1 \right) \tag{8.4}$$

Ein hoher Generalisierungsverlust ist ein naheliegendes Kriterium, um das Training abzubrechen. Diese Überlegung definiert die Stoppkriterien wie folgt:

$$GL_{\alpha} : \text{stoppe nach dem ersten Trainingsschritt } t_{\mathrm{T}} \text{ bei } GL(t_{\mathrm{T}}) > \alpha$$

2. Stoppkriterium

Nimmt der Trainingsfehler sehr stark ab, muss das Training rechtzeitig abgebrochen werden. Daher benötigt man ein neues Stoppkriterium. Für das neue Stoppkriterium wird zunächst der *Trainingsfortschritt* P_l definiert, als:

$$P_l(t_{\mathrm{T}}) = 1000 \cdot \left(\frac{\sum\limits_{\acute{t}_{\mathrm{T}} \in t_{\mathrm{T}}-l+1}^{t_{\mathrm{T}}} E_{\mathrm{Valid}}(\acute{t}_{\mathrm{T}})}{l \cdot \min\limits_{\acute{t}_{\mathrm{T}} \in t_{\mathrm{T}}-l+1}^{t_{\mathrm{T}}} E_{\mathrm{Valid}}(\acute{t}_{\mathrm{T}})} - 1 \right) \tag{8.5}$$

Diese Gleichung beschreibt, um wieviel der Validierungsfehler während des Trainingsstreifens der Länge l höher war als der minimale Fehler. Der Trainingsstreifen der Länge l wird durch eine Sequenz von Trainingsschritten mit den Nummern $n+1, ..., n+l$ definiert, wobei n durch l teilbar ist.

Das Verhältnis von Generalisierungsverlust und Trainingsfortschritt wird anschließend beim Stoppkriterium betrachtet.

Daraus folgt:

$$\begin{aligned} PQ_{\beta,l} \quad : \quad & \text{stoppe nach dem ersten durch } l \text{ teilbaren Trainingschritt } t_{\mathrm{T}} \\ & \text{mit } \frac{GL(t_{\mathrm{T}})}{P_l(t_{\mathrm{T}})} > \beta \end{aligned}$$

3. Stoppkriterium

Die weitere Klasse von Stoppkriterien betrachtet nur das Vorzeichen aufeinanderfolgender Änderungen des Generalisierungsfehlers. Es wird gestoppt, wenn dieser mehrmals hintereinander wechselt.

4. Stoppkriterium

Alle drei vorher erwähnte Klassen von Stoppkriterien wurden bereits für automatisches Lernen beim KNN in [Pre98] vorgeschlagen. Jedoch sind sie nicht hinreichend, um das Training abzubrechen, wenn keine oder nur sehr kleine Überanpassung auftritt. Deshalb soll das Training dann abbrechen, wenn

- eine maximale Anzahl von Trainingsschritten erreicht wird, oder
- der Trainingsfehler länger als ein bestimmte Anzahl von Traininsschritten stagniert.

Die maximale Anzahl von Trainingsschritten bei GP und GA ist die maximale Generation $G_{\max}$. Dieses 4. Stoppkriterium wird in dieser Arbeit verwendet.

8.3 Prädiktionsfehler

Normalized Mean Squared Error

Der Prädiktionsfehler einer Zeitreihe wird im Normalfall mit *Normalized Mean Squared Error (NMSE)* berechnet.

NMSE ist folgendermaßen definiert:

$$E_{\mathrm{NMSE}_k} = \frac{1}{\sigma_x^2 \cdot N} \sum_{t=0}^{N-1} (x_{t+k} - \hat{x}_{t+k})^2 \tag{8.6}$$

Dabei

- N die Länge der Zeitreihe,
- x_{t+k} der tatsächliche Wert zur Zeit $t + k$,
- $\hat{x}_{t+k}$ der prädiktierte Wert zur Zeit $t + k$,
- k der Prädiktionsschritt ($k \geq 1$),
- σ_x die Standardabweichung der zur Fehlerberechnung verwendeten tatsächlichen Werte.

ist.

Je kleiner E_{NMSE_k} ist, desto besser ist die Prädiktion. Nimmt man an, dass alle prädiktierten Werte $\hat{x}_{t+k}$ dem Mittelwert der Zeitreihe entsprechen, wird E_{NMSE_k} ungefähr eins betragen.

Verlauf des Prädiktionsfehlers bei Prädiktion mit der globalen und lokalen Annäherung

A.A. Tsonis verwendet in seiner Arbeit [Tso92] *Normalized Root Mean Squared Error* als Prädiktionsfehler, das der Quadratwurzel des NMSE entspricht. Sein Prädiktionsfehler wächst bei der Prädiktion mit dem Verfahren der globalen und lokalen Annäherung.

- Bei iterativer Einschrittprädiktion wird die Quadratwurzel des NMSE proportional zu dem größten Lyapunov-Exponent der Zeitreihe $\lambda_{\max}$ wachsen. Daher wird Prädiktionsfehler NMSE folgendermaßen erwartet:

$$\sqrt{E_{\mathrm{NMSE}_k}} \sim N^{\frac{-(m+1)}{d_\mathrm{A}}} e^{\lambda_{\max} \cdot k} \tag{8.7}$$

- Bei direkter Mehrschrittprädiktion wird Prädiktionsfehler NMSE folgendermaßen erwartet.

$$\sqrt{E_{\mathrm{NMSE}_k}} \sim N^{\frac{-(m+1)}{d_\mathrm{A}}} e^{(m+1)\cdot\lambda_{\max} \cdot k} \tag{8.8}$$

  Wobei:

  - E_{NMSE_k} NMSE beim k. Prädiktionsschritt,
  - d_A die Dimension des Attraktors,
  - N die Länge der Zeitreihe,
  - m die Ordnung der Prädiktionsfunktion,
  - $\lambda_{\max}$ der größte Lyaponov-Exponent der Zeitreihe und
  - k der Prädiktionsschritt ist.

Mit den Gleichungen (8.7) und (8.8) kann man feststellen, dass der Prädiktionsfehler $\sqrt{E_{\mathrm{NMSE}_k}}$ mit dem Prädiktionsschritt exponential steigt. Außerdem ist die iterative Einschrittprädiktion besser als die direkte Mehrschrittprädiktion, wenn $m > 0$. Diese Feststellung ist jedoch nicht immer richtig. Offensichtlich sind die beide Gleichungen (8.7) und (8.8) nicht ausreichend, um den Prädiktionsfehler der iterativen und der direkten Mehrschrittprädiktion zu beurteilen.

Prädiktionsfehler bei beim Prädiktion mit GP

Bei der Prädiktion mit GP (einfaches Prädiktionsverfahren oder rekursives Prädiktionsverfahren) wird normalerweise GP zunächst lernen, die Trainingsdaten mit Einschrittprädiktion zu prädiktieren. Danach wird die durch GP entwickelte Prädiktionsfunktion die zukünftigen Werte berechnen, wie in der iterativen Einschrittprädiktion vorgeschrieben ist.

Nicht alle Prädiktionsfunktionen der Einschrittprädiktion sind zur iterativen Einschrittprädiktion geeignet. Sie können Pol- oder/und Nullstellen besitzen. Falls ein prädiktierter Wert der Pol- oder Nullstelle der Prädiktionsfunktion entspricht, wird die Prädiktionsfunktion nicht mehr in der Lage sein, weitere Werte vernünftig zu prädiktieren.

Betrachten wir nochmals den prädiktierten Wert x_{t+k} aus der iterativen Einschrittprädiktion mit Gleichung (7.2). Definiert man E_k als die Abweichung der Prädiktion beim k. Prädiktionsschritt, d.h. $E_k = x_{t+k} - \hat{x}_{t+k}$, folgt daraus:

$$\begin{aligned}
E_1 &= x_{t+1} - \hat{x}_{t+1} \\
&= x_{t+1} - \hat{f}(x_t, x_{t-1}, x_{t-2}, ..., x_{t-m}) \\
E_2 &= x_{t+2} - \hat{x}_{t+2} \\
&= x_{t+2} - \hat{f}(x_{t+1} - E_1, x_t, x_{t-1}, ..., x_{t-m+1}) \\
E_3 &= x_{t+3} - \hat{x}_{t+3} \\
&= x_{t+3} - \hat{f}(x_{t+2} - E_2, x_{t+1} - E_1, x_t, ..., x_{t-m+2}) \\
&\vdots \\
E_k &= x_{t+k} - \hat{x}_{t+k} \\
&= x_{t+k} - \hat{f}(x_{t+k-1} - E_{k-1}, x_{t+k-2} - E_{k-2}, x_{t+k-3} - E_{k-3}, ... \\
&\qquad ..., x_{t-m+k-1} - E_{-m+k-1})
\end{aligned} \tag{8.9}$$

Die Abweichungen $E_1, E_2, E_3, ..., E_k$ können positives oder negatives Vorzeichen besitzen. Nach Gleichung (8.7) sollen die absoluten Werte der Abweichung $|E_1|, |E_2|, |E_3|, ..., |E_k|$ mit dem Prädiktionsschritt steigen, damit $\sqrt{E_{\mathrm{NMSE}_k}}$ exponential wächst.

Wenn aber die durch GP entwickelte Prädiktionsfunktion für die iterative Einschrittprädiktion stabil ist, und deren prädiktierte Werte innerhalb eines bestimmten Bereichs, der nicht sehr von dem Bereich der deterministischen Zeitreihe abweicht, liegen, werden die Abweichungen $E_1, E_2, E_3, ..., E_k$ also in einem Grenzbereich bleiben. Folglich steigt der Prädiktionsfehler E_{NMSE_k} nicht bis unendlich. Außerdem kann es vorkommen, dass der Prädiktionsfehler im k. Prädiktionsschritt E_{NMSE_k} kleiner als Prädiktionsfehler im $(k-1)$. Prädiktionsschritt $E_{\mathrm{NMSE}_{k-1}}$ ist.

Da es keine Garantie dafür gibt, dass die durch GP entwickelte Prädiktionsfunktion für die iterative Einschrittprädiktion gut ist, wird die direkte Mehrschrittprädiktion eine Alternative zur iterativen Einschrittprädiktion sein. In diesem Fall lernt GP direkt den zukünftigen Wert zu prädiktieren. Der Prädiktionsfehler der direkte Mehrschrittprädiktion soll kleiner als der Prädiktionsfehler der iterativen Einschrittprädiktion sein. Dieses Verfahren hat aber den Nachteil, dass der Zeitaufwand extrem hoch ist. Angenommen, die Originalfunktion einer chaotischen Zeitreihe ist eine mathematische Funktion 2.-Ordnung, wird die direkte Prädiktionsfunktion für den 10. Prädiktionsschritt eine mathematische Funktion 2^{10}.-Ordnung. Diese Funktion ist sehr komplex, und GP braucht viel Zeit, um sie zu finden.

8.4 Zusammenfassung

Zur Prädiktion der deterministischen chaotischen Zeitreihe werden in dieser Arbeit zwei Verfahren angewendet, die auf GP basieren. Diese beiden Verfahren sind das einfache Prädiktionsverfahren und das rekursive Prädiktionsverfahren.

Beim einfachen Prädiktionsverfahren wird zuerst die Prädiktionsfunktion $\hat{f}_{\mathrm{GP}}$ durch GP erzeugt. Danach werden die Koeffizienten in der Prädiktionsfunktion $\hat{f}_{\mathrm{GP}}$ eingefügt. Anschließend werden alle Koeffizienten und die Konstanten optimiert.

Das rekursive Prädiktionsverfahren besteht aus mehreren Stufen, jede Stufe des rekursiven Prädiktionsverfahrens hat die gleiche Funktionsweise wie beim einfachen Prädiktionsverfahren. Jedoch sind die Trainingsdaten der jeweiligen Stufe der Prädiktionsfehler der vorherigen Stufe. Dadurch wird in jeder Rekursionsstufe eine Sub-Prädiktionsfunktion entwickelt, die einen Teil der richtigen Prädiktionsfunktion darstellt. Am Ende des Prädiktionsprozesses bilden alle Sub-Prädiktionsfunktionen $\hat{f}_{\mathrm{sub}_1}$, $\hat{f}_{\mathrm{sub}_2}$, $\hat{f}_{\mathrm{sub}_3}$, ..., $\hat{f}_{\mathrm{sub}_n}$ gemeinsam die Prädiktionsfunktion $\hat{f}_{\mathrm{sum}_n}$. Das rekursive Prädiktionsverfahren hat den Vorteil gegenüber dem einfachen Prädiktionsverfahren, dass die durch GP gut entwickelten Prädiktionsfunktionen während des GP-Laufes nicht verloren gehen.

Die beiden Prädiktionsverfahren fordern keine höheren mathematischen Kenntnisse. Bei der Verwendung der beiden Verfahren kann man nicht garantieren, dass man immer die Originalfunktion findet. Die beiden Verfahren sind aber in der Lage, eine Annäherungsfunktion zu finden. Obwohl diese Annäherungsfunktion nicht mit der Originalfunktion identisch ist, verhält sie sich ähnlich wie die Originalfunktion.

Beide Prädiktionsverfahren haben ein gemeinsames Problem, das so genannte Overfitting. Dies bedeutet, dass die gefundene Prädiktionsfunktion innerhalb der Trainingsdaten gut prädiktieren kann. Jedoch ist diese Prädiktionsfunktion für die Prädiktion der Daten außerhalb der Trainingsdaten ungeeignet.

Dieses Problem kann durch das frühe Stoppen umgegangen werden. Das in dieser Arbeit verwendete frühe Stoppen funktioniert wie folgt: man teilt zunächst die Trainingsdaten in zwei Teile, nämlich in Trainingsmenge und Validierungsmenge. Die Daten der Trainingsmenge werden zum Training eingesetzt. Die Daten der Validierungsmenge werden zur Schätzung des Fehlers außerhalb der Trainingsdaten verwendet. Während des Trainings wird dann bei jedem Trainingschritt der Fehler aus der Validierungsmenge (Validierungsfehler) beobachtet. Das Training wird gestoppt, wenn der Validierungsfehler seinen niedrigsten Wert erreicht hat, und anzusteigen beginnt. Das beste gefundene Ergebnis ist das Ergebnis mit dem niedrigsten Validierungsfehler.

Der Verlauf des Validierungsfehlers hat in Wirklichkeit mehrere lokale Minima, bevor das endgültige Overfitting einsetzt. Aus dem Vorhandensein mehrerer lokaler Minima in der Validierungsfehlerkurve folgt, dass ein Kriterium für das frühe Stoppen

nicht so einfach sein kann. Das Training soll dann abbrechen, wenn eine Maximalzahl von Trainingsschritten erreicht wird, oder der Trainingsfehler länger als eine bestimmte Anzahl von Trainingsschritten stagniert.

Bei der Prädiktion mit GP wird normalerweise GP zunächst lernen, die Trainingsdaten mit Einschrittprädiktion zu prädiktieren. Danach wird die durch GP entwickelte Prädiktionsfunktion die zukünftigen Werte verwenden, wie in der iterativen Einschrittprädiktion vorgeschrieben ist. Da es keine Garantie dafür gibt, dass die durch GP entwickelte Prädiktionsfunktion für die iterative Einschrittprädiktion gut ist, wird die direkte Mehrschrittprädiktion eine Alternative zur iterativen Einschrittprädiktion sein. Jedoch ist dieses Verfahren sehr rechenintensiv. GP braucht viel Zeit, um eine Prädiktionsfunktion für die direkte Mehrschrittprädiktion zu entwickeln.

9 Beispiele der Prädiktion von chaotischen Zeitreihen

Dieses Kapitel beschäftigt sich mit den Experimenten der Prädiktion von chaotischen Zeitreihen. Wir beginnen zunächst mit dem Vergleich von dem einfachen Prädiktionsverfahren und dem rekursiven Prädiktionsverfahren. Danach werden vier weitere Experimente der Prädiktion von chaotischen Zeitreihen mit dem einfachen Prädiktionsverfahren oder dem rekursiven Prädiktionsverfahren dargestellt. Dabei wird der Unterschied zwischen der Prädiktion der chaotischen Zeitreihe ohne und mit Störungsreduzierung gezeigt. Zum Schluss werden die Ergebnisse der Experimente diskutiert.

9.1 Vergleich von einfachem Prädiktionsverfahren und rekursivem Prädiktionsverfahren

In diesem ersten Teil werden die Prädiktionen einer chaotischen Zeitreihe durch das einfache Pradiktionsverfahren und das rekursive Prädiktionsverfahren durchgeführt. Anschließend werden Ergebnisse aus den beiden Prädiktionsverfahren verglichen. Dieser Vergleich der beiden Prädiktionsverfahren wurde von C. Kraikhaw durchgeführt und bereits in seiner Diplomarbeit [Krai01] detailliert angegeben.

Beim Experiment wird versucht, die durch die Mackey-Glass-Gleichung oder M-G-Gleichung beschriebenen chaotischen Zeitreihen zu prädiktieren. Die M-G-Gleichung ist ein Modell für die Produktion der weißen Blutkörperchen. Hier wird die M-G-Gleichung nochmals dargestellt:

$$x_{t+1} = x_t + \frac{b \cdot x_{t-\Delta}}{1 + (x_{t-\Delta})^c} - a \cdot x_t \tag{9.1}$$

wobei die Konstanten $a = 0,1$ $b = 0,2$ und $c = 10$ eingesetzt sind. Diese Gleichung ist eine Funktion des Verschiebungsparameters Δ. Wenn Δ größer als 16,8 ist, verhält sich die Gleichung chaotisch. Beim Experiment wird $\Delta = 30$ eingesetzt.

Die Experimente werden in den zwei Situationen durchgeführt.

1. **Situation:** Simulation mit den durch die M-G-Gleichung beschriebenen Daten.

2. **Situation:** Simulation mit den durch die M-G-Gleichung beschriebenen Daten, jedoch werden diese von weißem Rauschen mit SNR = 20 dB überlagert.

Für das einfache Prädiktionsverfahren und das rekursive Prädiktionsverfahren wird die erste Möglichkeit zum Einfügen der Koeffizienten verwendet, d.h. Einfügen von nur einem Koeffizient in die mathematische Funktion. Zur Konstantenoptimierung wird das Least-squares-Verfahren eingesetzt.

Zur Vermeidung des Overfittings wird das frühe Stoppen eingesetzt. Die Schätzung des Generalisierungsfehlers erfolgt nach der Teilung der Trainingsdaten. Folglich entspricht der Generalisierungsfehler dem Validierungsfehler. Da der Verlauf des Validierungsfehlers mehrere lokale Minima hat, bevor das endgültige Overfitting einsetzt, wird das Training nur gestoppt, wenn die Maximalzahl von Trainingsschritten erreicht wird. Das beste gefundene Ergebnis ist das Ergebnis mit dem niedrigsten Generalisierungsfehler.

9.1.1 Einstellung des Experiments

Zu Beginn der Simulation wurde die M-G-Gleichung mit den bestimmten Anfangswerten gestartet, um weitere Daten zu erzeugen. Damit die Anfangswerte keinen Einfluss auf die Daten haben, wurden zuerst 1000 Daten berechnet. Nach der Erzeugung der ersten 1000 Daten wurden weiteren 400 Daten berechnet. Nur die letzten 300 Daten von diesen 400 Daten wurden bei der Simulation verwendet. Dabei sind die Daten in drei Mengen wie folgt aufgeteilt:

- Die Daten von der 101. Position (x_{101}) bis zur 200. Position (x_{200}) bilden die Trainingsmenge $\mathbf{S}_{\text{Train}}$.
- Die Daten von der 201. Position (x_{201}) bis zur 300. Position (x_{300}) bilden die Validierungsmenge $\mathbf{S}_{\text{Valid}}$.
- Die Daten von der 301. Position (x_{301}) bis zur 400. Position (x_{400}) bilden die Testmenge $\mathbf{S}_{\text{Test}}$.

Wichtige Bedingungen des GP-Laufes sind folgende:

Terminalmenge und Funktionenmenge

Die CTF-GP [Hoer96] wurde in den beiden Prädiktionsverfahren verwendet. Allerdings entspricht die in den Simulationen verwendete kontextfreie Grammatik der folgenden Funktionenmenge $\mathcal{F}_{\text{MG}}$ und der folgenden Terminalmenge $\mathcal{T}_{\text{MG}}$.

Die Funktionenmenge $\mathcal{F}_{\text{MG}}$ ist also:

$$\mathcal{F}_{\text{MG}} = \{\sin, \cos, \log, \exp, +, -, \times, \div\} \tag{9.2}$$

Die Terminalmenge $\mathcal{T}_{\text{MG}}$ ist also:

$$\begin{aligned} \mathcal{T}_{\text{MG}} \quad = \quad & \{x_t, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}, \\ & -10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0\} \end{aligned} \tag{9.3}$$

wobei: x_t der Wert zum Zeitpunkt t, x_{t-1} der Wert zum Zeitpunkt $t-1$ usw. ist. $-10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0$ sind die einfachen konstanten Werte im Bereich von $-10,0$ bis $10,0$.

Fitnessfunktion

Die Rohfitness f_{r} ist die Summe des Quadrates der Differenz zwischen dem tatsächlichen Wert und dem prädiktierten Wert in der Trainingsmenge $\mathbf{S}_{\text{Train}}$. Daraus folgt:

$$f_{\text{r}} = \sum_{t=101}^{200} (x_t - \hat{x}_t)^2 \tag{9.4}$$

Dabei ist:

- x_t der tatsächliche Wert zum Zeitpunkt t,
- $\hat{x}_t$ der prädiktierte Wert zum Zeitpunkt t.

Man kann außerdem diese Rohfitness f_{r} als Trainingsfehler E_{Train} bezeichnen.

Zur Vermeidung des Overfittings wird der Validierungsfehler E_{Valid} beim GP-Lauf und bei der Konstantenoptimierung berücksichtigt.

Der Validierungsfehler E_{Valid} und der Testfehler E_{Test} entsprechen ebenfalls der Summe des Quadrates der Differenz zwischen dem tatsächlichen Wert und dem prädiktierten Wert in der Validierungsmenge $\mathbf{S}_{\text{Valid}}$ und Testmenge $\mathbf{S}_{\text{Test}}$.

Einstellung der GP-Parameter und Abbruchkriterium

Zum Vergleich zwischen dem einfachen Prädiktionsverfahren und dem rekursiven Prädiktionsverfahren müssen die beiden Verfahren mit dem gleichen Rechenaufwand arbeiten.

CTF-GP ist der Hauptanteil des einfachen und des rekursiven Prädiktionsverfahrens und arbeitet auf der Basis mehrerer Individuen. Außerdem ist die Summe aller Individuen leicht zu bestimmen oder zu begrenzen. Aus diesem Grund ist der Rechenaufwand in diesem Experiment als die Summe aller Individuen definiert. Es gibt weitere Gründe für diese Definition.

- Der verfügbare Rechner wurde von mehreren Benutzern verwendet, folglich lässt sich die Dauer der Simulation nicht als Rechenaufwand definieren.
- Beim Versuch wurde das rekursive Prädiktionsverfahren mit 5 Stufen simuliert. Dies bedeutet, dass in einem rekursiven Prädiktionsverfahren 5-mal Konstantenoptimierung mit dem Least-Squares Verfahren durchgeführt wurde, anstatt 1-mal wie im einfachen Prädiktionsverfahren. Da der Zeitaufwand einer Konstantenoptimierung mit dem Least-Squares Verfahren im Vergleich mit dem Zeitaufwand eines CTF-GP-Laufens extrem klein ist, kann der Zeitaufwand der Konstantenoptimierung vernachlässigt werden.

Die Summe der Individuen wurde für die beiden Verfahren folgendermaßen berechnet:

- Für das einfache Prädiktionsverfahren beträgt die Summe der Individuen S_{einfach}:
$$S_{\text{einfach}} = M \times (G_{\max} + 1) \tag{9.5}$$
Wobei M Populationsgröße, und $G_{\max}$ die max. Generation in einem CTF-GP-Lauf ist.
- Für das rekursive Prädiktionsverfahren beträgt die Summe der Individuen S_{rekursiv}:
$$S_{\text{rekursiv}} = M \times (G_{\max} + 1) \times N_{\text{r}} \tag{9.6}$$
Wobei N_{r} die Anzahl der Stufen des rekursiven Prädiktionsverfahrens ist.

Die Einstellungen der Parameter der GP sind in der Tabelle 9.1 dargestellt. Der GP-Lauf wird nur gestoppt, wenn die Anzahl der maximalen Generation $G_{\max}$ erreicht ist.

9.1.2 Simulationsergebnisse

Für jedes Prädiktionsverfahren werden zehn Versuche durchgeführt. Allerdings werden nur die Versuche mit dem kleinsten Trainings-, Validierungs- und Testfehler aus jedem Prädiktionsverfahren (die besten Versuche) ausgewählt und in diesem Abschnitt dargestellt.

1. Situation: M-G-Gleichung

Der Trainingsfehler E_{Train} und der Validierungsfehler E_{Valid} vor und nach der Konstantenoptimierung beim einfachen Prädiktionsverfahren werden in der Tabelle 9.2 dargestellt. Der Testfehler E_{Test} am Ende der Versuche wird ebenso gezeigt.

Einstellungen der GP-Parameter	Einfaches Verfahren	Rekursives Verfahren
Anzahl der Rekursionsstufen N_{r}	-	5
Anzahl der GP-Läufe in einer Rekursionsstufe	-	1
Populationsgröße M	500	200
Max. Generation $G_{\max}$	99	49
Summe der Individuen	50000	50000
Kreuzungswahrscheinlichkeit p_{c}	0,9	
Reproduktionswahrscheinlichkeit p_{r}	0,1	
Mutationswahrscheinlichkeit p_{m}	0,1	
Max. Ableitungstiefe der Ausgangspopulation $D_{\mathrm{div}_{\mathrm{initial}}}$	30	
Max. Ableitungstiefe während des restl. Laufes $D_{\mathrm{div}_{\mathrm{created}}}$	60	
Initialisierungsmethode	Zufällige Initialisierung	
Selektionsverfahren	Fitnessproportional	
Elitismus	Ja	

Tabelle 9.1: Einstellungen der GP-Parameter im einfachen Prädiktionsverfahren und im rekursiven Prädiktionsverfahren

Konstanten-optimierung	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
Vorher	$4,40 \cdot 10^{-3}$	$8,19 \cdot 10^{-3}$	-
Nachher	$4,23 \cdot 10^{-3}$	$8,01 \cdot 10^{-3}$	$7,12 \cdot 10^{-3}$

Tabelle 9.2: Trainingsfehler E_{Train}, Validierungsfehler E_{Valid} vor und nach der Konstantenoptimierung und Testfehler E_{Test} am Ende der Versuche beim einfachen Prädiktionsverfahren

Obwohl die Rekursionsstufe von fünf eingestellt ist, werden die Ergebnisse bei dem rekursiven Prädiktionsverfahren aus dem besten Versuch bis zur 4. Rekursionsstufe dargestellt, da keine Sub-Prädiktion in der 5. Rekursionsstufe gefunden wurde, die einen kleineren Trainings- und Validierungsfehler aufweist, als in der 4. Rekursionsstufe. Die Trainings- und Validierungsfehler vor und nach der Konstantenoptimierung bis zur 4. Rekursionsstufe sind in der Tabelle 9.3 dargestellt.

Nach der Konstantenopitimierung sind die Trainingsfehler E_{Train} und die Validierungsfehler E_{Valid} in den beiden Prädiktionsverfahren kleiner als vor der Konstantenoptimierung. Die Trainingsfehler E_{Train} und Validierungsfehler E_{Valid} des rekursiven Prädiktionsverfahren sind erst ab 3. Rekursionsstufe kleiner als die Trainingsfehler E_{Train} und Validierungsfehler E_{Valid} des einfachen Prädiktionsverfahrens.

Zum Schluss werden fünf unabhängige neue Datenreihen erzeugt, die durch die M-G-Gleichung beschrieben werden. Die Länge jeder Datenreihe beträgt 1000. Mit diesen Datenreihen wird die iterative Einschrittprädiktion durch die gefundene Prädiktionsfunktion aus den beiden Verfahren getestet. Die Fehler bei der iterativen

Rekursions-stufe	Konstanten-optimierung	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
1.	Vorher	$8,66 \cdot 10^{-2}$	$1,40 \cdot 10^{-1}$	-
	Nachher	$8,64 \cdot 10^{-2}$	$1,40 \cdot 10^{-1}$	-
2.	Vorher	$4,30 \cdot 10^{-3}$	$7,12 \cdot 10^{-3}$	-
	Nachher	$4,24 \cdot 10^{-3}$	$7,11 \cdot 10^{-3}$	-
3.	Vorher	$3,20 \cdot 10^{-3}$	$6,18 \cdot 10^{-3}$	-
	Nachher	$2,72 \cdot 10^{-3}$	$3,38 \cdot 10^{-3}$	-
4.	Vorher	$2,45 \cdot 10^{-3}$	$3,34 \cdot 10^{-3}$	-
	Nachher	$2,28 \cdot 10^{-3}$	$3,28 \cdot 10^{-3}$	$6,86 \cdot 10^{-3}$

Tabelle 9.3: Trainingsfehler E_{Train}, Validierungsfehler E_{Valid} vor und nach der Konstantenoptimierung und Testfehler E_{Test} am Ende der Versuche beim rekursiven Prädiktionsverfahren

Einschrittprädiktion werden mit Gleichung (8.6) berechnet. Die durchschnittlichen Prädiktionsfehler bei der iterativen Einschrittprädiktion E_{NMSE_k} aus den beiden Prädiktionsverfahren sind in Tabelle 9.4 dargestellt.

Der Prädiktionsfehler bei der iterativen Einschrittprädiktion aus dem rekursiven Prädiktionsverfahren ist kleiner als der Prädiktionsfehler aus dem einfachen Prädiktionsverfahren. Dennoch steigen die Prädiktionsfehler aus beiden Prädiktionsverfahren mit dem Prädiktionsschritt k an, d.h. die simulierte Funktion kann nur kurzfristig prädiktieren, aber sie ist nicht für die langfristige Prädiktion geeignet, weil die Fehler mit der Größe der Prädiktionsschritte zunehmen.

2. Situation: M-G-Gleichung mit 20 dB Störung

Die Trainingsfehler E_{Train}, Validierungsfehler E_{Valid} und Testfehler E_{Test} vor und nach der Konstantenoptimierung beim einfachen Prädiktionsverfahren sind in der Tabelle 9.5 dargestellt.

Die Trainings- und Validierungsfehler vor und nach der Konstantenoptimierung bis zur 5. Rekursionsstufe beim rekursiven Prädiktionsverfahren sind in Tabelle 9.6 dargestellt.

Die Ergebnisse in der 2. Situation (mit 20 dB Störung) sind ähnlich wie in der 1. Situation (ohne Störung). Nach der Konstantenopitimierung sind die Trainingsfehler E_{Train} und die Validierungsfehler E_{Valid} in den beiden Prädiktionsverfahren kleiner als vor der Konstantenoptimierung. Die Trainingsfehler E_{Train} und Validierungsfehler E_{Valid} des rekursiven Prädiktionsverfahrens sind erst ab 2. Rekursionsstufe kleiner als die Trainingsfehler E_{Train} und Validierungsfehler E_{Valid} des einfachen Prädiktionsverfahrens.

Prädiktions-schritt k	Prädiktionsfehler E_{NMSE_k} aus dem einfachen Prädiktionsverfahren	Prädiktionsfehler E_{NMSE_k} aus dem rekursiven Prädiktionsverfahren
1.	$5,92 \cdot 10^{-4}$	$3,20 \cdot 10^{-4}$
2.	$5,05 \cdot 10^{-2}$	$2,72 \cdot 10^{-3}$
3.	$1,89 \cdot 10^{-2}$	$1,03 \cdot 10^{-2}$
4.	$4,83 \cdot 10^{-2}$	$2,68 \cdot 10^{-2}$
5.	$9,91 \cdot 10^{-2}$	$5,49 \cdot 10^{-2}$
6.	$1,75 \cdot 10^{-1}$	$9,64 \cdot 10^{-2}$
7.	$2,79 \cdot 10^{-1}$	$1,52 \cdot 10^{-1}$
8.	$4,12 \cdot 10^{-1}$	$2,19 \cdot 10^{-1}$
9.	$5,70 \cdot 10^{-1}$	$2,99 \cdot 10^{-1}$
10.	$7,53 \cdot 10^{-1}$	$3,87 \cdot 10^{-1}$
11.	$9,57 \cdot 10^{-1}$	$4,83 \cdot 10^{-1}$
12.	1,18	$5,84 \cdot 10^{-1}$

Tabelle 9.4: Durchschnittliche Prädiktionsfehler bei der iterativen Einschrittprädiktion E_{NMSE_k} mit den Prädiktionsfunktionen aus dem einfachen Prädiktionsverfahren und dem rekursiven Prädiktionsverfahren bis zum 12. Prädiktionsschritt

Konstanten-optimierung	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
Vorher	1,56	2,04	-
Nachher	1,54	2,01	1,38

Tabelle 9.5: Trainingsfehler E_{Train}, Validierungsfehler E_{Valid} und Testfehler E_{Test} vor und nach der Konstantenoptimierung beim einfachen Prädiktionsverfahren (mit 20 dB Störung)

Rekursions-stufe	Konstanten-optimierung	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
1.	Vorher	1,58	2,16	-
	Nachher	1,57	2,16	-
2.	Vorher	1,52	1,95	-
	Nachher	1,48	1,94	-
3.	Vorher	1,42	1,87	-
	Nachher	1,37	1,79	-
4.	Vorher	1,25	1,64	-
	Nachher	1,25	1,62	-
5.	Vorher	1,20	1,54	-
	Nachher	1,19	1,53	1,33

Tabelle 9.6: Trainingsfehler E_{Train}, Validierungsfehler E_{Valid} und Testfehler E_{Test} vor und nach der Konstantenoptimierung beim rekursiven Prädiktionsverfahren (mit 20 dB Störung)

Prädiktions-schritt k	Prädiktionsfehler E_{NMSE_k} aus dem einfachen Prädiktionsverfahren	Prädiktionsfehler E_{NMSE_k} aus dem rekursiven Prädiktionsverfahren
1.	$1,90 \cdot 10^{-1}$	$1,61 \cdot 10^{-1}$
2.	$2,20 \cdot 10^{-1}$	$1,83 \cdot 10^{-1}$
3.	$2,68 \cdot 10^{-1}$	$2,15 \cdot 10^{-1}$
4.	$3,35 \cdot 10^{-1}$	$2,58 \cdot 10^{-1}$
5.	$4,06 \cdot 10^{-1}$	$2,95 \cdot 10^{-1}$
6.	$4,79 \cdot 10^{-1}$	$3,42 \cdot 10^{-1}$
7.	$5,54 \cdot 10^{-1}$	$3,94 \cdot 10^{-1}$
8.	$6,26 \cdot 10^{-1}$	$4,41 \cdot 10^{-1}$
9.	$6,93 \cdot 10^{-1}$	$4,89 \cdot 10^{-1}$
10.	$7,56 \cdot 10^{-1}$	$5,32 \cdot 10^{-1}$
11.	$8,13 \cdot 10^{-1}$	$5,79 \cdot 10^{-1}$
12.	$8,61 \cdot 10^{-1}$	$6,20 \cdot 10^{-1}$

Tabelle 9.7: Durchschnittliche Prädiktionsfehler bei der iterativen Einschrittprädiktion E_{NMSE_k} mit den Prädiktionsfunktionen aus dem einfachen Prädiktionsverfahren und dem rekursiven Prädiktionsverfahren (mit 20 dB Störung) bis zum 12. Prädiktionsschritt

Zum Schluss werden fünf unabhängige neue Datenreihen, die durch die M-G-Gleichung beschrieben werden, mit 20 dB Störung erzeugt. Die Länge jeder Datenreihe beträgt 1000. Mit diesen Datenreihen wird die iterative Einschrittprädiktion mit Hilfe der gefundenen Prädiktionsfunktion aus den beiden Verfahren durchgeführt. Die Fehler bei der iterativen Einschrittprädiktion werden mit Gleichung (8.6) berechnet. Die durchschnittlichen Prädiktionsfehler bei der iterativen Einschrittprädiktion E_{NMSE_k} aus den beiden Prädiktionsverfahren sind in Tabelle 9.7 dargestellt.

Diese Ergebnisse sind ähnlich wie in der 1. Situation (ohne Störung). Der Prädiktionsfehler aus dem rekursiven Prädiktionsverfahren ist kleiner als der Prädiktionsfehler aus dem einfachen Prädiktionsverfahren. Außerdem steigt er mit dem Prädiktionsschritt k an.

9.2 Weitere Experimente

Im zweiten Teil werden weitere Experimente durchgeführt, um den effizienten Einsatz unseres Prädiktionsverfahrens zu zeigen. Die folgenden chaotischen Zeitreihen werden bei den Experimenten verwendet:

- die durch die logistische Abbildung in Gleichung (6.11) beschriebenen chaotischen Zeitreihen ohne und mit der Überlagerung der Störung bei SNR = 30, 20, 10 dB,
- die monatliche durchschnittliche Sonnenfleckenrelativzahl ohne und mit Störungsreduzierung,
- die jährliche durchschnittliche Sonnenfleckenrelativzahl ohne und mit Störungsreduzierung und
- der Wechselkurs zwischen Euro € und Baht ฿ (die thailändische Währung) ohne und mit Störungsreduzierung.

Die ersten drei chaotischen Zeitreihen sind bereits in Kapitel 6 erwähnt. Der Wechselkurs von Euro € in Baht ฿ wird später im Abschnitt 9.2.4 beschrieben. Bei jeder chaotischen Zeitreihe versuchen wir entweder mit dem einfachen Prädiktionsverfahren oder mit dem rekursiven Prädiktionsverfahren eine Prädiktionsfunktion durch Lernen aus den Trainingsdaten zu entwickeln.

Für die letzten drei chaotischen Zeitreihen wird vermutet, dass diese chaotischen Zeitreihen von Rauschen überlagert sind. Daher versuchen wir eine Prädiktionsfunktion nicht nur allein durch das Lernen der chaotischen Zeitreihen zu entwickeln, sondern auch durch das Lernen der chaotischen Zeitreihen nach der Störungsreduzierung. Die Störungsreduzierung kann durch eine Filterung mit einem Tiefpass oder einer Mittelwertsberechung durchgeführt werden.

Zur Vermeidung des Overfittings wird das frühe Stoppen wie im ersten Teil eingesetzt. Die Schätzung des Generalisierungsfehlers erfolgt nach der Teilung der Trainingsdaten. Daher entspricht der Generalisierungsfehler dem Validierungsfehler. Da der Verlauf des Validierungsfehlers mehrere lokale Minima hat bevor das endgültige Overfitting einsetzt, wird das Training nur gestoppt, wenn die Maximalzahl von Trainingsschritten erreicht wird. Das beste gefundene Ergebnis ist das Ergebnis mit dem niedrigsten Validierungsfehler.

In den letzten drei Experimenten wurden dabei die GP in [Wein97] und GA in [Houck95] eingesetzt. Um Ergebnisse des Experimentes leicht zu verstehen, wird die mit den originalen Trainingsdaten entwickelte Prädiktionsfunktion als f_{sum_I} und die mit den Trainingsdaten nach Störungsreduzierung entwickelte Prädiktionsfunktion als f_{sum_II} bezeichnet.

9.2.1 Logistische Abbildung

Die Prädiktion der durch die logistische Abbildung:

$$x_{t+1} = 4 \cdot x_t \cdot (1 - x_t) \quad \text{mit} \quad t \in \mathbb{Z} \tag{9.7}$$

beschriebenen chaotischen Zeitreihe ist eine sehr leichte Aufgabe. Sie wird als nächste Simulation dargestellt. Dabei wird nur das einfache Prädiktionsverfahren mit der ersten Möglichkeit zum Einfügen der Koeffizienten verwendet, d.h. Einfügen von nur einem Koeffizienten in die mathematische Funktion. Zur Konstantenoptimierung wird das Least-squares-Verfahren eingesetzt.

Die Simulationen werden bei vier Situationen durchgeführt.

1. **Situation:** Simulation mit den durch die logistische Abbildung beschriebenen Daten.

2. **Situation:** Simulation mit den durch die logistische Abbildung beschriebenen Daten, jedoch werden diese von weißem Rauschen mit SNR = 30 dB überlagert.

3. **Situation:** Simulation mit den durch die logistische Abbildung beschriebenen Daten, jedoch werden diese von weißem Rauschen mit SNR = 20 dB überlagert.

4. **Situation:** Simulation mit den durch die logistische Abbildung beschriebenen Daten, jedoch werden diese von weißem Rauschen mit SNR = 10 dB überlagert.

Einstellung des Experiments

Zum Beginn der Simulation wurde die logistische Abbildung mit einem bestimmten Anfangswert gestartet, danach werden weitere Daten erzeugt. Damit die Anfangswerte keinen Einfluss auf die Daten haben, wurden zuerst 1000 Daten berechnet. Nach der Erzeugung der ersten 1000 Daten wurden weiteren 400 Daten berechnet. Die letzten 300 Daten von diesen 400 Daten wurden bei der Simulation verwendet. Dabei sind die Daten in drei Mengen wie folgt aufgeteilt:

- Die Daten von der 101. Position (x_{101}) bis zur 200. Position (x_{200}) bilden die Trainingsmenge $\mathbf{S}_{\text{Train}}$.
- Die Daten von der 201. Position (x_{201}) bis zur 300. Position (x_{300}) bilden die Validierungsmenge $\mathbf{S}_{\text{Valid}}$.
- Die Daten von der 301. Position (x_{301}) bis zur 400. Position (x_{400}) bilden die Testmenge $\mathbf{S}_{\text{Test}}$.

Wichtige Bedingungen des GP-Laufes sind folgende:

Terminalmenge und Funktionenmenge: Die folgende Funktionenmenge $\mathcal{F}_1$ wird bei den Simulationen verwendet:

$$\mathcal{F}_1 = \{\sin, \cos, \log, \exp, +, -, \times, \div\} \tag{9.8}$$

Die verwendete Terminalmenge $\mathcal{T}_1$ ist:

$$\begin{aligned} \mathcal{T}_1 \;=\; & \{x_t, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}, \\ & -10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0\} \end{aligned} \tag{9.9}$$

wobei: x_t der Wert zum Zeitpunkt t, x_{t-1} der Wert zum Zeitpunkt $t-1$ usw. ist. $-10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0$ sind die einfachen konstanten Werte im Bereich von $-10,0$ bis $10,0$.

Fitnessfunktion: Die Rohfitness f_{r} wird folgendermaßen berechnet:

$$f_{\text{r}} = \frac{1}{\sigma^2_{\text{Train}} \cdot N_{\text{Train}}} \sum_{t=101}^{200} (x_t - \hat{x}_t)^2 \tag{9.10}$$

Dabei ist:

- x_t der tatsächliche Wert zum Zeitpunkt t,
- $\hat{x}_t$ der prädiktierte Wert zum Zeitpunkt t,

- σ_{Train} die Standardabweichung des tatsächlichen Wertes, $x_{101}, x_{102}, x_{103}, ..., x_{200}$,
- N_{Train} die Anzahl der Daten im berechneten Bereich, hier $N_{\text{Train}} = 100$.

Die Rohfitness f_{r} entspricht dem Prädiktionsfehler einer Zeitreihe E_{NMSE_1} aus Gleichung (8.6) in der Trainingsmenge $\mathbf{S}_{\text{Train}}$, deshalb kann man diese Rohfitness f_{r} als Trainingsfehler E_{Train} bezeichnen.

Zur Vermeidung des Overfittings wird der Validierungsfehler E_{Valid} beim GP-Lauf und bei der Konstantenoptimierung berücksichtigt. Der Validierungsfehler E_{Valid} wird also mit der Gleichung (8.6) berechnet, wenn der Prädiktionsschritt eins ist.

Der Prädiktionsfehler außerhalb der Trainingsdaten oder der Testfehler E_{Test} wurde ebenfalls mit der Gleichung (8.6) berechnet, wenn der Prädiktionsschritt eins ist.

Einstellung der GP-Parameter und Abbruchkriterium: Die Einstellungen der Parameter der GP sind in der Tabelle 9.8 dargestellt. Der GP-Lauf wird nur gestoppt, wenn die Anzahl der maximalen Generation G_{max} erreicht ist.

GP-Parameter	Einstellung
Anzahl der GP-Läufe RUN_{max}	10
Populationsgröße M	300
Max. Generation G_{max}	50
Kreuzungswahrscheinlichkeit p_{c}	0,9
Reproduktionswahrscheinlichkeit p_{r}	0,1
Mutationswahrscheinlichkeit p_{m}	0,2
Max. Tiefe der Ausgangspopulation D_{initial}	6
Max. Tiefe während des restl. Laufes D_{created}	17
Initialisierungsmethode	*half-ramping*
Selektionsverfahren	Fitnessproportional
Elitismus	Ja

Tabelle 9.8: Einstellungen der GP-Parameter

Simulationsergebnisse

Die logistische Abbildung ist eine eindimensionale diskrete Abbildung. Sie ist daher leicht zu prädiktieren. In diesem Abschnitt werden die besten Ergebnisse aus zehn GP-Läufen in jeder Situation präsentiert.

Tabelle 9.9 zeigt die Prädiktionsfehler der besten Prädiktionsfunktion aus den vier Situationen.

Situation von Daten	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
Keine Störung	$4,38 \cdot 10^{-8}$	$5,20 \cdot 10^{-8}$	$7,13 \cdot 10^{-8}$
30 dB	$2,47 \cdot 10^{-2}$	$2,09 \cdot 10^{-2}$	$2,30 \cdot 10^{-2}$
20 dB	$1,45 \cdot 10^{-1}$	$1,53 \cdot 10^{-1}$	$2,05 \cdot 10^{-1}$
10 dB	$5,98 \cdot 10^{-1}$	$7,10 \cdot 10^{-1}$	$9,37 \cdot 10^{-1}$

Tabelle 9.9: Prädiktionsfehler der Prädiktionsfunktion aus den vier Situationen

Je größer das SNR ist, desto höher ist die Möglichkeit zur Prädiktion. Besonders bei keiner Störung sind die Fehler in der Training-, Validierung- und Testmenge deutlich kleiner.

Da alle besten Prädiktionsfunktionen nur eine Variable x_t besitzen, kann der (x_t-x_{t+1})-Graph oder das Zustandsdiagramm für jede Prädiktionsfunktion dargestellt werden. Abbildung 9.1 stellt den (x_t-x_{t+1})-Graph jeder Prädiktionsfunktion dar.

Trotz einer Störung in der chaotischen Zeitreihe ist GP noch fähig, bis zum SNR von 20 dB, die geeignete Prädiktionsfunktion zu entwickeln. Auf Grund der niedrigen Anzahl von Trainingsdaten ist es nicht möglich, gute Koeffizienten und Konstanten durch die Konstantenoptimierung zu finden.

Die Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktion der Prädiktionsfunktionen in jeder Situation sind in Abbildung 9.2 dargestellt. Je stärker die übergelagerte Störung ist, desto schneller steigt der Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktion an.

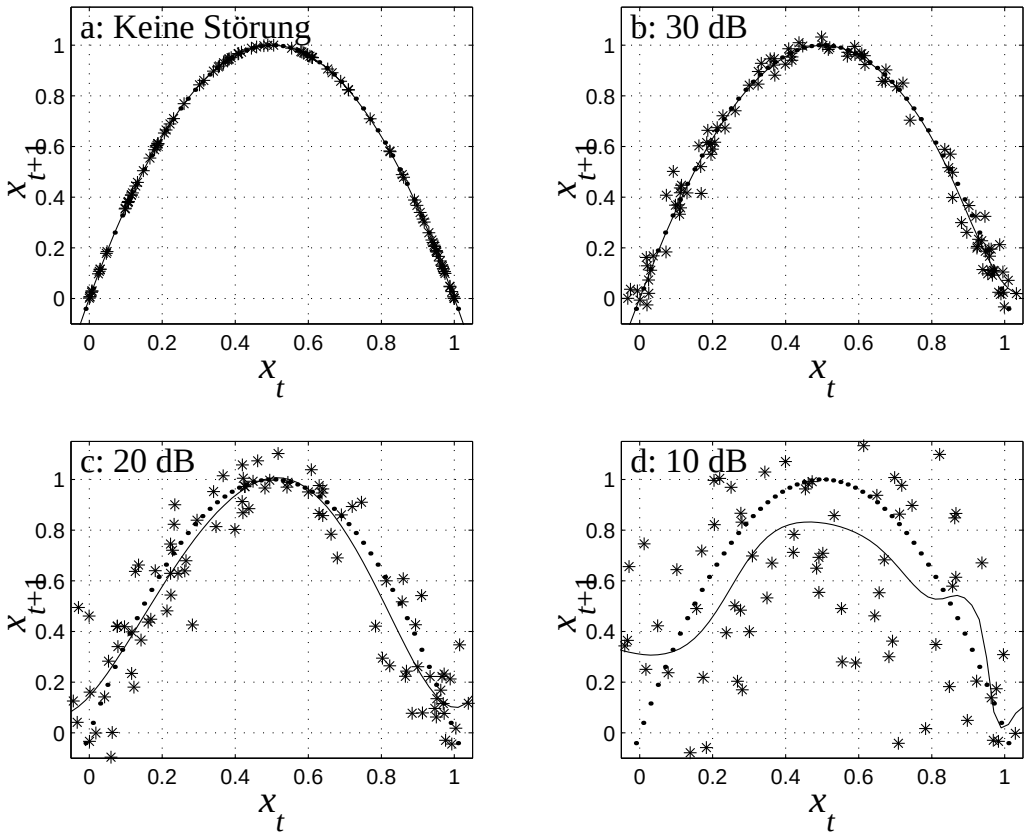


Abbildung 9.1: (x_t, x_{t+1})-Graph oder Zustandsdiagramm der Prädiktionsfunktionen aus den vier Situationen, wobei: $\cdots$ tatsächliches Zustandsdiagramm, $-$ Zustandsdiagramm der Prädiktionsfunktion und $***$ Verlauf der Trainingsdaten ist.

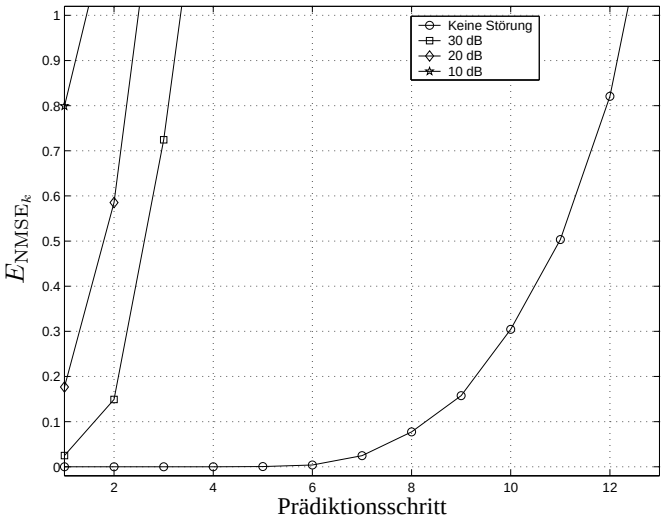


Abbildung 9.2: Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktionen der logistischen Abbildung bei den vier verschiedenen Störungen (keine Störung, Situationen mit SNR $= 30, 20, 10$ dB).

9.2.2 Monatliche durchschnittliche Sonnenfleckenrelativzahl

Bei dem zweiten Experiment wird versucht, die monatliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s}_{\mathrm{month}}}$ mit dem rekursiven Prädiktionsverfahren zu prädiktieren.

Außerdem wird bei diesem Experiment versucht, die monatliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s}_{\mathrm{month}}}$ nach der Störungsreduzierung durch die Filterung mit einem Tiefpass zu prädiktieren. Der eingesetzte Tiefpass ist ein digitales Butterworth-Filter zweiter Ordnung, dessen Grenzfrequenz $f_{\mathrm{c}} = 1/6$ Jahr^{-1} ist. Die gefilterte Zeitreihe ist in Abbildung 9.3 dargestellt.

Einstellung des Experiments

In diesem Experiment wird das rekursive Prädiktionsverfahren mit fünf Rekursionsstufen verwendet. Zur Konstantenoptimierung wird GA eingesetzt. Dabei wird die erste Möglichkeit zur Einfügung der Koeffizienten verwendet. Die Schätzung des Generalisierungsfehlers erfolgt nach der Teilung der Trainingsdaten. Die Teilung von Daten in Training-, Validierungs- und Testmenge ist in Tabelle 9.10 gezeigt.

Wichtige Bedingungen des GP-Laufes sind folgende:

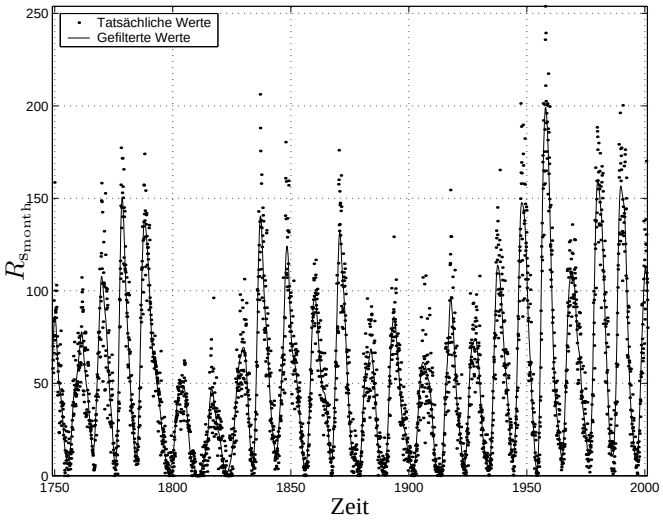


Abbildung 9.3: Monatliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ vor und nach der Störungsreduzierung

Mengentypen	Daten
Trainingsmenge	von 10/1956 bis 09/1986
Validierungsmenge	von 10/1986 bis 09/1996
Testmenge	von 10/1996 bis 02/2001

Tabelle 9.10: Die Aufteilung der Daten in Training-, Validierungs- und Testmenge bei dem Experiment mit der monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$

Terminalmenge und Funktionenmenge: Die folgende Funktionenmenge $\mathcal{F}_2$ wird bei den Simulationen verwendet:

$$\mathcal{F}_2 = \{\sin, \cos, \log, \exp, +, -, \times, \div\} \tag{9.11}$$

Die Sonnenfleckenrelativzahl steigt und fällt durchschnittlich alle ca. 11 Jahre, die monatliche durchschnittliche Sonnenfleckenrelativzahl wird daher bis zu einem Zeitpunkt, der kleiner als $t-(12\cdot 11)$ ist, benötigt. Daraus folgt, dass für die verwendete Terminalmenge $\mathcal{T}_2$ gilt:

$$\begin{aligned}\mathcal{T}_2 \;=\; \{&x_t, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}, x_{t-5}, x_{t-6}, x_{t-7}, x_{t-8}, x_{t-9},\\ &x_{t-10}, x_{t-11}, x_{t-12}, x_{t-13}, x_{t-14}, x_{t-15}, x_{t-16}, x_{t-17}, x_{t-18},\\ &x_{t-19}, x_{t-29}, x_{t-49}, x_{t-69}, x_{t-89}, x_{t-109}, x_{t-129}, x_{t-149},\\ &-10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0\}\end{aligned} \tag{9.12}$$

wobei: x_t der Wert zum Zeitpunkt t, x_{t-1} der Wert zum Zeitpunkt $t-1$ usw. ist. $-10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0$ sind die einfachen konstanten Werte im Bereich von $-10,0$ bis $10,0$.

Fitnessfunktion: Ähnlich wie bei anderen Experimenten werden der Trainingsfehler E_{Train} (entspricht der Rohfitness f_{r}), der Validierungsfehler E_{Valid} und der Testfehler E_{Test} mit Gleichung (8.6) mit dem Prädiktionsschritt von eins berechnet.

Einstellung der GP-Parameter und Abbruchkriterium Die Einstellungen der Parameter der GP sind in der Tabelle 9.11 dargestellt.

GP-Parameter	Einstellung
Anzahl der Rekursionsstufen N_{r}	5
Anzahl der GP-Läufe in einer Rekursionsstufe $RUN_{\max}$	50
Populationsgröße M	500
Max.Generation $G_{\max}$	50
Kreuzungswahrscheinlichkeit p_{c}	0,9
Reproduktionswahrscheinlichkeit p_{r}	0,1
Mutationswahrscheinlichkeit p_{m}	0,2
Max. Tiefe der Ausgangspopulation D_{initial}	6
Max. Tiefe während des restl. Laufes D_{created}	17
Initialisierungsmethode	*half-ramping*
Selektionsverfahren	Wettkampfselektion
Wettkampfumfang ξ	10
Elitismus	Ja

Tabelle 9.11: Einstellungen der GP-Parameter

In jeder Rekursionsstufe werden 50 GP-Läufe durchgeführt. Nur die beste durch GP entwickelte Prädiktionsfunktion wird ausgewählt und anschließend zur Konstantenoptimierung benutzt. Der GP-Lauf wird nur gestoppt, wenn die Anzahl der maximalen Generation $G_{\max}$ erreicht ist.

Simulationsergebnisse

Obwohl die Anzahl der Rekursionsstufe von fünf eingestellt ist, wurde das Training mit den Trainingsdaten nach der Störungsreduzierung in der 4. Rekursionsstufe gestoppt, da keine gute Sub-Prädiktion in der 4. Rekursionsstufe gefunden wurde, die den Trainings- und Validierungsfehler kleiner als den Trainings- und Validierungsfehler der 3. Rekursionsstufe macht. Allerdings sind der Trainings- und Validierungsfehler in der 3. Rekursionsstufe bereits extrem klein. Das Training mit originalen Trainingsdaten wurde dagegen nach der 5. Rekursionsstufe gestoppt.

Die Prädiktionsfehler der entwickelten Prädiktionsfunktionen aus den beiden Trainingssituationen sind in Tabelle 9.12 gezeigt, wenn sie die originalen Daten prädiktieren.

Situation der Trainingsdaten	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
$f_{\text{sum}_{\text{I}}}$	$4,77 \cdot 10^{-2}$	$7,54 \cdot 10^{-2}$	$23,24 \cdot 10^{-2}$
$f_{\text{sum}_{\text{II}}}$	$7,94 \cdot 10^{-2}$	$9,20 \cdot 10^{-2}$	$16,10 \cdot 10^{-2}$
	$(1,51 \cdot 10^{-6})$	$(1,30 \cdot 10^{-6})$	$(4,38 \cdot 10^{-6})$

Tabelle 9.12: Prädiktionsfehler der Prädiktionsfunktion in den beiden Situationen der Trainingsdaten, wobei: $f_{\text{sum}_{\text{I}}}$ die mit den originalen Trainingsdaten entwickelte Prädiktionsfunktion und $f_{\text{sum}_{\text{II}}}$ die mit den Trainingsdaten nach Störungsreduzierung entwickelte Prädiktionsfunktion ist. Die in runde Klammern gesetzten Werte sind die Prädiktionsfehler der Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$ für die Daten nach der Störungsreduzierung.

Abbildung 9.4 stellt den Vergleich zwischen den prädiktierten Werten der monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\text{s}_{\text{month}}}$ durch die Einschrittprädiktion und den tatsächlichen Werten dar.

Der Trainingsfehler und der Validierungsfehler der mit den Trainingsdaten nach Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$ sind größer als die Fehler der mit den originalen Trainingsdaten entwickelten Prädiktionsfunktion $f_{\text{sum}_{\text{I}}}$. Jedoch ist der Fehler außerhalb der Trainingsdaten der Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$, der sogenannte Testfehler, deutlich kleiner. Außerdem ist die Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$ sehr geeignet für die iterative Einschrittprädiktion.

Abbildung 9.5 zeigt Beispiele der iterativen Einschrittprädiktionen von beiden Prädiktionsfunktionen. Die Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktion von beiden Prädiktionsfunktionen sind in Abbildung 9.6 dargestellt. Bis zum 15. Prädiktionsschritt ist der Prädiktionsfehler der Prädiktionsfunktion $f_{\text{sum}_{\text{I}}}$ kleiner als eins. Dagegen bleibt der Prädiktionsfehler der Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$ unter eins bis zum 35. Prädiktionsschritt.

Die Sonnenflecken sind ein Naturphänomen. Anscheinend kann die Sonnenfleckenrelativzahl durch die Genauigkeit der Messung in verschiedenen Zeiten beeinflusst werden. Daher ist die iterative Einschrittprädiktionen im Nahbereich der Trainingsdaten deutlich besser als die iterative Einschrittprädiktion im gesammten Bereich der Daten.

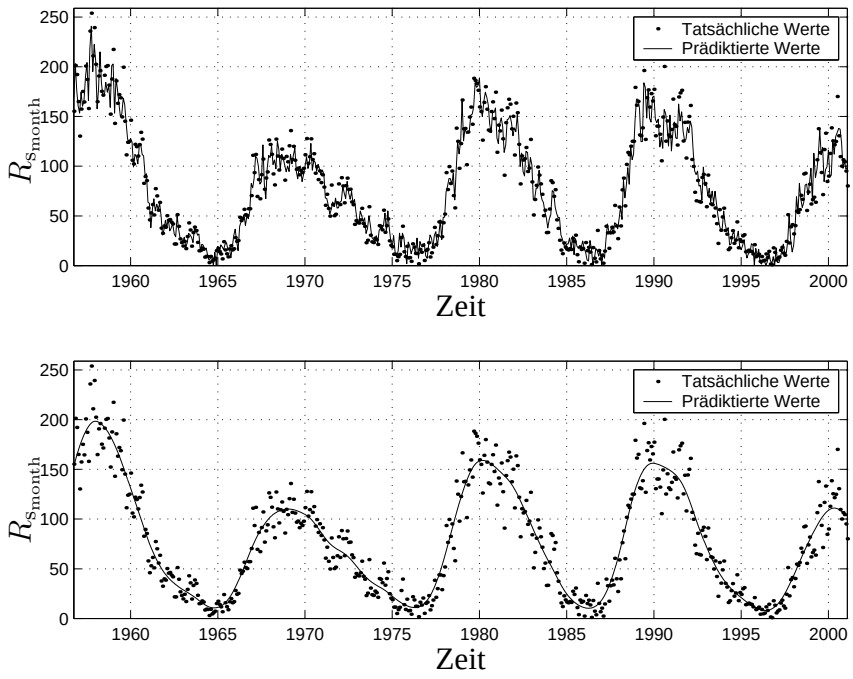


Abbildung 9.4: Vergleich zwischen den prädiktierten Werten der monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ durch die Einschrittprädiktion und den tatsächlichen Werten. *Oben:* Prädiktion durch die Prädiktionsfunktion $f_{\mathrm{sum_I}}$, *Unten:* Prädiktion durch die Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$.

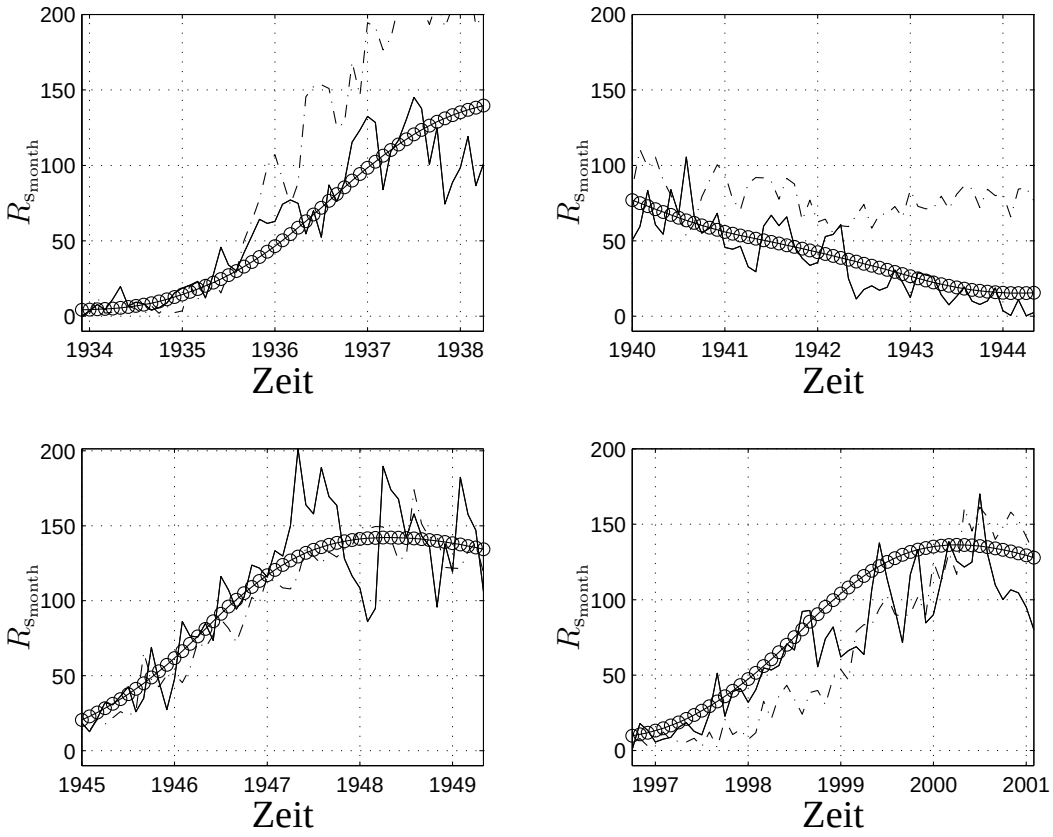


Abbildung 9.5: Beispiele der iterativen Einschrittprädiktionen der monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ außerhalb der Trainingsdaten, wobei: $-$ die tatsächlichen Werte, $-\cdot$ die prädiktierten Werte der mit den originalen Trainingsdaten entwickelten Prädiktionsfunktion $f_{\mathrm{sum_I}}$ und $\ominus$ die prädiktierten Werte der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ sind.

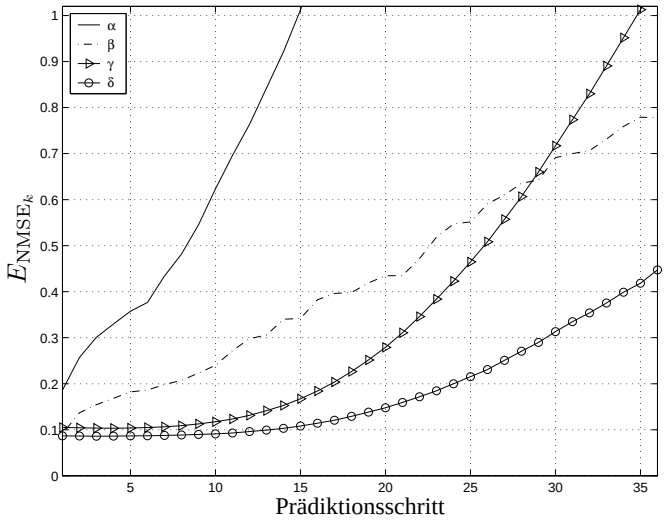


Abbildung 9.6: Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktionen der monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$, wobei: α und β der Prädiktionsfehler der mit den originalen Trainingsdaten entwickelte Prädiktionsfunktion $f_{\mathrm{sum_I}}$ in dem gesamten Bereich der Daten (von 01.1749 bis 02.2001) und im Nahbereich der Trainingsdaten (von 09.1918 bis 02.2001), γ und δ der Prädiktionsfehler der mit den Trainingsdaten nach der Störungsreduzierung entwickelte Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ im gesamten Bereich der Daten (von 01.1749 bis 02.2001) und im Nahbereich der Trainingsdaten (von 09.1918 bis 02.2001) ist.

9.2.3 Jährliche durchschnittliche Sonnenfleckenrelativzahl

Bei dem dritten Experiment wird versucht, die jährliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ mit dem rekursiven Prädiktionsverfahren zu prädiktieren.

Ähnlich wie beim vorherigen Experiment wird bei diesem Experiment versucht, die jährliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ nach der Störungsreduzierung zu prädiktieren. Die jährliche durchschnittliche Sonnenfleckenrelativzahl wird durch die folgende Gleichung gefiltert:

$$\bar{R}_{\mathrm{s_{year},}t} = \frac{1}{3}(R_{\mathrm{s_{year},}t-1} + R_{\mathrm{s_{year},}t} + R_{\mathrm{s_{year},}t+1}) \tag{9.13}$$

Wobei:

- $\bar{R}_{\mathrm{s_{year},}t}$ der gefilterte Wert der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ nach Störungsreduzierung im Jahr t,
- $R_{\mathrm{s_{year},}t}$ die jährliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ im Jahr t, usw. ist.

Die gefilterten Werte der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $\bar{R}_{t,\mathrm{s_{year}}}$ in Gleichung (9.13) sind eigentlich der Mittelwert aus der jährlichen durchschnittlichen Sonnenfleckenrelativzahl im Jahr t und ihren nächsten Nachbarn $R_{\mathrm{s_{year}},t-1}$ und $R_{\mathrm{s_{year}},t+1}$.

Der Wert der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ ist selbst ein Mittelwert, der aus den Sonnenfleckenrelativzahlen innerhalb eines Jahres berechnet wird. Dadurch wird der große Anteil der Störung in der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ reduziert. Ob die Störungsreduzierung durch die Filterung mit einem Tiefpass, oder ob die Störungsreduzierung durch die Berechnung des Mittelwerts mit den Nachbarn erfolgt: man bekommt ähnliche Ergebnisse. Abbildung 9.7 stellt die jährliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ vor und nach der Störungsreduzierung dar.

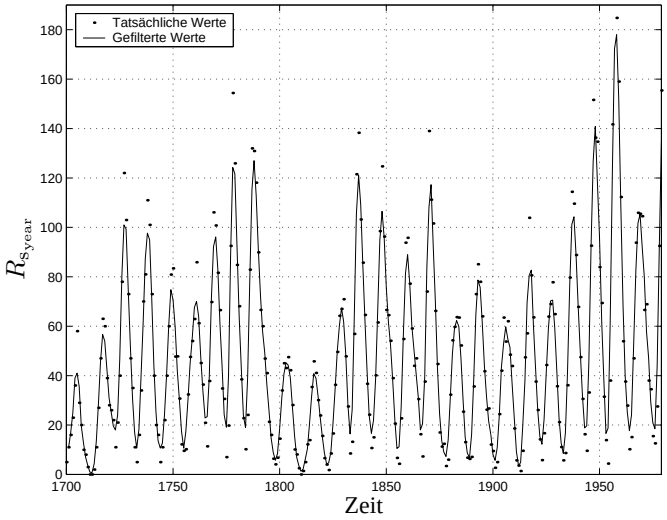


Abbildung 9.7: Jährliche durchschnittliche Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ vor und nach der Störungsreduzierung

Einstellung des Experiments

In diesem Experiment wird das rekursive Prädiktionsverfahren mit fünf Rekursionsstufen verwendet. Zur Konstantenoptimierung wird GA eingesetzt. Dabei wird die erste Möglichkeit zur Einfügung der Koeffizienten verwendet. Die Schätzung des Generalisierungsfehlers erfolgt nach der Teilung der Trainingsdaten. Die Teilung von Daten in Training-, Validierungs- und Testmenge ist in Tabelle 9.13 gezeigt.

Wichtige Bedingungen des GP-Laufes sind folgende:

Terminalmenge und Funktionenmenge: Die folgende Funktionenmenge $\mathcal{F}_3$ wird bei den Simulationen verwendet:

$$\mathcal{F}_3 = \{\sin, \cos, \log, \exp, +, -, \times, \div\} \tag{9.14}$$

Mengentypen	Daten
Trainingsmenge	von 1700 bis 1920
Validierungsmenge	von 1921 bis 1950
Testmenge	von 1951 bis 1979

Tabelle 9.13: Die Teilung der Daten in Training-, Validierungs- und Testmenge bei dem Experiment mit der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$

Die Sonnenfleckenrelativzahl steigt und fällt durchschnittlich ca. alle 11 Jahre, es werden daher die jährlichen durchschnittlichen Sonnenfleckenrelativzahlen $R_{\mathrm{s_{year}}}$ bis zu einem Zeitpunkt, der kleiner als $t-11$ ist benötigt. Daraus folgt, dass die verwendete Terminalmenge $\mathcal{T}_3$ ist:

$$\begin{aligned}\mathcal{T}_3 \;=\; & \{x_t, x_{t-1}, x_{t-3}, x_{t-5}, x_{t-7}, x_{t-9}, x_{t-11}, x_{t-13}, \\ & -10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0\}\end{aligned} \tag{9.15}$$

wobei: x_t der Wert zum Zeitpunkt t, x_{t-1} der Wert zum Zeitpunkt $t-1$ usw. ist. $-10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0$ sind die einfachen konstanten Werte im Bereich von $-10,0$ bis $10,0$.

Fitnessfunktion: Ähnlich wie bei den anderen Experimenten werden der Trainingsfehler E_{Train} (entspricht der Rohfitness f_{r}), der Validierungsfehler E_{Valid} und der Testfehler E_{Test} durch Gleichung (8.6) mit dem Prädiktionsschritt von eins berechnet.

Einstellung der GP-Parameter und Abbruchkriterium: Die Parameter der GP sind identisch mit denen des vorherigen Versuches, gemäß Tabelle 9.11 (mit der Populationsgröße $M = 250$).

In jeder Rekursionsstufe werden 50 GP-Läufe durchgeführt. Nur die beste durch GP entwickelte Prädiktionsfunktion wird ausgewählt und anschließend zur Konstantenoptimierung benutzt. Der GP-Lauf wird nur gestoppt, wenn die Anzahl der maximalen Generation $G_{\max}$ erreicht ist.

Simulationsergebnisse

Obwohl die Rekursionsstufe von fünf eingestellt ist, wurde das Training mit den Trainingsdaten nach der Störungsreduzierung in der 3. Rekursionsstufe gestoppt, da keine gute Sub-Prädiktion in der 3. Rekursionsstufe gefunden wird, die den Trainings- und Validierungsfehler kleiner als Trainings- und Validierungsfehler in der 2. Rekursionsstufe macht. Das Training mit originalen Trainingsdaten wurde dagegen nach der 5. Rekursionsstufe gestoppt.

Die Prädiktionsfehler der beiden entwickelten Prädiktionsfunktionen sind in Tabelle 9.14 gezeigt, wenn sie die originalen Daten prädiktieren.

Situation der Trainingsdaten	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
f_{sum_I}	$7,22 \cdot 10^{-2}$	$3,77 \cdot 10^{-2}$	$5,26 \cdot 10^{-2}$
f_{sum_II}	$3,49 \cdot 10^{-2}$	$3,49 \cdot 10^{-2}$	$3,80 \cdot 10^{-2}$
	$(2,38 \cdot 10^{-2})$	$(6,20 \cdot 10^{-2})$	$(2,01 \cdot 10^{-2})$

Tabelle 9.14: Prädiktionsfehler der Prädiktionsfunktion in den beiden Situationen der Trainingsdaten, wobei: f_{sum_I} die mit den originalen Trainingsdaten entwickelte Prädiktionsfunktion und f_{sum_II} die mit den Trainingsdaten nach Störungsreduzierung entwickelte Prädiktionsfunktion ist. Die in runde Klammern gesetzten Werte sind die Prädiktionsfehler der Prädiktionsfunktion f_{sum_II} für die Daten nach der Störungsreduzierung.

Abbildung 9.8 stellt den Vergleich zwischen den prädiktierten Werten der jährlichen durchschnittlichen Sonnenfleckenrelativzahl R_{s_year} durch die Einschrittprädiktion und die tatsächlichen Werte dar. Der Trainings-, Validierungs- und Testfehler der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion f_{sum_II} sind kleiner als die Fehler der mit den originalen Trainingsdaten entwickelten Prädiktionsfunktion f_{sum_I}.

Außerdem ist die Prädiktionsfunktion f_{sum_II} sehr geeignet für die iterative Einschrittprädiktion. Abbildung 9.9 zeigt Beispiele der iterativen Einschrittprädiktion von beiden Prädiktionsfunktionen. Die Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktion von beiden Prädiktionsfunktionen sind in Abbildung 9.10 dargestellt. Anscheinend sind die origianlen Daten und die Daten nach der Störungsreduzierung nur wenig unterschiedlich, daher ist die Störungsreduzierung in diesem Experiment nicht so nützlich wie im vorherigen Experiment.

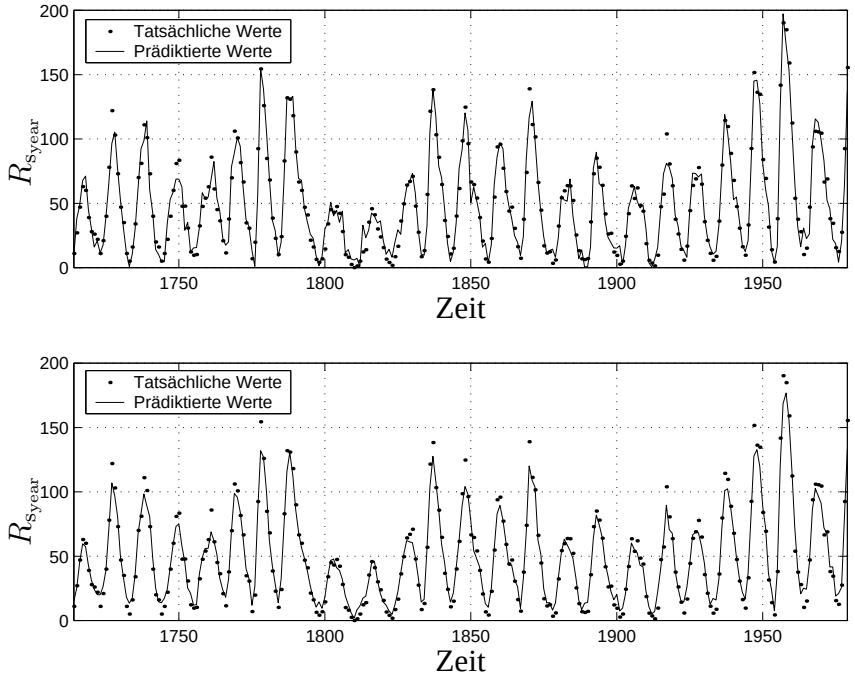


Abbildung 9.8: Vergleich zwischen den prädiktierten Werten der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{S_{year}}}$ durch die Einschrittprädiktion und den tatsächlichen Werten, *Oben:* Prädiktion durch die Prädiktionsfunktion $f_{\mathrm{sum_I}}$, *Unten:* Prädiktion durch die Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$.

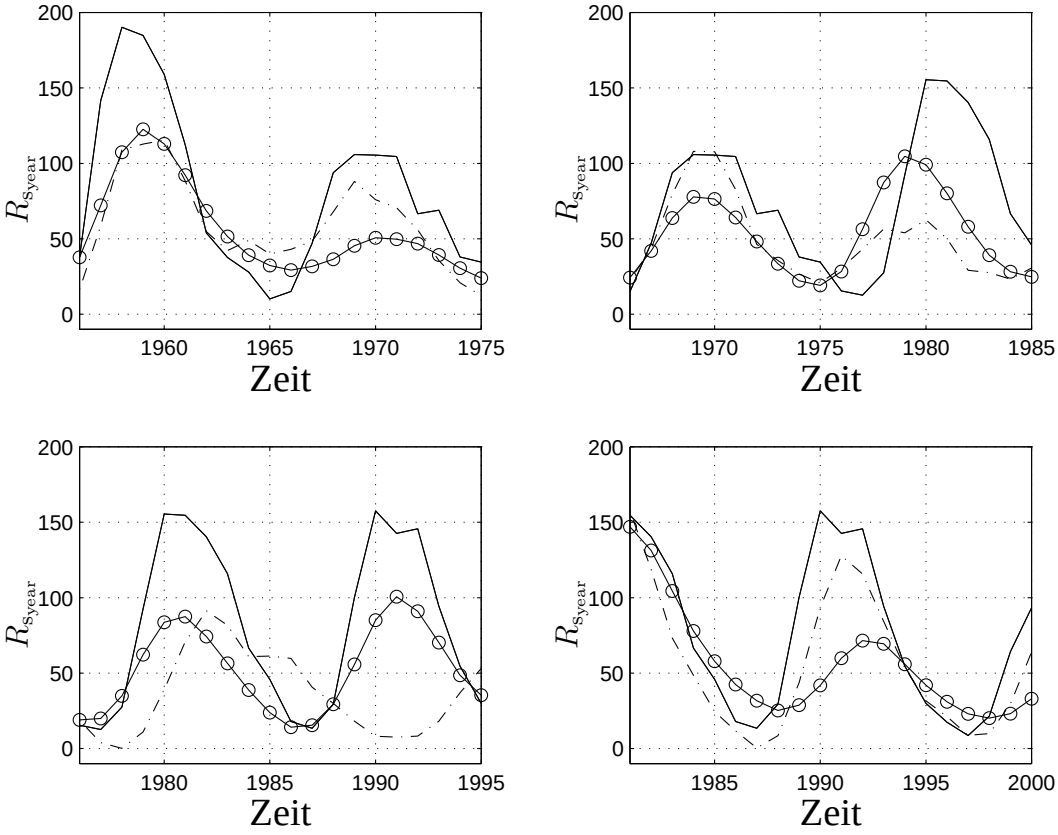


Abbildung 9.9: Beispiele der iterativen Einschrittprädiktion der jährlichen durchschnittlichen Sonnenfleckenrelativzahl R_{Syear} außerhalb der Trainingsdaten, wobei: $-$ die tatsächlichen Werte, $-\cdot$ die prädiktierten Werte der mit den originalen Trainingsdaten entwickelten Prädiktionsfunktion $f_{\mathrm{sum_I}}$ und $\ominus$ die prädiktierten Werte der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ sind.

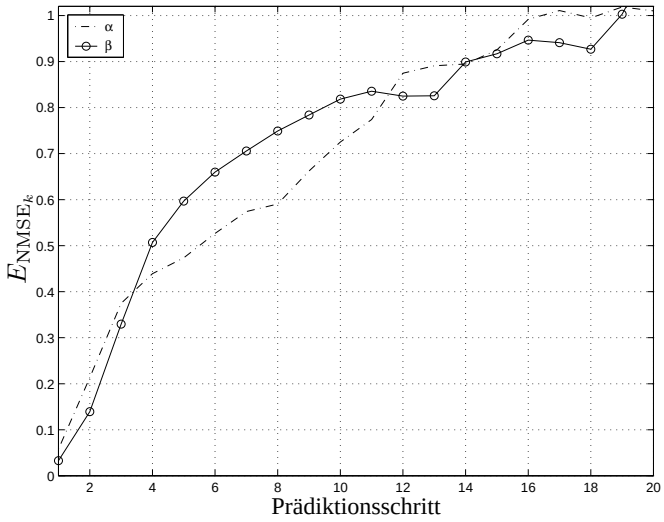


Abbildung 9.10: Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktion der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{S_{year}}}$ in dem gesamten Bereich der Daten (von 1700 bis 2000), wobei: α der Prädiktionsfehler der mit den originalen Trainingsdaten entwickelten Prädiktionsfunktion $f_{\mathrm{sum_I}}$, und β der Prädiktionsfehler der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ ist.

9.2.4 Wechselkurs: Euro € und Baht ฿

Das letzte Experiment ist die Prädiktion eines Wechselkurses. Der Wechselkurs ist eine Art wirtschaftliche Zeitreihe. Abbildung 9.11 (oben) zeigt den täglichen durchschnittlichen Verkaufspreis eines Euro € in Baht ฿ vom 2.01.2001 bis 15.08.2001. Die Zeitreihe umfasst die Daten aller Börsentage. Sie wurde von der Zentralbank in Thailand[1] gesammelt.

Der Wechselkurs von Euro € in Baht ฿ ist komplexer als andere chaotische Zeitreihen. Er ist von mehreren Faktoren abhängig, wie z.B. Wechselkurs anderer Währungen, Marktssituationen oder Währungstrategien der Regierungen in bestimmten Zeiträumen. Trotzdem versuchen wir die Prädiktionsfunktion allein mit den vorherigen Werten des Wechselkurses zu entwickeln.

Änhlich wie beim Experiment mit der Sonnenfleckenrelativzahl wird die Zeitreihe durch einen Tiefpass gefiltert. Der eingesetzte Tiefpass ist ein digitales Butterworth-Filter zweiter Ordnung, dessen Grenzfrequenz $f_{\mathrm{c}} = 0.025$ Tag^{-1} ist. Die gefilterte Zeitreihe ist in Abbildung 9.11 (unten) dargestellt.

[1] http://www.bot.or.th/bothomepage/databank/FinMarkets/ExchangeRate/exchange_t.asp

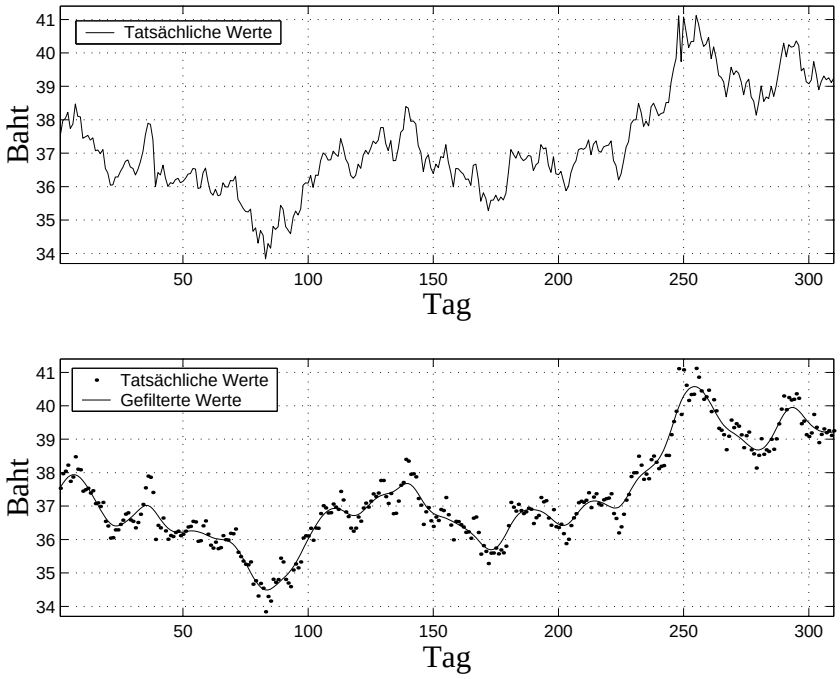


Abbildung 9.11: *Oben:* Täglicher durchschnittlicher Verkaufspreis eines Euro € in Baht ฿ vom 02.01.2001 bis 15.08.2001, *Unten:* Neuer täglicher durchschnittlicher Verkaufspreis von Euro in Baht nach der Störungsreduzierung

Einstellung des Experiments

In diesem Experiment wird das rekursive Prädiktionsverfahren mit fünf Rekursionsstufen verwendet. Zur Konstantenoptimierung wird GA eingesetzt. Dabei wird die erste Möglichkeit zur Einfügung der Koeffizienten verwendet. Die Schätzung des Generalisierungsfehlers erfolgt nach der Teilung der Trainingsdaten. Die Teilung der Daten in Training-, Validierungs- und Testmenge ist in Tabelle 9.15 gezeigt.

Mengentypen	Daten
Trainingsmenge	$x_{22}, x_{23}, x_{25}, x_{26}, x_{28}, x_{29}, ..., x_{299}$
Validierungsmenge	$x_{21}, x_{24}, x_{27}, x_{30}, ..., x_{300}$
Testmenge	$x_{301}, x_{302}, x_{303}, ..., x_{400}$

Tabelle 9.15: Teilung von Daten in Training-, Validierungs- und Testmenge beim Experiment Wechselkurs Euro € und Baht ฿, wobei: x_{22} der durchschnittliche Verkaufspreis am 22. Tag, x_{23} der durchschnittliche Verkaufspreis am 23. Tag, usw. ist.

Wichtige Bedingungen des GP-Laufes sind folgende:

Terminalmenge und Funktionenmenge: Die folgende Funktionenmenge $\mathcal{F}_4$ wird bei den Simulationen verwendet:

$$\mathcal{F}_4 = \{\sin, \cos, \log, \exp, +, -, \times, \div\} \tag{9.16}$$

Die verwendete Terminalmenge $\mathcal{T}_4$ ist:

$$\begin{aligned} \mathcal{T}_4 \;=\; & \{x_t, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}, x_{t-5}, x_{t-6}, x_{t-7}, \\ & x_{t-9}, x_{t-11}, x_{t-13}, x_{t-15}, x_{t-17}, x_{t-19}, \\ & -10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0\} \end{aligned} \tag{9.17}$$

wobei: x_t der Wert zum Zeitpunkt t, x_{t-1} der Wert zum Zeitpunkt $t-1$ usw. ist. $-10.0, -9.5, -9.0, ..., 9.0, 9.5, 10.0$ sind die einfachen konstanten Werte im Bereich von $-10,0$ bis $10,0$.

Fitnessfunktion: Ähnlich wie bei den anderen Experimenten werden der Trainingsfehler E_{Train} (entspricht der Rohfitness f_{r}), der Validierungsfehler E_{Valid} und der Testfehler E_{Test} mit Gleichung (8.6) mit dem Prädiktionsschritt von eins berechnet.

Einstellung der GP-Parameter und Abbruchkriterium: Die Parameter der GP sind in Tabelle 9.11 (mit der Populationsgröße $M = 1000$) dargestellt. In jeder Rekursionsstufe werden 50 GP-Läufe durchgeführt. Nur die beste durch GP entwickelte Prädiktionsfunktion wird ausgewählt und anschließend zur Konstantenoptimierung benutzt. Der GP-Lauf wird nur gestoppt, wenn die Anzahl der maximalen Generation $G_{\max}$ erreicht ist.

Simulationsergebnisse

Ähnlich wie im vorherigen Experiment wurde das Training mit Trainingsdaten nach Störungsreduzierung nach der 4. Rekursionsstufe gestoppt, da keine gute Sub-Prädiktion in der 4. Rekursionsstufe gefunden wird, die den Trainings- und Validierungsfehler kleiner als in der 3. Rekursionsstufe macht. Allerdings sind der Trainings- und Validierungsfehler in der 3. Rekursionsstufe bereits extrem klein. Das Training mit originalen Trainingsdaten wurde dagegen nach der 5. Rekursionsstufe gestoppt.

Die Prädiktionsfehler der beiden entwickelten Prädiktionsfunktionen sind in Tabelle 9.16 gezeigt, wenn sie die originalen Daten prädiktieren. Abbildung 9.12 stellt den Vergleich zwischen den prädiktierten Werten des täglichen Verkaufspreises von Euro € in Baht ฿ durch die Einschrittprädiktion und den tatsächlichen Werten dar.

Situation der Trainingsdaten	Trainingsfehler E_{Train}	Validierungsfehler E_{Valid}	Testfehler E_{Test}
$f_{\text{sum}_{\text{I}}}$	$2,67 \cdot 10^{-2}$	$5,37 \cdot 10^{-2}$	$13,93 \cdot 10^{-2}$
$f_{\text{sum}_{\text{II}}}$	$8,67 \cdot 10^{-2}$	$7,21 \cdot 10^{-2}$	$16,96 \cdot 10^{-2}$
	$(1,84 \cdot 10^{-6})$	$(2,41 \cdot 10^{-6})$	$(21,62 \cdot 10^{-6})$

Tabelle 9.16: Prädiktionsfehler der Prädiktionsfunktion in den beiden Situationen der Trainingsdaten, wobei: $f_{\text{sum}_{\text{I}}}$ die mit den originalen Trainingsdaten entwickelte Prädiktionsfunktion und $f_{\text{sum}_{\text{II}}}$ die mit den Trainingsdaten nach Störungsreduzierung entwickelte Prädiktionsfunktion ist. Die in runde Klammern gesetzten Werte sind die Prädiktionsfehler der Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$ für die Daten nach der Störungsreduzierung.

Bei der Einschrittprädiktion ist die mit Trainingsdaten nach der Störungsreduzierung entwickelte Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$ ungeeignet. Ihre Prädiktionsfehler in Training-, Validierungs- und Testmenge sind größer, als die Prädiktionsfehler der mit den originalen Trainingsdaten entwickelte Prädiktionsfunktion $f_{\text{sum}_{\text{I}}}$. Jedoch ist sie deutlich besser bei der iterativen Einschrittprädiktion.

Abbildung 9.13 zeigt Beispiele der iterativen Einschrittprädiktion von beiden Prädiktionsfunktionen. Die Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktion von beiden Prädiktionsfunktionen sind in Abbildung 9.14 dargestellt.

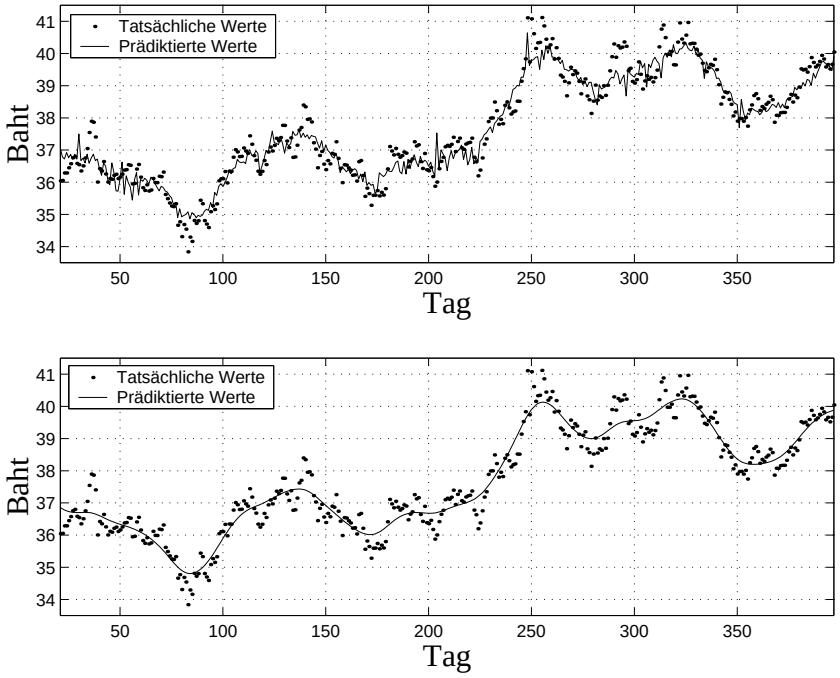


Abbildung 9.12: Vergleich zwischen den prädiktierten Werten des täglichen Verkaufspreises von Euro € in Baht ฿ durch die Einschrittprädiktion und den tatsächlichen Werten, *Oben:* Prädiktion durch die Prädiktionsfunktion $f_{\mathrm{sum_I}}$, *Unten:* Prädiktion durch die Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$.

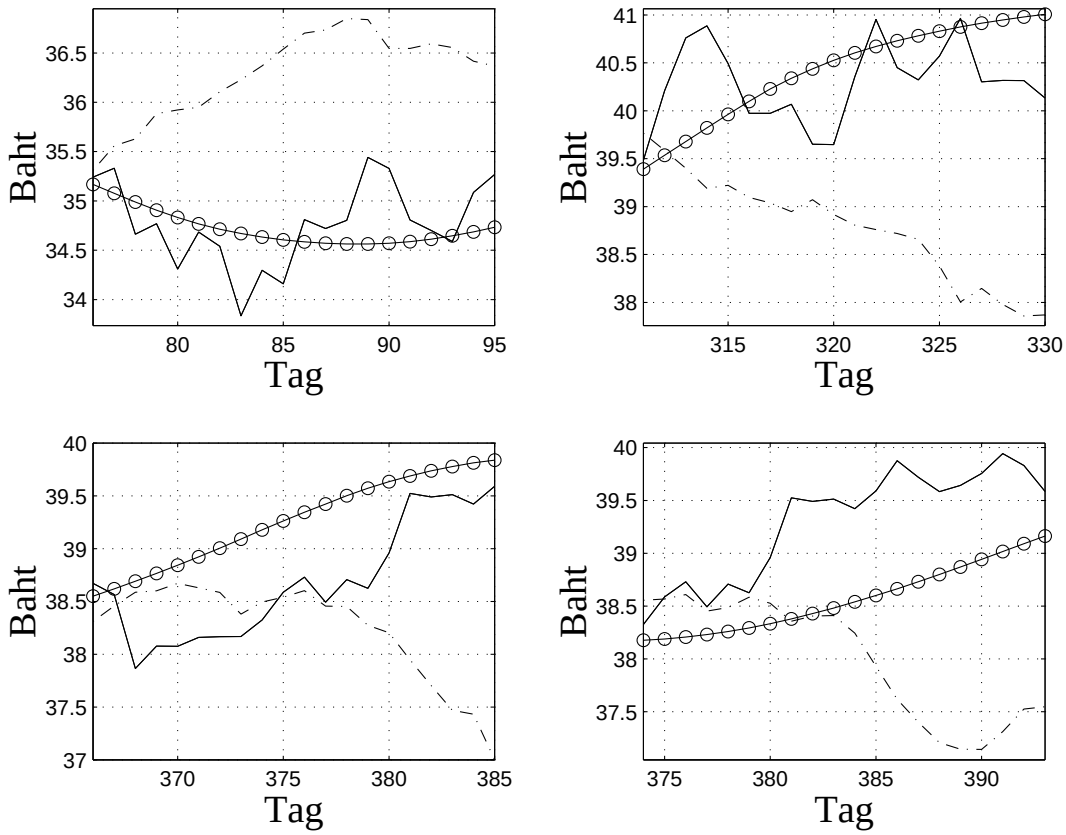


Abbildung 9.13: Beispiele der iterativen Einschrittprädiktion des täglichen Verkaufspreises von Euro € in Baht ฿ außerhalb der Trainingsdaten, wobei: $-$ die tatsächlichen Werte, $-\cdot$ die prädiktierten Werte der mit den originalen Trainingsdaten entwickelten Prädiktionsfunktion $f_{\mathrm{sum_I}}$ und $\ominus$ die prädiktierten Werte der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ sind.

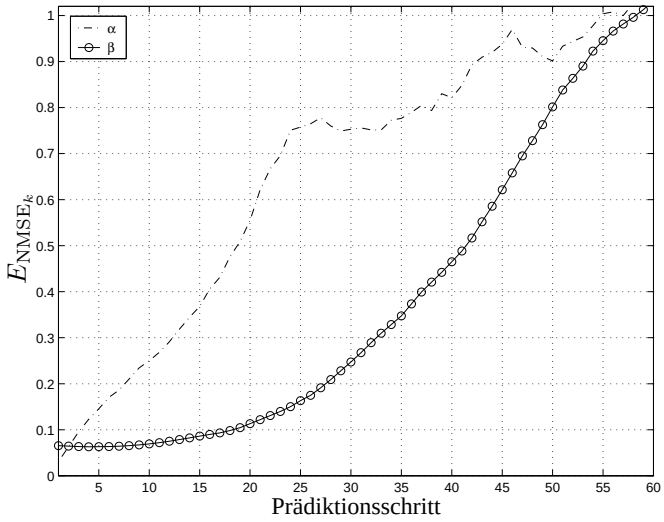


Abbildung 9.14: Prädiktionsfehler E_{NMSE_k} der iterativen Einschrittprädiktionen der täglichen Verkaufspreise von Euro € in Baht ฿ in den gesamten Daten (vom 02.01.2001 bis15.08.2001), wobei: α Prädiktionsfehler der mit den originalen Trainingsdaten entwickelten Prädiktionsfunktion $f_{\mathrm{sum_I}}$ und β Prädiktionsfehler der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ ist.

9.3 Diskussion

Bei Prädiktion einer chaotischen Zeitreihe mit anderen Prädiktionsverfahren muss immer eine geeignete Struktur der Prädiktionsfunktion ausgewählt werden. In unseren Prädiktionsverfahren nimmt GP diese Aufgabe wahr. GP entwickelt die geeignete Struktur der Prädiktionsfunktion aus der Terminalmenge $\mathcal{T}$ und der Funktionenmenge $\mathcal{F}$, die durch die Analyse der chaotischen Zeitreihe oder durch praktische Erfahrung festgelegt werden können. Die Anforderung höherer mathematischer Kenntnisse ist daher unnötig. Übrigens kann der Inhalt der durch GP entwickelten Prädiktionsfunktion die Information über die chaotische Zeitreihe verraten, falls diese Prädiktionsfunktion sehr gut ist.

In unserem Prädiktionsverfahren bestimmt die Konstantenoptimierung die besten numerischen Variablen und Konstanten für die durch GP entwickelte Prädiktionsfunktion, wodurch diese Prädiktionsfunktion effizienter wird. Da die durch GP entwickelte Prädiktionsfunktion nichtlinear ist, benötigt man eine Konstantenoptimierung, die das globale Optimum sucht. Eine solche Konstantenoptimierung ist zum Beispiel GA. Bei einer einfachen Aufgabe kann jedoch das Least-squares- Verfahren verwendet werden, da das Least-squares- Verfahren im Vergleich zu GA nicht zeitaufwendig ist.

Mit dem Simulationsergebnis des Vergleichs von dem einfachen Prädiktionsverfahren und dem rekursiven Prädiktionsverfahren kann man feststellen, dass das re-

kursive Prädiktionsverfahren effizienter als das einfache Prädiktionsverfahren arbeitet. Bei dem rekursiven Prädiktionsverfahren wird in jeder Rekursionsstufe eine Sub-Prädiktionsfunktion, die ein Teil der richtigen Prädiktionsfunktion sein soll, bestimmt. Das rekursive Prädiktionsverfahren hat folglich den Vorteil, dass die duch GP entwickelten guten Prädiktionsfunktionen während des GP-Laufes nicht verloren gehen. Am Ende des Prädiktionsprozesses bilden alle Sub-Prädiktionsfunktionen $\hat{f}_{\text{sub}_1}$, $\hat{f}_{\text{sub}_2}$, $\hat{f}_{\text{sub}_3}$, ..., $\hat{f}_{\text{sub}_n}$ gemeinsam die Prädiktionsfunktion $\hat{f}_{\text{sum}_n}$. Die Anzahl der Stufen des rekursiven Prädiktionsverfahrens kann jedoch nicht beliebig groß gemacht werden. Wenn keine gute Sub-Prädiktionsfunktion gefunden wird, oder Overfittig auftritt, wird dieses Verfahren abgebrochen.

Die weiteren Prädiktionsexperimente mit vier verschiedenen chaotischen Zeitreihen, nämlich die logistische Abblidung, die monatliche durchschnittliche Sonnenfleckenrelativzahl, die jährliche durchschnittliche Sonnenfleckenrelativzahl und der Wechselkurs zwischen Euro € und Baht ฿ zeigen, dass unsere Prädiktionsverfahren bei der Prädiktion aller vier chaotischen Zeitreihen effizient sind. Die Experimente sind zwar unterschiedlich kompliziert. Je komplexer die chaotische Zeitreihe oder je stärker das übergelagerte Rauschen ist, desto schwieriger ist die Entwicklung einer Prädiktionsfunktion.

Die Ergebnisse aus den Experimenten zeigen, dass der Validierungsfehler E_{Valid} meistens größer ist als der Trainingsfehler E_{Train} und der Testfehler E_{Test} meistens größer als der Validierungsfehler E_{Valid} ist. Ohne Vermeidung von Overfitting würden die Prädiktionsfunktionen für die Prädiktion außerhalb der Trainingsdaten ungeeignet sein, insbesondere wenn die chaotische Zeitreihe stark von Störungen überlagert ist.

Unsere Prädiktionsfunktionen sind auf der Basis der Einschrittprädiktion entwickelt, daher gibt es keine Garantie dafür, dass diese Prädiktionsfunktionen zur iterativen Einschrittprädiktion gut eingesetzt werden können. Zur Verbesserung der Mehrschrittprädiktion durch die iterative Einschrittprädiktion sollen Störungen in den chaotischen Zeitreihen reduziert werden. Ein anderer Vorteil der Störungsreduzierung ist, dass die Prädiktionsfunktion einer chaotischen Zeitreihe nach Störungsreduzierung sehr leicht zu entwickeln ist, da die chaotischen Zeitreihen nach Störungsreduzierung nicht mehr komplex sind. Die Störungsreduzierung durch die Filterung mit Tiefpässen ist eine Möglichkeit. Jedoch ist die Filterung mit Tiefpass nicht in jeder Situationen zu verwenden, da die chaotischen Zeitreihen meistens einen hohen Frequenzanteil besitzen, der durch die Filterung verloren geht.

9.4 Vergleich mit anderen Prädiktionsverfahren

Abbildung 9.15 stellt die Prädiktion der monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\text{s}_{\text{month}}}$ durch die mit den Trainingsdaten nach Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\text{sum}_{\text{II}}}$ bei den verschiedenen Startwerten im

Vergleich zur Prädiktion von National Oceanic and Atmospheric Administration (NOAA)[2] dar.

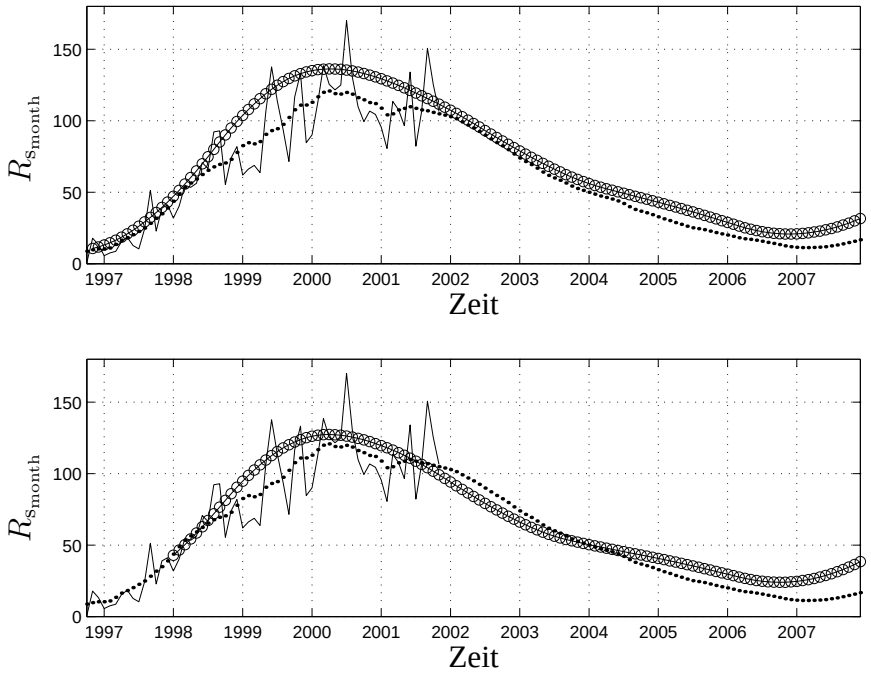


Abbildung 9.15: Prädiktion der monatlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{month}}}$ bis zum 12.08.2008 durch die mit den Trainingsdaten nach Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ bei den verschiedenen Startwerten (10.1996 und 01.1998), wobei $-$ die tatsächlichen Werte, $\ominus$ die prädiktierten Werte der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion $f_{\mathrm{sum_{II}}}$ und $\cdots$ die prädiktierten Werte von NOAA sind.

Die prädiktierten Werte von NOAA sind in Wirklichkeit die Prädiktion des aus 13 monatlichen durchschnittlichen Sonnenfleckenrelativzahlen berechneten Mittelwerts. Die beiden prädiktierten Werte in Abbildung 9.15 verlaufen mit sehr ähnlicher Form, und sind nur wenig unterschiedlich. Die aktuelle maximale monatliche durchschnittliche Sonnenfleckenrelativzahl liegt im Jahr 2000, und die nächste minimale monatliche durchschnittliche Sonnenfleckenrelativzahl soll im Jahr 2007 sein.

Die Prädiktion der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\mathrm{s_{year}}}$ wurde bereits in verschiedenen Arbeiten durchgeführt. Die meisten Arbeiten benutzen die gleiche Funktion (8.6) zum Berechnen des Prädiktionsfehlers, legen jedoch die Varianz $\sigma^2 = 1535$ fest. Zum Vergleich mit anderen Prädiktionsverfahren werden unsere Ergebnisse auch mit der Varianz $\sigma^2 = 1535$ bestimmt. Tabelle 9.17 zeigt unsere neu bestimmten Prädiktionsfehler im Vergleich zu den Prädiktionsfehlern einiger Prädiktionsverfahren, die die Idee von GP oder KNN als Hauptstruktur des

[2]ftp://ftp.ngdc.noaa.gov/STP/SOLAR_DATA/SUNSPOT_NUMBERS/sunspot.predict

Prädiktionsverfahrens verwenden.

Namen der Arbeit	Haupt-struktur	E_{Train} (1700-1920)	E_{Valid} (1921-1955)	E_{Test} (1956-1979)
Angeline [Ang98]	GP	0.103	0.079*	-
Nikolaev [Nik00]	GP	0.113	0.115	0.128
Nikolaev [Nik01]	GP	0.073	0.073	0.114
Weigend [Weig92]	KNN	0.082	0.086	0.350
Uluyol [Ulu98]	KNN	0.127	0.112	0.326
Khalaf [Kha98]	KNN	0.149	0.168	-
Cloarec [Clo98]	KNN	0.090	0.085	0.157
Chan [Cha99]	KNN	0.085	0.092	0.367
$f_{\text{sum}_{\text{I}}}$	GP	0.056	0.042	0.109
$f_{\text{sum}_{\text{II}}}$	GP	0.027	0.039	0.079

Tabelle 9.17: Vergleich des Prädiktionsfehlers E_{NMSE_1} aus Gleichung 8.6 für die Einschrittprädiktionen der jährlichen durchschnittlichen Sonnenfleckenrelativzahl $R_{\text{S}_{\text{year}}}$, die mit $\sigma^2 = 1535$ berechnet wurden. Wobei: GP die genetische Programmierung und KNN das künstliche neuronale Netz ist. (*Der Validierungsfehler von Angeline in [Ang98] wird mit der jährlichen durchschnittlichen Sonnenfleckenrelativzahl von 1920 bis 1950 berechnet.)

Die Fehler der mit den Trainingsdaten nach der Störungsreduzierung entwickelten Prädiktionsfunktion sind kleiner als die Fehler bei den anderen Ergebnissen, insbesondere bei den Testfehlern.

9.5 Zusammenfassung

Im ersten Teil dieses Kapitels wurden Experimente mit dem einfachen und dem rekursiven Prädiktionsverfahren durchgeführt. Das rekursive Prädiktionsverfahren ist besser als das einfache Prädiktionsverfahren, da dieses Verfahren stufenweise die Teile der Originalfunktion (Sub-Prädiktionsfunktion) sucht, und die schon gefundenen Funktionen nicht verlorengehen. Die Anzahl der Stufen des rekursiven Prädiktionsverfahrens kann nicht beliebig groß gemacht werden. Wenn keine gute Sub-Prädiktionsfunktion gefunden wird, oder Overfittig auftritt, wird dieses Verfahren abgebrochen.

Im zweiten Teil wurden Prädiktionsexperimente mit vier verschiedenen chaotischen Zeitreihen gezeigt, nämlich die logistische Abbildung, die monatliche durchschnittliche Sonnenfleckenrelativzahl, die jährliche durchschnittliche Sonnenfleckenrelativzahl und der Wechselkurs zwischen Euro € und Baht ฿. Die Ergebnisse zeigen, dass unsere Prädiktionsverfahren bei der Prädiktion aller vier chaotischen Zeitreihen effizient sind. Die Experimente sind zwar unterschiedlich kompliziert. Je komplexer die chaotische Zeitreihe oder je stärker das übergelagerte Rauschen ist, desto schwieriger ist die Entwicklung einer Prädiktionsfunktion. Dabei wird die Maßnahme gegen

Overfitting immer eingesetzt. Anderfalls ist die entwickelte Prädiktionsfunktion für die Prädiktion von Daten außerhalb der Trainingsdaten nicht verwendbar.

Unsere Prädiktionsfunktionen sind auf der Basis der Einschrittprädiktion entwickelt, es gibt daher keine Garantie dafür, dass diese Prädiktionsfunktionen zur iterativen Einschrittprädiktion gut eingesetzt werden können. Zur Verbesserung der Mehrschrittprädiktion wird das der chaotischen Zeitreihe übergelagerte Rauschen durch Filterung mit einem Tiefpass oder durch Mittelwertberechnung reduziert. Ein anderer Vorteil der Störungsreduzierung ist, dass die Prädiktionsfunktion einer chaotischen Zeitreihe nach Störungsreduzierung sehr leicht zu entwickeln ist.

A Verzeichnis der Abkürzungen und Symbole

A.1 Abkürzungen

ADF	Automatische definierte Funktion	*Automatically defined function*
CTF-GP	Genetische Programmierung mit kontextfreier Grammatik	*Genetic programming using Context-Free Grammar*
EA	Evolutionäre Algorithmen	*Evolutionary computation*
EP	Evolutionäre Programmierung	*Evolutionary programming*
ES	Evolutionsstrategien	*Evolution strategies*
GA	Genetische Algorithmen	*Genetic algorithms*
GP	Genetische Programmierung	*Genetic programming*
KNN	künstliche neuronale Netz	*Artificial neural network*
LFSR	Schieberegister mit linearer Rückkopplung	*Linear feedback shift register*
L-Bit-LFSR	L-Bit- Schieberegister mit linearer Rückkopplung	*L-bits-linear feedback shift register*
SNR	Störabstand	*Signal-to-Noise Ratio*

A.2 Symbole und ihre Bedeutung

$\alpha(G)$	Der Faktor des Vereinfachungszwangs in der Generation G
฿	Baht (die thailändische Währung)
Δ	Der Verschiebungsparameter der Mackey-Glass-Gleichung
€	Euro
λ_i	Der Lyapunov-Exponent der i. Dimension
$\lambda_{\max}$	Der größte Lyaponov-Exponent
σ	Die Standardabweichung
$\phi(r)$	Die radiale Basisfunktion
ξ	Der Wettkampfumfang (bei der Wettkampfselektion)
$\mathcal{A}$	Der Attraktor
$b_{j,G}$	Das Individuum j in der Generation G
$C_{b_{j,G}}$	Die Komplexität des Individuums j in der Generation G
d_{A}	Die Dimension des Attraktors einer Zeitreihe
D	Die Tiefe des Individuums
D_{created}	Die maximale Tiefe des Individuums während des GP-Laufes
D_{div}	Die Ableitungstiefe des Individuums bei CTF-GP
$D_{\mathrm{div}_{\mathrm{created}}}$	Die maximale Ableitungstiefe des Individuums während des GP-Laufes bei CTF-GP
$D_{\mathrm{div}_{\mathrm{initial}}}$	Die maximale Ableitungstiefe des Individuums in Ausgangspopulation bei CTF-GP
D_{initial}	Die maximale Tiefe des Individuums in Ausgangspopulation
D_F	Die fraktale Dimension

D_{E}	Die Einbettungsdimension
E_k	Die Abweichung der Prädiktion beim k. Prädiktionsschritt
E_{NMSE_k}	Das *Normalized Mean Squared Error* beim k. Prädiktionsschritt
E_{Test}	Der Testfehler
E_{Train}	Der Trainingsfehler
E_{Valid}	Der Validierungsfehler
$\hat{f}$	Die Prädiktionsfunktion zur Prädiktion des nächsten zukünftigen Wertes
f_{a}	Die angepasste Fitness
$f_{\mathrm{a}_{b_{j,G}}}$	Die angepasste Fitness des Individuums j in der Generation G
$\hat{f}_k$	Die Prädiktionsfunktion zur Prädiktion des k. zukünftigen Wertes
f_{n}	Die normalisierte Fitness
$f_{\mathrm{n}_{b_{j,G}}}$	Die normalisierte Fitness des Individuums j in der Generation G
$\bar{f}_{\mathrm{n}_{b_G}}$	Die durchschnittliche normalisierte Fitness in der Generation G
f_{r}	Die Rohfitness
$f_{\mathrm{r}_{b_{j,G}}}$	Die Rohfitness des Individuums j in der Generation G
$f_{\mathrm{r,p}_{b_{j,G}}}$	Die Rohfitness des Individuums j in Generation G unter Berücksichtigung des Vereinfachungszwangs
f_{s}	Die Einzelfleckenzahl der Sonnenfleckenrelativzahl
f_{st}	Die standardisierte Fitness
$f_{\mathrm{st}_{b_{j,G}}}$	Die standardisierte Fitness des Individuums j in der Generation G
$\hat{f}_{\mathrm{sub}_n}$	Die Sub-Prädiktionsfunktion aus der n. Rekursionsstufe

$\hat{f}_{\mathrm{sum}_n}$	Die Prädiktionsfunktion aus dem Rekursion-Prädiktionsverfahren mit n Rekursionsstufen
$\boldsymbol{f}$	Der reellwertige Funktionsvektor der gewöhnlichen Differenzengleichung
$\boldsymbol{f}^t$	Der Fluss eines dynamischen Systems
$\boldsymbol{F}$	Der reellwertige Funktionsvektor der gewöhnlichen Differentialgleichung
$\mathcal{F}$	Die Funktionenmenge
g_{s}	Die Gruppenzahl der Sonnenfleckenrelativzahl
G	Die Generation
$G_{1:n_{\mathrm{c}}}$	Der relative Geschwindigkeitsfaktor bei der Parallelisierung
$G_{\max}$	Die maximale Generation
GL	Der Generalisierungsverlust
$\mathcal{G}$	Die kontextfreie Grammatik
$I(M, G, z)$	Die minimale Anzahl benötigter Individuen zum Erreichen einer Erfolgswahrscheinlichkeit von z beim GP-Lauf mit der Populationsgröße M und der maximalen Generation G
k	Der Prädiktionsschritt
k_{s}	Der Korrekturfaktor der Sonnenfleckenrelativzahl
M	Die Populationsgröße
n_{c}	Die Anzahl der Client-Rechner
n_{p}	Die Anzahl der Pakete
$N(\varepsilon)$	Die Anzahl des Kubens der Kantenlänge ε
N_{r}	Die Anzahl der Rekursionsstufen
$\mathcal{N}$	Das Nichtterminalsymbol
p_{c}	Die Kreuzungswahrscheinlichkeit

p_{creat}	Die Wahrscheinlichkeit der stochastischen Erzeugung eines neues Individuums
p_{ed}	Die Editierenswahrscheinlichkeit
p_{en}	Die Einkapselungswahrscheinlichkeit
p_{ep}	Die Kreuzungswahrscheinlichkeit im inneren Knoten
p_{ip}	Die Kreuzungswahrscheinlichkeit im Endknoten
p_{m}	Die Mutationswahrscheinlichkeit
p_{p}	Die Permutationswahrscheinlichkeit
P_l	Der Trainingsfortschritt
$\mathcal{P}$	Die Relation der kontextfreien Grammatik
$P(M, G)$	Die kumulative Wahrscheinlichkeit eines erfolgreichen GP-Laufes mit der Populationsgröße M und der maximalen Generation G
$\mathbf{rank}(b_{j,G})$	Der Rang des Individuums j in der Generation G
r	Der Kontrollparameter der logistischen Abbildung
R_{s}	Die Sonnenfleckenrelativzahl
$R_{\text{s}_{\text{month}}}$	Die monatliche durchschnittliche Sonnenfleckenrelativzahl
$R_{\text{s}_{\text{year}}}$	Die jährlichen durchschnittliche Sonnenfleckenrelativzahl
R_{E}	Der akkumulierte Rechenaufwand
$R(M, G, z)$	Die Anzahl der benötigten unabhängigen GP-Läufe mit der Populationsgröße M und der maximalen Generation G, um mit der Wahrscheinlichkeit z mindestens einen erfolgreichen Versuch zu bekommen
Run	Die Anzahl des GP-Laufes
Run_{max}	Die maximale Anzahl des GP-Laufes
S	Das Startsymbol der kontextfreien Grammatik

$\mathbf{S}_{\text{Test}}$	Die Testmenge
$\mathbf{S}_{\text{Train}}$	Die Trainingsmenge
$\mathbf{S}_{\text{Valid}}$	Die Validierungsmenge
$t_{\text{sr}_{b_{j,G}}}$	Die Auswahlwahrscheinlichkeit des Individuums j in der Generation G
t_{T}	Der Trainingsschritt
$T_{n_{\text{c}}-\text{Clients}}$	Der Zeitverbrauch zur Durchführung einer gegebenen Aufgabe bei der Parallelisierung mit n Client-Rechnern
T_{G}	Der Zeitaufwand zur Durchführung aller Vorgänge innerhalb einer Generation
T_{H}	Die Vorlaufzeit bei der Parallelisierung
T_{M}	Der Gesamtzeitaufwand der Fitnessberechnung im Multi-Verfahren
T_{P}	Der Gesamtzeitaufwand der Fitnessberechnung eines Paketes
T_{S}	Der Gesamtzeitaufwand der Fitnessberechnung im Single-Verfahren
T_{Single}	Der Zeitverbauch zur Durchführung einer gegebenen Aufgabe von einem Rechner
$\mathcal{T}$	Die Terminalmenge
x_i	Die Zustandsvariable i
x_t	Der tatsächliche Wert zur Zeit t
$\hat{x}_t$	Der prädiktierte Wert zur Zeit t
$\{x_t\}$	Die Zeitreihe
$\boldsymbol{x}_t$	Der Delay-Vektor
$\boldsymbol{x}(t)$	Der Zustand des Systems zur Zeit t
z	Die Erfolgswahrscheinlichkeit

B Prädiktionsfunktionen aus den Experimenten

In diesem Anhang werden die Prädiktionsfunktionen und die dazugehörigen Koeffizienten aus den Experimenten im Abschnitt 9.2 dargestellt. (Die Prädiktionsfunktionen aus den Experimenten im Abschnitt 9.2 sind bereits in [Krai01] gegeben.)

B.1 Definierte Mathematische Funktionen

Die definierten Funktionen, z.B. Add, Sub, usw. sind als mathematische Funktionen zu interpretieren:

$$\mathbf{Add}(a,b) \quad \Longrightarrow \quad a+b$$

$$\mathbf{Sub}(a,b) \quad \Longrightarrow \quad a-b$$

$$\mathbf{Mul}(a,b) \quad \Longrightarrow \quad a \times b$$

$$\mathbf{Div}(a,b) \quad \Longrightarrow \quad \begin{cases} a/10^{-10}, & \text{wenn: } 0.0 \leq b < 10^{-10} \\ a/-10^{-10}, & \text{wenn: } -10^{-10} < b < 0.0 \\ a/b, & \text{wenn: } |b| \geq 10^{-10} \end{cases}$$

$$\mathbf{Sin}(a) \quad \Longrightarrow \quad \sin(a)$$

$$\mathbf{Cos}(a) \quad \Longrightarrow \quad \cos(a)$$

$$\mathbf{Log}(a) \quad \Longrightarrow \quad \begin{cases} \log(10^{-10}), & \text{wenn: } a < 10^{-10} \\ \log(a), & \text{wenn: } a \geq 10^{-10} \end{cases}$$

$$\mathbf{Exp}(a) \quad \Longrightarrow \quad \begin{cases} \exp(100), & \text{wenn: } a > 100 \\ \exp(a), & \text{wenn: } a \leq 100 \end{cases}$$

$$\mathbf{Xsin}(a,b) \quad \Longrightarrow \quad b \cdot \sin(a)$$

$$\mathbf{Xcos}(a,b) \quad \Longrightarrow \quad b \cdot \cos(a)$$

$$\mathbf{Xlog}(a,b) \quad \Longrightarrow \quad \begin{cases} b \cdot \log(10^{-10}), & \text{wenn: } a < 10^{-10} \\ b \cdot \log(a), & \text{wenn: } a \geq 10^{-10} \end{cases}$$

$$\mathbf{Xexp}(a,b) \quad \Longrightarrow \quad \begin{cases} b \cdot \exp(10), & \text{wenn: } a > 10 \\ b \cdot \exp(a), & \text{wenn: } a \leq 10 \end{cases}$$

Außerdem bedeutet x_{t-1} den Wert zum Zeitpunkt $t-1$, x_{t-2} den Wert zum Zeitpunkt $t-2$ und usw.

B.2 Logistische Abbildung

B.2.1 Keine Störung

u_1 = {1.13748146811767, 0.32355473773869, -1.1978392865771, 1.46778248930964, 0.85336710803223, -1.0995860025185, 1.16549806692872, 1.56165017553010, 0.58119638311112, 1.03344407470228, 0.81504259043283, -1.1218499007713, 1.16071378762191, 1.01081396908413, 0.73913543357378, 0.72477061632655, -0.9859129698924, 1.19178603415206, 1.43779853308421}

$\hat{f}$ = (u_1(1)·Sin ((Mul ((Sub ((u_1(2)·Sin ((Mul ((Sub ((u_1(3)·Exp ((u_1(4)· x_{t-1}))) , (u_1(5)·Sin (((u_1(6)·4.0)))))) , (u_1(7)·Sin ((Mul ((Sub ((u_1(8)· x_{t-1}) , (u_1(9)·Sin (((u_1(10)·4.0)))))) , (u_1(11)·Sin ((u_1(12)·Exp ((u_1(13)·Sin ((u_1(14)· x_{t-1}))))))))))))) , (u_1(15)·Sin (((u_1(16)·4.0)))))) , (u_1(17)·Sin ((Mul ((Sub ((u_1(18)·Exp ((u_1(19)·Sin ((u_1(20)· x_{t-1}))))) , (u_1(21)·Sin (((u_1(22)·4.0)))))) , (u_1(23)· x_{t-1})))))))))

B.2.2 Störung mit SNR = 30 dB

u_1 = {1.20808889935060, 1.52595025466711, 0.93626862836253, 1.38763280819204, 1.26637820653230, 1.49487626920418, 1.56548810240862, 1.09764658379341, 1.04561720299847, 1.39279374466989, 0.90684574810825, 1.38342825287460, 0.43100052156626, 0.84773383797233, 1.13936710926682, 1.46035784533754, 1.11491049205244, 1.23331734474781, 1.13132727833189, 0.99959468351353, 1.17680657133697}

$\hat{f}$ = (u_1(1)·Sin ((Mul ((Add ((u_1(2)·Sin ((Add ((u_1(3)·Sin ((Add ((u_1(4)· x_{t-1}) , (Add ((u_1(5)·Cos ((u_1(6)· x_{t-1}))) , (u_1(7)·Cos ((u_1(8)·Cos ((Add ((u_1(9)· x_{t-1}) , (Add (((0.0)) , (u_1(10)·Cos ((u_1(11)·Cos ((u_1(12)· x_{t-1}) , (u_1(13)· x_{t-1}))))) , (Add ((u_1(14)·Cos ((u_1(15)·Cos (((u_1(16)·-6.5)))))) , (Add ((u_1(17)·Cos ((

(u_1(18)·-6.5)))) , (u_1(19)·Cos ((u_1(20)·Cos ((Add ((u_1(21)·Cos (((u_1(22)·-6.0)))) , (Add ((u_1(23)· x_{t-1}) , (u_1(24)·Cos (((u_1(25)·-6.5)))))))))))))))))) , (u_1(26)· x_{t-1})))))

B.2.3 Störung mit SNR = 20 dB

u_1 = {0.31644633222825, 0.83809921607329, 2.87351619586779, 2.91398078837725, 0.90120312115978, -0.3676571978572, 0.80234362997407, 3.27108116666533, -0.2884265400451, 0.61173577875572, 2.23297455078256, 0.94507828747884, -0.3078330389815, 2.06913845944007}

$\hat{f}$ = (u_1(1)·Exp ((Sub ((u_1(2)·Sin ((Sub ((u_1(3)·Sin ((u_1(4)·Exp ((Sub ((u_1(5)· x_{t-1}) , (u_1(6)·Exp ((u_1(7)·Sin ((Div ((u_1(8)· x_{t-1}) , ((-1.0·u_1(9)))))))))))))))) , (u_1(10)·Exp ((u_1(11)·Sin ((Div ((u_1(12)· x_{t-1}) , (u_1(13)·Sin ((u_1(14)·Log ((u_1(15)·Cos (((7.0·u_1(16) , (u_1(17)·Exp ((u_1(18)·Sin ((Div ((u_1(19)· x_{t-1}) , (u_1(20)·Sin ((u_1(21)·Log ((u_1(22)·Cos (((7.0·u_1(23))))))))))))))))))))

B.2.4 Störung mit SNR = 10 dB

u_1 = {1.36969017176464, 1.16157002679067, 0.94880691978672, 0.68043530351333, 0.22786225147926, 2.51961668808961, 1.41474331449367, 0.69465501371114, 1.46281326334860, 1.43340919039010, 2.34038256766255, 1.73385912719518, 0.57746418149818, 1.76534401228997, 1.73000428219608}

$\hat{f}$ = u_1(1)·Sin (u_1(2)·Sin (u_1(3)·Sin (u_1(4)·Sin (u_1(5)·Exp (u_1(6)·Cos ((u_1(7)·Exp (u_1(8)·i)) - (u_1(9)·Sin (u_1(10)·Sin (u_1(11)·Sin (u_1(12)·Cos ((u_1(13)·Exp (u_1(14)·i)) - (u_1(15)·1.0))))))))))))

B.3 Monatliche durchschnittliche Sonnenfleckenrelativzahl

B.3.1 Lernen mit den originalen Trainingsdaten

a_0 = {0.83943434965220}

a_1 = {0.97532746805252}

a_2 = {1.09397981999240}

a_3 = {1.01955069108767}

$a_4 = \{1.0\}$

$a_5 = \{1.0\}$

$$\hat{f}_{\text{sum}_n} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) + ... + (a_5 \cdot \hat{f}_{sub_5})$$

1. Rekursionsstufe

$v_1 = \{5.11100330950262, -0.5156526097115, 9.98266372452124, -1.4876744113247, 3.68483356350453, -3.1397091084050, -3.7263237960965, 10.0000000000000, 1.84533752111615, -8.2927193454107, 6.30573370055677, 7.96592910654913, 9.99999999999999, -2.9215982936647, 4.21285425932369, 10.0000000000000, -5.9118129399262, 3.53720063161797, 0.31206382156996, -10.000000000000, 2.04165177953696, -6.8045237521821, 3.16459793667763, 9.99999999999999, -1.0283872669857, -9.7631688443220, -2.3940059438521, -5.1214886871739, -10.000000000000, -4.0827034948961, -4.2921152630273, -6.2354519096909, 1.32166621141266, 3.91191709729366, 2.46676683846216, -0.4293276427383, -9.4222102845343\}$

$u_1 = \{0.70606037794386, 1.00000000000000, 1.25909550485705, 0.68729169870901, 1.27766131952943, 0.97684384717247, 1.00046056321056, 1.00037442489043, 1.00872826690057, 0.88215176964389, 0.97500917882854, 0.96716966570497, 1.02207982670294, 1.02207713912470, 0.99867854816707, 0.76575776745984, 1.14414939649124, 1.12318256138907, 0.88396608876737, 1.12346014386474, 1.12345869356761, 0.88342047290600, 1.46990222349624, 1.55395128125697, 1.02912830405968, 1.02912656807434, 0.97421000038775, 0.31015200452022, 1.08234293651289, 1.00092332800053, 1.00024739868037, 0.99939982296540, 0.98816048673819, 0.99888268709942, 2.40628388529627, 1.22208502970757, 1.49986652751895, 2.38972154767160, 0.86307248692182, 0.99633016481807, 0.95330244807287, 1.07265067772877, 0.97020788396584, 0.76850580145104, 1.03488842101287, 1.65647138948288, 1.00752059479175, 0.93281380609100, 0.99975305443467, 0.80778224758902, 1.04282950845269, 0.99681770540156, 1.85643359933125, 1.36769363486361, 0.62462294824656, 0.55309345172824, 3.90068560613276, 1.00758414495922, 0.94955790555381, 1.04345008588475, 0.84476864281214, 1.03243481350725, 1.07429592215883, 0.92325150483960, 0.98647214532040, 0.99538465058497, 1.00510079343719, 0.99538644156973, 1.00510621826447, 0.99537970099314, 0.99539146833958, 1.02655409382517\}$

$\hat{f}_{\text{sub}_1}$ = Sub ((Xcos ((Sub ((Xcos ((Xsin ((Add ((Xlog (((0.0)) , (u_1(1)))) , (Xcos ((Xexp (((v_1(1)·-4.5)) , (u_1(2)))) , (u_1(3)))))) , (u_1(4)))) , (u_1(5)))) , (Xcos ((Sub ((Div ((Add (((u_1(6)·x_{t-1})) , ((v_1(2)·0.5)))) , (Div ((Sub (((u_1(7)·x_{t-6})) , ((v_1(3)·6.5)))) , (Mul ((Xsin ((Xexp ((Xcos (((u_1(8)·x_{t-150})) , (u_1(9)))) , (u_1(10)))) , (u_1(11)))) , (Add ((Xcos ((Div ((Xsin (((v_1(4)·-9.5)) , (u_1(12)))) , (Xlog (((v_1(5)·10.0)) , (u_1(13)))))) , (u_1(14)))) , (Xcos (((u_1(15)·x_{t-150})) , (u_1(16))))))))))) , (Xcos ((Sub ((Div ((Sub ((Xcos (((v_1(6)·8.5)) , (u_1(17)))) , (Xcos ((Xcos (((v_1(7)·-1.5)) , (u_1(18)))) , (u_1(19)))))) , ((v_1(8)·-7.5)))) , (Xlog ((Xlog (((v_1(9)·4.0)) , (u_1(20)))) , (u_1(21)))))) , (u_1(22))))) , (u_1(23)))))) , (u_1(24)))) , (Sub ((Sub ((Xcos (((v_1(10)·8.5)) , (u_1(25)))) , (Xcos ((Xcos (((v_1(11)·-1.5)) , (u_1(26)))) , (u_1(27)))))) , (Add ((Div ((Sub (((u_1(28)·x_{t-6})) , ((v_1(12)·6.5)))) , (Add (((u_1(29)·x_{t-1})) , ((v_1(13)·0.5)))))) , (Mul ((Sub ((Sub ((Xcos ((Sub ((Xlog ((Add ((Xsin (((v_1(14)·-0.5)) , (u_1(30)))) , (Xcos (((u_1(31)·x_{t-150})) , (u_1(32)))))) , (u_1(33)))) , (Add (((v_1(15)·-1.0)) , ((u_1(34)·x_{t-1})))))) , (u_1(35)))) , (Sub ((Xsin ((Div ((Sub ((Add (((v_1(16)·-1.0)) , ((u_1(36)·x_{t-1})))) , (Xcos (((v_1(17)·8.5)) , (u_1(37))))))) , (Mul (((v_1(18)·-9.5)) , ((v_1(19)·5.0)))))) , (u_1(38)))) , (Add ((Div ((Sub (((u_1(39)·x_{t-6})) , ((v_1(20)·6.5)))) , (Mul ((Xsin ((Xexp ((Xcos (((u_1(40)·x_{t-150})) , (u_1(41)))) , (u_1(42)))) , (u_1(43)))) , (Add ((Sub (((u_1(44)·x_{t-30})) , ((v_1(21)·-3.0)))) , (Xcos (((u_1(45)·x_{t-150})) , (u_1(46))))))))) , (Mul ((Sub ((Sub ((Add (((v_1(22)·-1.0)) , ((u_1(47)·x_{t-1})))) , (Xcos (((v_1(23)·8.5)) , (u_1(48)))))) , (Xcos ((Sub ((Xlog ((Xsin ((Add (((u_1(49)·x_{t-1})) , ((v_1(24)·0.5)))) , (u_1(50)))) , (u_1(51)))) , (Add (((v_1(25)·-1.0)) , ((u_1(52)·x_{t-1})))))) , (u_1(53)))))) , (Xexp ((Div ((Xsin (((v_1(26)·-9.5)) , (u_1(54)))) , (Sub (((u_1(55)·x_{t-2})) , (Div (((v_1(27)·5.0)) , (Xlog ((Xexp (((v_1(28)·4.0)) , (u_1(56)))) , (u_1(57))))))))))) , (u_1(58)))))))))))) , (Xcos ((Mul (((v_1(29)·-9.5)) , ((v_1(30)·5.0)))) , (u_1(59)))))) , (Xexp ((Div ((Xsin (((v_1(31)·-9.5)) , (u_1(60)))) , (Add ((Xcos ((Xsin ((Add ((Xsin (((v_1(32)·-0.5)) , (u_1(61)))) , (Div ((Sub (((u_1(62)·x_{t-6})) , ((v_1(33)·6.5)))) , (Add (((u_1(63)·x_{t-1})) , ((v_1(34)·0.5)))))))) , (u_1(64)))) , (u_1(65)))) , (Mul ((Sub ((Xcos (((v_1(35)·8.5)) , (u_1(66)))) , (Xcos ((Xcos (((v_1(36)·-1.5)) , (u_1(67)))) , (u_1(68)))))) , (Xexp ((Div ((Xsin (((v_1(37)·-9.5)) , (u_1(69)))) , (Xlog (((0.0)) , (u_1(70)))))) , (u_1(71)))))))))) , (u_1(72)))))))))))

2. Rekursionsstufe

v_2 = {0.01659989618739, -3.5454681522141, 5.50090850253201, -4.8328595848499, 1.52599700660086, 9.54356474271123, -9.9995639519189, -0.9095270677725}

u_2 = {0.53374287271753, 1.01107156068950, 1.16234685522768, 0.97612012231435, 1.02162674446185, 0.99904806637339, 1.00680968714067, 1.07923112939983, 1.00027168506659, 0.99963531996717, 1.00027164121921, 1.26072617299498, 1.00171199177472, 0.99853951917343, 1.00001154760169, 1.00496344137162, 1.09381721568869, 1.04921981771392, 0.91531124806709, 0.99996100220348, 0.99999986305938, 0.99996201997942, 1.09203799247281, 1.26976981398473, 0.91405889245112, 0.96418406377091, 0.98784362356428, 0.99376105100355,

0.99224509203438, 1.00731126439934, 1.00001632063173, 1.00820949319977, 0.96562137225020, 1.00014972613685, 0.99990702979829, 1.00014972546518, 0.97820189523483, 1.01692656919771, 0.98560676899548, 1.05610348900235, 1.08095527805714, 2.19679554571155, 1.00424029399239, 2.88449486836718, 1.06769660790128, 1.18706708444846, 1.13201557927847, 1.12475661892383, 1.00649308079182, 1.14792417266744, 1.01397843432359}

$\hat{f}_{\text{sub}_2}$ = Mul ((Sub ((Mul ((Sub ((Xlog ((Xlog ((Div ((($u_2(1)\cdot x_{t-110}$)) , (($v_2(1)$·-2.5)))) , ($u_2(2)$))) , ($u_2(3)$))) , (Xlog ((Add ((Div ((($u_2(4)\cdot x_{t-150}$)) , (($u_2(5)\cdot x_{t-7}$)))) , (Xcos ((($u_2(6)\cdot x_{t-130}$)) , ($u_2(7)$))))) , ($u_2(8)$))))) , (Xcos ((Mul ((($u_2(9)\cdot x_{t-110}$)) , (Xexp ((Xsin ((Add ((($v_2(2)$·-6.5)) , (($v_2(3)$·6.0)))) , ($u_2(10)$))) , ($u_2(11)$))))) , ($u_2(12)$)))) , (Xlog ((Add ((Div ((($u_2(13)\cdot x_{t-150}$)) , (($u_2(14)\cdot x_{t-7}$)))) , (Xcos ((($u_2(15)\cdot x_{t-30}$)) , ($u_2(16)$))))) , ($u_2(17)$))))) , (Add ((Add ((Xsin ((Xexp ((Xcos ((Div ((Xexp ((Xexp ((($u_2(18)\cdot x_{t-70}$)) , ($u_2(19)$))) , ($u_2(20)$))) , (Xsin ((Sub ((($v_2(4)$·-6.0)) , (($u_2(21)\cdot x_{t-10}$)))) , ($u_2(22)$))))) , ($u_2(23)$))) , ($u_2(24)$))) , ($u_2(25)$))) , (Xsin ((Xlog ((Sub ((Mul ((Sub ((Xlog ((Xlog ((Div ((($u_2(26)\cdot x_{t-110}$)) , (($v_2(5)$·-2.5)))) , ($u_2(27)$))) , ($u_2(28)$))) , (Xlog ((Add ((Div ((($u_2(29)\cdot x_{t-150}$)) , (($u_2(30)\cdot x_{t-7}$)))) , (Xcos ((($u_2(31)\cdot x_{t-130}$)) , ($u_2(32)$))))) , ($u_2(33)$))))) , (Xcos ((Mul ((($u_2(34)\cdot x_{t-110}$)) , (Xexp ((Xsin ((Add ((($v_2(6)$·-6.5)) , (($v_2(7)$·6.0)))) , ($u_2(35)$))) , ($u_2(36)$))))) , ($u_2(37)$))))) , (Xlog ((Div ((($u_2(38)\cdot x_{t-150}$)) , (($u_2(39)\cdot x_{t-7}$)))) , ($u_2(40)$))))) , ($u_2(41)$))) , ($u_2(42)$))))) , (Xsin ((Sub ((Xsin ((Xexp ((Xcos ((Sub ((Xcos ((($u_2(43)\cdot x_{t-18}$)) , ($u_2(44)$))) , (Xexp ((($v_2(8)$·-3.5)) , ($u_2(45)$))))) , ($u_2(46)$))) , ($u_2(47)$))) , ($u_2(48)$))) , (Xcos ((($u_2(49)\cdot x_{t-30}$)) , ($u_2(50)$))))) , ($u_2(51)$))))))

3. Rekursionsstufe

$v_3 = \{9.55944328379015, 1.11753992782573\}$

$u_3 = \{0.99999508681467, 2.56882942884643, 0.94862398107172, 0.93619577203843, 2.06337120896024, 1.01667595448688, 0.96640220815920\}$

$\hat{f}_{\text{sub}_3}$ = Add ((Add ((Xcos ((Mul ((($u_3(1)\cdot x_{t-8}$)) , (($v_3(1)$·9.0)))) , ($u_3(2)$))) , (Xsin ((Sub ((($u_3(3)\cdot x_{t-3}$)) , (($u_3(4)\cdot x_{t-1}$)))) , ($u_3(5)$))))) , (Div ((Sub ((($u_3(6)\cdot x_{t-3}$)) , (($u_3(7)\cdot x_{t-1}$)))) , (($v_3(2)$·7.0)))))

4. Rekursionsstufe

$v_4 = \{0.98793124787260, 1.01124049825059, 0.96930709934307, 1.06229229761619, 0.94242568156397, 1.07263571709767, 0.91484836464808, 1.07165053914083,$

0.93863502750847, 1.05634402743376, 0.89359603881371, 0.71880200508419, 0.89355969468604, 0.90742717262003, 1.22322336162380, 0.89373238211374, 1.09542865742669, 0.80813367564621, 1.05300465030303, 1.05309033815657, 1.17468100724971, 1.06491920005635, 1.06486632615505, 1.24717804363638, 0.93438448018047, 0.93430571087724, 1.07268136230398, 1.07254531447442, 1.07252759837191, 1.07270520199335, 1.07266355412027, 1.21101263884439, 0.86609668255976, 1.15476659795622, 7.66356288531058, 4.14144831716446, 1.54952452232184, 1.22989557308667, 0.73844290165728, 0.73919011801945, 1.05550503677576, 0.91414162261339}

u_4 = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0}

$\hat{f}_{\mathrm{sub}_4}$ = Add ((Mul ((Mul ((Xlog ((Xcos ((Div ((Xlog ((($u_4(1)\cdot x_{t-9}$)) , ($u_4(2)$)))) , (Sub ((
($v_4(1)\cdot$-5.5)) , (($v_4(2)\cdot$-10.0)))))) , ($u_4(3)$))) , ($u_4(4)$))) , (Xsin ((Sub ((Xcos ((($v_4(3)\cdot$-5.5)
) , ($u_4(5)$))) , (Mul ((($u_4(6)\cdot x_{t-50}$)) , (($u_4(7)\cdot x_{t-50}$)))))) , ($u_4(8)$))))) , (Sub ((Sub (
(Div ((Xcos ((($v_4(4)\cdot$-5.5)) , ($u_4(9)$))) , (Div ((($v_4(5)\cdot$-10.0)) , (($u_4(10)\cdot x_{t-7}$)))))) , (Add
((Xexp ((($v_4(6)\cdot$-5.5)) , ($u_4(11)$))) , (Mul ((Add ((Mul ((Mul ((Xlog ((Xsin ((Add ((Xexp
((($v_4(7)\cdot$-2.0)) , ($u_4(12)$))) , (Mul ((($u_4(13)\cdot x_{t-50}$)) , (($u_4(14)\cdot x_{t-50}$)))))) , ($u_4(15)$))
) , ($u_4(16)$))) , (Xsin ((Sub ((Xcos ((($v_4(8)\cdot$-5.5)) , ($u_4(17)$))) , (Mul ((($u_4(18)\cdot x_{t-50}$)) ,
(($u_4(19)\cdot x_{t-50}$)))))) , ($u_4(20)$))))) , (Sub ((Sub ((Div ((Xcos ((($v_4(9)\cdot$-5.5)) , ($u_4(21)$)
)) , (Div ((($v_4(10)\cdot$-10.0)) , (($u_4(22)\cdot x_{t-7}$)))))) , (Add ((Xexp ((($v_4(11)\cdot$-5.5)) , ($u_4(23)$
))) , (Mul ((Xlog ((($u_4(24)\cdot x_{t-3}$)) , ($u_4(25)$))) , (Xcos ((($u_4(26)\cdot x_{t-4}$)) , ($u_4(27)$))))
))))) , (Add ((Add ((Mul ((Xlog ((Xcos ((Xlog ((($v_4(12)\cdot$-9.5)) , ($u_4(28)$))) , ($u_4(29)$)
)) , ($u_4(30)$))) , (Sub ((Sub ((Xcos ((($v_4(13)\cdot$-5.5)) , ($u_4(31)$))) , (Xexp ((($v_4(14)\cdot$-10.0)
) , ($u_4(32)$))))) , (Add ((Xcos ((Xlog ((($u_4(33)\cdot x_{t-9}$)) , ($u_4(34)$))) , ($u_4(35)$))) , (Div
((Xlog ((($v_4(15)\cdot$-9.0)) , ($u_4(36)$))) , (Xlog ((($u_4(37)\cdot x_{t-9}$)) , ($u_4(38)$)))))))))))
, (Xcos ((($u_4(39)\cdot x_{t-4}$)) , ($u_4(40)$))))) , (Sub ((($v_4(16)\cdot$-5.5)) , (($v_4(17)\cdot$-10.0))))))
)))) , (Xlog ((($u_4(41)\cdot x_{t-4}$)) , ($u_4(42)$))))) , (Xcos ((($u_4(43)\cdot x_{t-4}$)) , ($u_4(44)$))))
))))) , (Add ((Add ((Mul ((Mul ((Xlog ((Xcos ((Xlog ((($v_4(18)\cdot$-9.5)) , ($u_4(45)$))) ,
($u_4(46)$))) , ($u_4(47)$))) , (Xsin ((Add ((Sub ((($v_4(19)\cdot$4.5)) , (($v_4(20)\cdot$-5.0)))) , (Mul (
(($u_4(48)\cdot x_{t-50}$)) , (($u_4(49)\cdot x_{t-50}$)))))) , ($u_4(50)$))))) , (Sub ((Sub ((Div ((Xcos ((
($v_4(21)\cdot$-5.5)) , ($u_4(51)$))) , (Xcos ((($u_4(52)\cdot x_{t-4}$)) , ($u_4(53)$))))) , (Xsin ((Xcos ((Sub
((($v_4(22)\cdot$4.5)) , (($v_4(23)\cdot$-5.0)))) , ($u_4(54)$))) , ($u_4(55)$))))) , (Xcos ((($v_4(24)\cdot$-5.5))
, ($u_4(56)$))))))) , (Xcos ((($u_4(57)\cdot x_{t-4}$)) , ($u_4(58)$))))) , (Mul ((Sub ((Sub ((Div (
(Xcos ((($v_4(25)\cdot$-5.5)) , ($u_4(59)$))) , (Sub ((($v_4(26)\cdot$-5.5)) , (($v_4(27)\cdot$-10.0)))))) , (Sub ((
($v_4(28)\cdot$4.5)) , (($v_4(29)\cdot$-5.0)))))) , (Add ((Xcos ((Sub ((($v_4(30)\cdot$4.5)) , (($v_4(31)\cdot$-5.0))))
, ($u_4(60)$))) , (Mul ((Xlog ((($u_4(61)\cdot x_{t-3}$)) , ($u_4(62)$))) , (Xlog ((($v_4(32)\cdot$8.5)) , ($u_4(63)$
))))))))) , (Xexp ((($v_4(33)\cdot$-2.0)) , ($u_4(64)$))))))))))) , (Mul ((Sub ((Sub ((Div

((Xcos (((v_4(34)·-5.5)) , (u_4(65)))) , (Sub ((Xcos (((v_4(35)·-5.5)) , (u_4(66)))) , (Add ((Xexp ((Xexp (((v_4(36)·-5.5)) , (u_4(67)))) , (u_4(68)))) , (Mul ((Xlog (((u_4(69)·x_{t-3})) , (u_4(70)))) , (Xlog (((v_4(37)·8.5)) , (u_4(71))))))))))) , (Xexp ((Add (((v_4(38)·2.0)) , ((v_4(39)·-10.0)))) , (u_4(72)))))) , (Add ((Xexp ((Xexp (((v_4(40)·-5.5)) , (u_4(73)))) , (u_4(74)))) , (Mul ((Xlog (((u_4(75)·x_{t-3})) , (u_4(76)))) , (Xlog (((v_4(41)·8.5)) , (u_4(77)))))))))) , (Xexp (((v_4(42)·-2.0)) , (u_4(78)))))))

5. Rekursionsstufe

v_5 = {1.0, 0.99981921138819, 1.00111615504002, 0.99989266890279, 1.00000272397479, 1.00000238624130, 0.98688448038467, 1.01186286581497, 1.00083980533589, 1.00001325861658, 1.00000943208853, 1.02191790274193, 0.98657272012999, 0.98657321629427, 0.99377181423964, 1.00514129871894, 1.00000000000000, 0.99981921138271, 1.00111615493629, 0.96143062918802, 0.96143230868709, 0.97134949862429, 0.98688448411779, 0.98444349608936, 0.99927761328869, 0.99927756000419, 0.99998708934137, 1.00297216859371, 0.98688444563616, 0.98688448320184, 1.01186289440898, 1.00065841082794, 0.99998993635822, 0.99999391792337, 0.99999833712215}

u_5 = {7.64527030118287, 2.74451172459327, -10.000000000000, -9.9989103192661, 1.09480224167386, 4.13394760712780, 2.24925613361578, 4.14111563083367, -9.8822606872185, 1.73522061393048, -0.4453183673581, -2.2748401483429, 0.47004859497785, -9.2807448462046, 6.80740591564751, -2.9428164542915, -2.9570678114711, 3.13968867613140, 3.58092226590711, -8.9389379928478, -10.000000000000, 0.27356750376254, 1.77984351444304, -6.8728213813779, 1.82743300422383, 1.98972411808063, 1.16198734770406, -0.9596351045946, -5.5933436530330, -10.000000000000, 5.63933414820638, 0.74713573581165, -1.3947152267950, -4.7238725965467, -1.9406267504201, -5.5828909798398, 0.98317185848127, -0.2837063026545, 1.15722024204242, -10.000000000000, 4.41549794672485, 3.93724857497149, -0.6064168485391, 5.45556523059236, -10.000000000000, 1.50027817222930, -0.6759119755263, -1.9694888553163, -10.000000000000, 5.76147829197060, 0.40184112164706, 0.87161373320720, -1.2922853204668, 3.46542528718352, 3.00695559947444, -9.5436605696227, -1.6762155967290, -9.6568747985861, 0.07050441122026, -6.9570885138872, -1.5826811688781, -0.0904691612625, 2.22578123541413, -10.000000000000, -10.000000000000, 1.47521534426457, 8.54750289074980, -1.6981698955686, 2.44750258997954, 5.05092794322877, -10.000000000000, -8.0797947849416, -0.9305674270836, 1.86635386803540, -5.8766793465699, -6.1629922381361, -3.0685149287400, -2.3537693768672, 3.92870752883601, 10.0000000000000, -10.000000000000, -3.1594913685619, 6.95282113157722, 0.25382764738119, -1.4548901219994, 4.13046062186902, 2.77658617257765, -3.1311747228960, -10.000000000000, 0.03226695761375, 3.07800938275266, 6.96069202686817,

2.52150316007867, 3.14457810734431, -3.5616234879410, -4.6539894729128, 4.91371781927688, -10.000000000000, -3.8154580303658, -2.7880633047479, 1.79727564160835, -0.0263942164484, -2.0123992980575, -5.5760168611310, -4.1217366972495, -2.6392722712729, 2.27273034211806, -5.2436172722416, 10.0000000000000, 0.63576703934191, 3.26386816132676, -4.1483199092321, 10.0000000000000, -0.4353644348364, 5.62583397598604, 3.09361778641302, 1.08205385924636, -4.5766458238213, 0.55737083809743, -6.5608763881640, 4.51066876606400, -2.8259537819194, -3.2657526528945, 4.12231763250853, 8.24415656907254, -3.4498991675255, -2.8535615858479, -7.6985181312687, -2.5126769803600, 9.11764385257668, 7.08947849960100, -0.3472016842777}

$\hat{f}_{\text{sub}_5}$ = Add ((Sub ((Sub ((Sub ((Add ((Add ((Sub ((Add ((Sub ((Xcos ((Sub ((Xcos ((Xlog ((($v_5(1)$·1.0)) , ($u_5(1)$))) , ($u_5(2)$))) , (Mul ((($v_5(2)$·1.5)) , (Sub ((Sub ((Div ((($u_5(3)$·x_{t-4})) , (($u_5(4)$·x_{t-6})))) , (($u_5(5)$·x_{t-70})))) , (Xcos ((Xlog ((($v_5(3)$·1.5)) , ($u_5(6)$))) , ($u_5(7)$))))))))) , ($u_5(8)$))) , (Xsin ((($u_5(9)$·x_{t-70})) , ($u_5(10)$))))) , (Xcos ((($u_5(11)$·x_{t-8})) , ($u_5(12)$))))) , (Xlog ((Sub ((Xcos ((Sub ((Xsin ((Xlog ((($v_5(4)$·1.5)) , ($u_5(13)$))) , ($u_5(14)$))) , (Mul ((($v_5(5)$·1.5)) , (Sub ((Sub ((Div ((($u_5(15)$·x_{t-4})) , (($u_5(16)$·x_{t-6})))) , (Xsin ((($u_5(17)$·x_{t-70})) , ($u_5(18)$))))) , (Mul ((($u_5(19)$·x_{t-90})) , (($v_5(6)$·-10.0)))))))))) , ($u_5(20)$))) , (Xsin ((($u_5(21)$·x_{t-70})) , ($u_5(22)$))))) , ($u_5(23)$)))) , (Div ((Xlog ((($v_5(7)$·1.5)) , ($u_5(24)$))) , (($v_5(8)$·5.0)))))) , (Xcos ((($u_5(25)$·x_{t-8})) , ($u_5(26)$))))) , (Sub ((Xlog ((Add ((Sub ((Add ((Sub ((Xcos ((Sub ((Xcos ((Xlog ((($v_5(9)$·1.5)) , ($u_5(27)$))) , ($u_5(28)$))) , (Mul ((($v_5(10)$·1.5)) , (Sub ((Sub ((Div ((($u_5(29)$·x_{t-4})) , (($u_5(30)$·x_{t-6})))) , (Xsin ((($u_5(31)$·x_{t-70})) , ($u_5(32)$))))) , (Mul ((($u_5(33)$·x_{t-90})) , (($v_5(11)$·-10.0)))))))))) , ($u_5(34)$))) , (Xsin ((($u_5(35)$·x_{t-70})) , ($u_5(36)$))))) , (Add ((Sub ((Xcos ((Sub ((Xsin ((($u_5(37)$·x_{t-70})) , ($u_5(38)$))) , (($v_5(12)$·1.5)))) , ($u_5(39)$))) , (Xsin ((($u_5(40)$·x_{t-70})) , ($u_5(41)$))))) , (Xcos ((($u_5(42)$·x_{t-8})) , ($u_5(43)$))))))) , (Xlog ((Xlog ((Sub ((Xcos ((Xlog (((Add ((Sub ((($v_5(13)$·-8.5)) , (($v_5(14)$·3.0)))) , (Mul ((($u_5(44)$·x_{t-110})) , (($v_5(15)$·7.0)))))) , ($u_5(45)$))) , ($u_5(46)$))) , (Xcos ((Xlog ((($v_5(16)$·1.5)) , ($u_5(47)$))) , ($u_5(48)$))))) , ($u_5(49)$))) , ($u_5(50)$))))) , (Xcos ((($u_5(51)$·x_{t-8})) , ($u_5(52)$))))) , ($u_5(53)$))) , (Add ((Add ((Sub ((Add ((Sub ((Xcos ((Sub ((Xcos ((Xlog ((($v_5(17)$·1.0)) , ($u_5(54)$))) , ($u_5(55)$))) , (Mul ((($v_5(18)$·1.5)) , (Sub ((Sub ((Div ((($u_5(56)$·x_{t-4})) , (($u_5(57)$·x_{t-6})))) , (($u_5(58)$·x_{t-70})))) , (Xcos ((Xlog ((($v_5(19)$·1.5)) , ($u_5(59)$))) , ($u_5(60)$))))))))) , ($u_5(61)$))) , (Xsin ((($u_5(62)$·x_{t-70})) , ($u_5(63)$))))) , (Sub ((Xcos ((Xlog ((Add ((Sub ((($v_5(20)$·-8.5)) , (($v_5(21)$·3.0)))) , (Mul ((($u_5(64)$·x_{t-110})) , (($v_5(22)$·7.0)))))) , ($u_5(65)$))) , ($u_5(66)$))) , (Xcos ((Xlog ((($v_5(23)$·1.5)) , ($u_5(67)$))) , ($u_5(68)$))))))) , (Xlog ((Xlog ((Sub ((Xcos ((Xlog ((($v_5(24)$·1.0)) , ($u_5(69)$))) , ($u_5(70)$))) , (Xcos ((Sub ((Div ((Add ((Sub ((($v_5(25)$·-8.5)) , (($v_5(26)$·3.0)))) , (Mul ((($u_5(71)$·x_{t-110})) , (($v_5(27)$·7.0)))))) , (Div ((($u_5(72)$·x_{t-4})) , (($u_5(73)$·x_{t-6})))))) , (Xlog ((Xlog ((Mul ((($u_5(74)$·x_{t-90})) , (($v_5(28)$·-10.0)))) , ($u_5(75)$))) , ($u_5(76)$))))) , ($u_5(77)$))))) , ($u_5(78)$))) , ($u_5(79)$))))) , (Xlog ((($v_5(29)$·1.0)) , ($u_5(80)$))))) , (Div ((Xlog ((($v_5(30)$·1.5)) , ($u_5(81)$))) , (($v_5(31)$·5.0))))))))) , (Xlog ((Add ((Sub ((Add ((Sub ((Xcos ((Sub ((Xsin ((($u_5(82)$·x_{t-70})) , ($u_5(83)$))) , (($v_5(32)$·1.5)))) , ($u_5(84)$))) , (Xsin ((($u_5(85)$·x_{t-70})) , ($u_5(86)$))))) , (Xcos ((($u_5(87)$·x_{t-8})) , ($u_5(88)$))))) ,

(Xlog ((Xlog ((Sub ((Xcos ((Sub ((Div ((($u_5(89)\cdot x_{t-4}$)) , (($u_5(90)\cdot x_{t-6}$)))) , (Xcos ((($u_5(91)\cdot x_{t-8}$)) , ($u_5(92)$))))) , ($u_5(93)$))) , (Xsin ((($u_5(94)\cdot x_{t-70}$)) , ($u_5(95)$))))) , ($u_5(96)$))) , ($u_5(97)$))))) , (Xlog ((Sub ((Div ((($u_5(98)\cdot x_{t-4}$)) , (($u_5(99)\cdot x_{t-6}$)))) , (Xsin ((($u_5(100)\cdot x_{t-70}$)) , ($u_5(101)$))))) , ($u_5(102)$))))) , ($u_5(103)$))))) , (Xlog ((Add ((Sub ((Add ((Sub ((Xcos ((Sub ((Xcos ((Div ((($u_5(104)\cdot x_{t-4}$)) , (($u_5(105)\cdot x_{t-6}$)))) , ($u_5(106)$))) , (Mul ((($v_5(33)$·1.5)) , (Sub ((Sub ((Div ((($u_5(107)\cdot x_{t-4}$)) , (($u_5(108)\cdot x_{t-6}$)))) , (Xsin ((($u_5(109)\cdot x_{t-70}$)) , ($u_5(110)$))))) , (Mul ((($u_5(111)\cdot x_{t-90}$)) , (($v_5(34)$·-10.0))))))))) , ($u_5(112)$))) , (Xsin ((($u_5(113)\cdot x_{t-70}$)) , ($u_5(114)$))))) , (Xcos ((($u_5(115)\cdot x_{t-8}$)) , ($u_5(116)$))))) , (Xlog ((Xlog ((Sub ((Xcos ((Xlog ((Xsin ((($u_5(117)\cdot x_{t-70}$)) , ($u_5(118)$))) , ($u_5(119)$))) , ($u_5(120)$))) , (Xcos ((Mul ((($u_5(121)\cdot x_{t-110}$)) , (($v_5(35)$·7.0)))) , ($u_5(122)$))))) , ($u_5(123)$))) , ($u_5(124)$))))) , (Xcos ((($u_5(125)\cdot x_{t-8}$)) , ($u_5(126)$))))) , ($u_5(127)$))))) , (Xlog ((Sub ((Div ((($u_5(128)\cdot x_{t-4}$)) , (($u_5(129)\cdot x_{t-6}$)))) , (Xsin ((($u_5(130)\cdot x_{t-70}$)) , ($u_5(131)$))))) , ($u_5(132)$))))

B.3.2 Lernen mit den gefilterten Trainingsdaten

$a_0 = \{$-0.00575776147493$\}$

$a_1 = \{0.99989451749779\}$

$a_2 = \{0.98562325121201\}$

$a_3 = \{1.0\}$

$$\hat{f}_{\text{sum}_n} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) + (a_3 \cdot \hat{f}_{\text{sub}_3})$$

1. Rekursionsstufe

$v_1 = \{1.0,\ 1.0\}$

$u_1 = \{1.0,\ 1.00000001490116,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0\}$

$\hat{f}_{\text{sub}_1}$ = Sub ((Add ((($u_1(1)\cdot x_{t-1}$)) , (Xcos ((Xcos ((Xcos ((Xcos ((Sub ((($u_1(2)\cdot x_{t-1}$)) , (Add ((($u_1(3)\cdot x_{t-1}$)) , (Xcos ((Xcos ((Sub ((($u_1(4)\cdot x_{t-1}$)) , (Add ((($u_1(5)\cdot x_{t-1}$)) , (Xsin ((Xlog ((Add ((($u_1(6)\cdot x_{t-1}$)) , (Xcos ((($v_1(1)$·-9.0)) , ($u_1(7)$))))) , ($u_1(8)$))) , ($u_1(9)$)))))) , ($u_1(10)$))) , ($u_1(11)$))))))) , ($u_1(12)$))) , ($u_1(13)$))) , ($u_1(14)$))) , ($u_1(15)$))))) , (Sub ((($u_1(16)\cdot x_{t-2}$)) , (Sub ((($u_1(17)\cdot x_{t-1}$)) , (Xcos ((Xcos ((Xexp ((Xcos ((Xlog ((Add ((($u_1(18)\cdot x_{t-1}$)) , (Xlog ((Add ((($u_1(19)\cdot x_{t-1}$)) , (Xsin ((Xexp ((Xcos ((($v_1(2)$·-9.0)

) , ($u_1(20)$))) , ($u_1(21)$))) , ($u_1(22)$))))) , ($u_1(23)$))))) , ($u_1(24)$))) , ($u_1(25)$))) , ($u_1(26)$))) , ($u_1(27)$))) , ($u_1(28)$))))))))

2. Rekursionsstufe

$v_2 = \{$1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.00000001490116, 1.0$\}$

$u_2 = \{$1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.00000001490116, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0$\}$

$\hat{f}_{\mathrm{sub}_2}$ = Div ((Mul ((Sub ((Mul ((($v_2(1)\cdot$-8.5)) , (($v_2(2)\cdot$5.0)))) , (Sub ((Mul ((Sub ((Add ((Sub ((Xcos ((Xexp ((Xexp ((Xexp ((Xexp ((($u_2(1)\cdot x_{t-2}$)) , ($u_2(2)$))) , ($u_2(3)$))) , ($u_2(4)$))) , ($u_2(5)$))) , ($u_2(6)$))) , (Xcos ((($v_2(3)\cdot$-6.0)) , ($u_2(7)$))))) , (Xcos ((Mul ((Add ((Mul ((($v_2(4)\cdot$-8.5)) , (($v_2(5)\cdot$5.0)))) , (Xexp ((Mul ((Xlog ((Xsin ((Xlog ((($v_2(6)\cdot$-1.0)) , ($u_2(8)$))) , ($u_2(9)$))) , ($u_2(10)$))) , (Xcos ((($v_2(7)\cdot$-4.5)) , ($u_2(11)$))))) , ($u_2(12)$))))) , (Xlog ((Sub ((($u_2(13)\cdot x_{t-110}$)) , (($v_2(8)\cdot$-0.5)))) , ($u_2(14)$))))) , ($u_2(15)$))))) , (Sub ((Mul ((Xlog ((Sub ((($u_2(16)\cdot x_{t-110}$)) , (($v_2(9)\cdot$-0.5)))) , ($u_2(17)$))) , (Xsin ((($v_2(10)\cdot$3.5)) , ($u_2(18)$))))) , (Sub ((($u_2(19)\cdot x_{t-110}$)) , (($v_2(11)\cdot$-0.5)))))))) , (Xcos ((Sub ((Mul ((Xsin ((($v_2(12)\cdot$3.5)) , ($u_2(20)$))) , (Sub ((Xcos ((Xexp ((Xexp ((Xexp ((($u_2(21)\cdot x_{t-2}$)) , ($u_2(22)$))) , ($u_2(23)$))) , ($u_2(24)$))) , ($u_2(25)$))) , (Xsin ((Xsin ((Div ((Xlog ((Xexp ((Xexp ((Xexp ((($u_2(26)\cdot x_{t-2}$)) , ($u_2(27)$))) , ($u_2(28)$))) , ($u_2(29)$))) , ($u_2(30)$))) , (($u_2(31)\cdot x_{t-16}$)))) , ($u_2(32)$))) , ($u_2(33)$))))))) , (Xlog ((Sub ((($u_2(34)\cdot x_{t-110}$)) , (($v_2(13)\cdot$-0.5)))) , ($u_2(35)$))))) , ($u_2(36)$))))) , (Sub ((($u_2(37)\cdot x_{t-110}$)) , (($v_2(14)\cdot$-0.5)))))))) , (Xcos ((Sub ((Xsin ((Sub ((Mul ((Mul ((Add ((Xlog ((($v_2(15)\cdot$-1.0)) , ($u_2(38)$))) , (Xsin ((($v_2(16)\cdot$5.5)) , ($u_2(39)$))))) , (Sub ((Xcos ((Xexp ((Xexp ((Xexp ((Xexp ((($u_2(40)\cdot x_{t-2}$)) , ($u_2(41)$))) , ($u_2(42)$))) , ($u_2(43)$))) , ($u_2(44)$))) , ($u_2(45)$))) , (Xcos ((($v_2(17)\cdot$-6.0)) , ($u_2(46)$))))))) , (Xsin ((($v_2(18)\cdot$5.5)) , ($u_2(47)$))))) , (Xlog ((Div ((Xsin ((Div ((Xlog ((Xexp ((Xexp ((Xexp ((($u_2(48)\cdot x_{t-2}$)) , ($u_2(49)$))) , ($u_2(50)$))) , ($u_2(51)$))) , ($u_2(52)$))) , (($u_2(53)\cdot x_{t-16}$)))) , ($u_2(54)$))) , (($u_2(55)\cdot x_{t-16}$)))) , ($u_2(56)$))))) , ($u_2(57)$))) , (Xlog ((Sub ((($u_2(58)\cdot x_{t-110}$)) , (($v_2(19)\cdot$-0.5)))) , ($u_2(59)$))))) , ($u_2(60)$))))) , (Div ((Xexp ((Xexp ((Xexp ((Xlog ((Mul ((($v_2(20)\cdot$-8.5)) , (($v_2(21)\cdot$5.0)))) , ($u_2(61)$))) , ($u_2(62)$))) , ($u_2(63)$))) , ($u_2(64)$))) , (Add ((Xlog ((Xsin ((Xlog ((($v_2(22)\cdot$-1.0)) , ($u_2(65)$))) , ($u_2(66)$))) , ($u_2(67)$))) , (Xlog ((Div ((Xsin ((Div ((Xlog ((Xexp ((Xexp ((Xexp ((($u_2(68)\cdot x_{t-2}$)) , ($u_2(69)$))) , ($u_2(70)$))) , ($u_2(71)$))) , ($u_2(72)$))) , (($u_2(73)\cdot x_{t-16}$)))) , ($u_2(74)$))) , (($u_2(75)\cdot x_{t-16}$)))) , ($u_2(76)$))))))))

3. Rekursionsstufe

v_3 = {-1.8292486920367, -0.1923765828242, -1.1038412921683, -0.7419025101309, -0.3554055697478, -1.1450447797584, 0.34014759721885, 0.96014956427048, 1.50680564222280, -1.6726114794431, -0.2523597036500, -1.7555473778193, 0.44140404260064, 1.86141428533790, 0.32777032018901, 0.85177738976033, 1.82495008084099, 1.86966116311116, 1.16646679874789, 0.19717341273980, -1.0107949807901, 1.69239568318158, -0.5369514461996, -1.9311539700200, 0.02461275813697, -1.9799246654728, -1.5599501261357, -0.9042263111385, 1.54445199264957, 0.62909620504271, -1.2190867409898, 0.67104915510205, 1.03893059557832, -0.1895237488338, -0.8386616653079}

u_3 = {0.26477431566196, 1.94214326118477, -1.0523227282568, -1.6392167374555, 1.06894872661733, -0.2504340750512, -1.7955547948179, -0.1610400222610, -0.2559442813077, -0.3138558077794, -0.6015190931498, -0.5639276107708, 1.94933593122338, -1.6866529984631, 1.12734484435146, -0.9302900675628, 0.82935745249619, 1.05897875222563, -0.9691486660795, 1.04583342350944, 1.79326252455026, 1.90448901609930, -0.8975738799235, -1.5058634372373, -1.2140649103762, -1.3939019605323, -0.4453732220399, -1.6058527808895, -0.5499339354652, -2.0000000000000, 0.70251706420747, 1.07934627208754, 0.53149833489236, -1.9513251182587, -1.9909068251063, 1.37507441231840, -1.1344311696874, -0.4665485047830, 0.15111441832183, -1.9732408175865, 2.00000000000000, -0.0689779449786, 1.70389387115873, 0.77504414464937, 0.54194882734234, -0.0871484380670, -1.0076619961584, 0.57738597418421, -1.1183234711142, -0.6060742611180, 0.53762633303473, 1.57016063429838, -1.9989546706869}

$\hat{f}_{\text{sub}_3}$ = Mul ((Mul ((Mul ((Xsin ((Xcos ((Div ((Div ((Xexp ((Mul ((Xsin ((Xexp ((Mul (((v_3(1)·-10.0)) , ((0.0)))) , (u_3(1)))) , (u_3(2)))) , (Mul ((Mul (((v_3(2)·-10.0)) , ((0.0)))) , (Xexp ((Div ((Xsin (((0.0)) , (u_3(3)))) , (Div (((u_3(4)·x_{t-9})) , ((v_3(3)·8.0)))))) , (u_3(5)))))))) , (u_3(6)))) , (Mul ((Mul ((Div ((Add (((v_3(4)·7.0)) , ((v_3(5)·-7.5)))) , (Xexp ((Xlog (((v_3(6)·-1.5)) , (u_3(7)))) , (u_3(8)))))) , (Xexp ((Add (((v_3(7)·7.0)) , ((v_3(8)·-7.5)))) , (u_3(9)))))) , (Xsin ((Xcos (((v_3(9)·-4.5)) , (u_3(10)))) , (u_3(11)))))))) , (Xlog (((u_3(12)·x_{t-30})) , (u_3(13)))))) , (u_3(14)))) , (u_3(15)))) , (Mul ((Div ((Add (((v_3(10)·7.0)) , ((v_3(11)·-7.5)))) , (Sub (((v_3(12)·9.0)) , ((v_3(13)·4.5)))))) , (Xexp ((Div ((Xsin ((Xcos ((Div ((Div ((Xexp ((Mul ((Xsin ((Xexp ((Mul (((v_3(14)·-10.0)) , ((0.0)))) , (u_3(16)))) , (u_3(17)))) , (Mul ((Div ((Add (((v_3(15)·7.0)) , ((v_3(16)·-7.5)))) , (Sub (((v_3(17)·9.0)) , ((v_3(18)·4.5)))))) , (Xexp ((Div ((Xsin (((0.0)) , (u_3(18)))) , (Div (((u_3(19)·x_{t-9})) , ((v_3(19)·8.0)))))) , (u_3(20)))))))) , (u_3(21)))) , (Mul ((Mul ((Div ((Add (((v_3(20)·7.0)) , ((v_3(21)·-7.5)))) , (Xexp ((Xlog (((v_3(22)·-1.5)) , (u_3(22)))) , (u_3(23)))))) , (Xexp ((Add (((v_3(23)·7.0)) , ((v_3(24)·-7.5)))) , (u_3(24))))))) ,

(Xsin ((Xcos (((v_3(25)·-4.5)) , (u_3(25)))) , (u_3(26))))))) , (Xlog (((u_3(27)·x_{t-30})) , (u_3(28)))))) , (u_3(29)))) , (u_3(30)))) , (Mul ((Xsin ((Xcos ((Sub ((Xlog (((v_3(26)·-1.5)) , (u_3(31)))) , (Xsin ((Div ((Xsin ((Xsin ((Xcos ((Mul (((v_3(27)·1.5)) , ((v_3(28)·-0.5)))) , (u_3(32)))) , (u_3(33)))) , (u_3(34)))) , (Mul (((v_3(29)·-10.0)) , ((0.0)))))) , (u_3(35))))) , (u_3(36)))) , (u_3(37)))) , (Xexp ((Xsin ((Xcos ((Mul (((v_3(30)·1.5)) , ((v_3(31)·-0.5)))) , (u_3(38)))) , (u_3(39)))) , (u_3(40)))))))) , (u_3(41)))))))) , (Xsin ((Div ((Xlog ((Xcos (((u_3(42)·x_{t-6})) , (u_3(43)))) , (u_3(44)))) , (Mul (((v_3(32)·-10.0)) , ((0.0)))))) , (u_3(45)))))) , (Xcos ((Add ((Mul ((Xexp ((Xlog (((v_3(33)·-1.5)) , (u_3(46)))) , (u_3(47)))) , (Xsin ((Xsin (((v_3(34)·3.5)) , (u_3(48)))) , (u_3(49))))))) , (Xexp ((Xlog ((Div (((u_3(50)·x_{t-9})) , ((v_3(35)·8.0)))) , (u_3(51)))) , (u_3(52)))))) , (u_3(53)))))

B.4 Jährliche durchschnittliche Sonnenfleckenrelativzahl

B.4.1 Lernen mit den originalen Trainingsdaten

$a_0 = \{$-2.33354899074079$\}$

$a_1 = \{1.05268217936234\}$

$a_2 = \{1.32279336887895\}$

$a_3 = \{1.15238773262259\}$

$a_4 = \{1.11943653315843\}$

$a_5 = \{0.97911264282707\}$

$$\hat{f}_{\text{sum}_n} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) + ... + (a_5 \cdot \hat{f}_{\text{sub}_5})$$

1. Rekursionsstufe

$v_1 = \{$-4.6291974307364, 1.15766665504953, 6.02856813327827, -10.000000000000, 4.95791968686557, 9.27685018588359, 5.46840206489350, -10.000000000000, 6.10593326903219, -4.9229510912886, -6.0130494928661, -1.0676236153772, -10.000000000000, 3.52406699618989, 9.46641347755145, 9.40849359616226, 10.0000000000000, 6.64030143123624, -3.6502845432516, 2.76539287223701, -5.7049403063789, -1.3019108669904, 0.40058752734659, 6.38539801120065, -10.000000000000, 0.21862553536389, 5.97131206715924, 10.0000000000000, -2.5331769772455, -7.8508152768764, -7.9440529715389, -2.6291840450762, -1.9125228604059, 0.54686080324540, -4.5461946588167, -9.3853645873862,

1.18461281827018, 0.26581731814894, -10.000000000000, 3.83964474057961, 4.72968347973133, -5.6820552804445, 0.89097628497630, 10.0000000000000, 1.42119717643660}

u_1 = {0.99999994483317, 0.98819732522914, 0.19206932394058, 0.99999995086768, 0.98986228214242, 1.79919187867388, 1.06372529974795, 6.00450826114319, 1.07493276812940, 1.64665443691638, 1.10655717148455, 1.00000001249412, 0.97719170924050, 1.26179392865802, 0.73644985220041, 1.03199612961921, 1.00731021870874, 0.99417906678600, 0.99404607225134, 1.00017187621408, 0.98988651764630, 1.04864230290252, 1.84924733271292, 0.99999992361439, 1.00208089290625, 6.99456583021610, 1.00059368499073, 1.00000000000000, 1.00514462295989, 0.98678361521540, 0.99557557177806, 1.00832521970892, 1.00581930940270, 0.99976960478040, 0.99976960481567, 1.06154134623296, 2.23828632655535, 0.71333089635805, 0.99999999537649, 0.97791096271577, 1.99534711886609, 1.01852233469247, 0.99999089481229, 1.05889840924811, 0.78893863460572, 1.21837288712852, 0.84289201820919, 1.86865069361892, 1.22630774552463, 0.99999996146413, 1.00414690551062, 1.01639742042706, 2.86779889441367, -0.3946527713130, 0.99999990177397, 1.01280966163798, 0.80274856181914, 1.00265591733507, 0.99448565703413, 1.01019569848588, 1.00765264916060, 1.00909411400740, 1.00148339524360, 1.00178714068436, 0.99958172200145, 0.99958172185665, 2.79803488960412, 0.99068439800287, 1.96586490498737, -0.0246150608335, 0.99999997739436, 0.95549018519186, 0.88363041188717, 1.11537128137932, 0.97764306712470, 1.09330123792002, 1.06856471737027, 1.00750450087207, 1.00178328598543, 0.99737293892454, 0.99979884276187}

$\hat{f}_{\text{sub}_1}$ = Div ((Add ((Xcos ((Add ((Xsin ((Mul ((Sub ((-4.5) , (($u_1(1)$·x_{t-2})))) , (Div (($v_1(1)$·2.0) , (($u_1(2)$·x_{t-8})))))) , ($u_1(3)$))) , (Add ((Mul ((Sub (($v_1(2)$·-4.5) , (($u_1(4)$·x_{t-2})))) , (Div (($v_1(3)$·2.0) , (($u_1(5)$·x_{t-8})))))) , (Xsin (($v_1(4)$·-4.0) , ($u_1(6)$))))))) , ($u_1(7)$))) , (Add ((Xsin ((Add ((Add ((Xcos ((Xcos ((Add ((Xsin ((Xsin ((Add ((Div (($v_1(5)$·2.0) , (($u_1(8)$·x_{t-8})))) , (Sub (($v_1(6)$·-4.5) , (($u_1(9)$·x_{t-2})))))) , ($u_1(10)$))) , ($u_1(11)$))) , (Add ((Mul ((Sub (($v_1(7)$·-4.5) , (($u_1(12)$·x_{t-2})))) , (Div (($v_1(8)$·2.0) , (($u_1(13)$·x_{t-8})))))) , (Add ((Xcos ((Xsin (($v_1(9)$·11) , ($u_1(14)$))) , ($u_1(15)$))) , (Sub (($v_1(10)$·-4.5) , (($u_1(16)$·x_{t-2}))))))))) , ($u_1(17)$))) , ($u_1(18)$))) , (Add ((Xlog ((Mul (($v_1(11)$·-4.5) , (($u_1(19)$·x_{t-2})))) , ($u_1(20)$))) , (Sub (($v_1(12)$·-4.5) , (($u_1(21)$·x_{t-2})))))))) , (Add ((Xcos ((Add ((Xsin ((Div (($v_1(13)$·2.0) , (($u_1(22)$·x_{t-8})))) , ($u_1(23)$))) , (Add ((Mul ((Sub (($v_1(14)$·-4.5) , (($u_1(24)$·x_{t-2})))) , (Div (($v_1(15)$·2.0) , (($u_1(25)$·x_{t-8})))))) , (Xsin (($v_1(16)$·-4.0) , ($u_1(26)$))))))) , ($u_1(27)$))) , (Add ((Xexp (($v_1(17)$·-4.0) , ($u_1(28)$))) , (Mul ((Xcos ((Xsin ((Xsin ((Add ((Div (($v_1(18)$·2.0) , (($u_1(29)$·x_{t-8})))) , (Xsin ((Div (($v_1(19)$·2.0) , (($u_1(30)$·x_{t-8})))) , ($u_1(31)$)))))) , ($u_1(32)$))) , ($u_1(33)$))) , ($u_1(34)$))) , (Mul ((($u_1(35)$·x_{t-1})) , ($v_1(20)$·4.0)))))))))) , ($u_1(36)$))) , (Add ((Xcos ((Add ((Xsin ((Div (($v_1(21)$·2.0) , (($u_1(37)$·x_{t-8})))) , ($u_1(38)$))) , (Add ((Mul ((Sub (($v_1(22)$·-4.5) , (($u_1(39)$·x_{t-2})))) , (Div

((v_1(23)·2.0) , ((u_1(40)·x_{t-8}))))) , (Xsin ((v_1(24)·-4.0) , (u_1(41)))))))) , (u_1(42)))) , (Add ((Xcos ((Add ((Xcos ((Xcos ((Mul ((Xsin ((Mul (((u_1(43)·x_{t-1})) , (v_1(25)·4.0))) , (u_1(44)))) , (Add ((Mul ((Xsin ((Div ((v_1(26)·2.0) , ((u_1(45)·x_{t-8})))) , (u_1(46)))) , (v_1(27)· (2/5.5)))) , (Sub ((v_1(28)·-4.5) , ((u_1(47)·x_{t-2})))))))) , (u_1(48)))) , (u_1(49)))) , (Mul ((Sub ((v_1(29)·-4.5) , ((u_1(50)·x_{t-2})))) , (Div ((v_1(30)·2.0) , ((u_1(51)·x_{t-8})))))))) , (u_1(52)))) , (Add ((Xcos ((Add ((Xsin ((Div ((2.0) , ((u_1(53)·x_{t-8})))) , (u_1(54)))) , (Add ((Mul ((Sub ((v_1(31)·-4.5) , ((u_1(55)·x_{t-2})))) , (Div ((v_1(32)·2.0) , ((u_1(56)·x_{t-8})))))) , (Xsin ((v_1(33)·-4.0) , (u_1(57)))))))) , (u_1(58)))) , (Add ((Xexp ((v_1(34)·-4.0) , (u_1(59)))) , (Mul ((Xcos ((Xsin ((Xsin ((Add ((Div ((v_1(35)·2.0) , ((u_1(60)·x_{t-8})))) , (Xsin ((Div ((v_1(36)·2.0) , ((u_1(61)·x_{t-8})))) , (u_1(62)))))) , (u_1(63)))) , (u_1(64)))) , (u_1(65)))) , (Mul (((u_1(66)·x_{t-1})) , (v_1(37)·4.0))))))))))))))))))) , (Xlog ((Add ((Xcos ((Xcos ((Add ((Xsin ((Xsin ((Add ((Div ((v_1(38)·2.0) , ((u_1(67)·x_{t-8})))) , (Sub ((v_1(39)·-4.5) , ((u_1(68)·x_{t-2}))))) , (u_1(69)))) , (u_1(70)))) , (Add ((Mul ((Sub ((v_1(40)·-4.5) , ((u_1(71)·x_{t-2})))) , (Div ((v_1(41)·2.0) , ((u_1(72)·x_{t-8})))))) , (Add ((Xcos ((Xsin ((v_1(42)·11) , (u_1(73)))) , (u_1(74)))) , (Sub ((v_1(43)·-4.5) , ((u_1(75)·x_{t-2})))))))))) , (u_1(76)))) , (u_1(77)))) , (Add ((Xlog ((Mul ((v_1(44)·-4.5) , ((u_1(78)·x_{t-2})))) , (u_1(79)))) , (Sub ((v_1(45)·-4.5) , ((u_1(80)·x_{t-2})))))))) , (u_1(81)))))

2. Rekursionsstufe

v_2 = {-9.99999999999998, 2.41151798989646, -3.3345989977552, 3.79601757176831, 2.56310071144439, 1.40527990493915, -6.1169179849652, -3.8130333960415, 2.66312274713770, 2.17564250869102, -6.5307820733634, -3.9355833490909, -3.8643638382589, -0.2267862094729, 0.41967942015315, -1.1847934138130, -1.5673405261565, 0.10335661406692, 0.84959533590282, 8.15751664781202, 1.86989216078196, -2.0779204583690, 0.64516333729512, -1.8052598774761, 0.56348286906211, -1.3978918624369, 0.34304292559876, -1.4071695090769, -0.7201768589324, 2.26062363979709, 0.46246315273683, -9.9999999999309, -9.9445706768353, -0.8960834564895, 8.74294841119275, 1.76131378780472, -6.6252274885813, 7.20532504073699, 1.47935213815704, 0.28466963115317, 0.10652112333616, -0.6380658163488, 2.26276096542180, 9.35069835287356, 7.30327019898680, 7.22499363845475, -2.2995759551182, -7.7962379238892, -5.0117461777043, 9.51808057368526}

u_2 = {1.09174307081978, 1.01939261799254, 1.03221908812842, 0.87219818739627, 1.06619214662912, 0.99998491999919, 1.02909752703836, 0.98751165719629, 1.01109389513171, 1.00916988313888, 1.02669819585563, 0.99652766200018, 1.00180391530161, 1.02480184216491, 1.00916987389444, 1.02146452239580, 1.07075512573085, 1.00916989467933, 1.07754996845892, 1.04415704908693, 1.25349121891229, 1.24462889132383, 1.01135853725839, 1.27292384278496, 1.00269988828482, 1.00197507643490, 0.97455786197646, 1.00271307888082, 0.99444578321959, 1.00042525147767, 0.93013939524562, 1.08404972308580,

1.61834222364211, 0.95478940336562, 1.01704756114351, 1.01719174877759,
1.05768698015474, 0.99918986292272, 1.02920134147601, 0.99799102363247,
1.12381997906211, 1.03260363573618, 1.01237956425192, 1.08634839824289,
1.01109071152649, 1.01719179970549, 1.36769563615316, 1.02158838640626,
0.99719018082897, 1.06438282683805, 0.99545321647799, 1.08280097242275,
0.99978699323827, 0.99998925187853, 1.01719182999359, 1.05078425335281,
1.01719174202268, 1.12982556653583, 1.05693828391406, 0.99929032743484,
1.35674781831867, 0.99096697827197, 1.00777270417014, 1.01719178701365,
1.01634817078789, 1.00293699899780, 1.00287499122022, 0.99383949824764,
1.00276196900716, 0.99818554046624, 1.00033656005592, 0.99967852365146,
0.99979382965009, 0.99960205294465, 1.01719181935041, 0.97843380723538}

$\hat{f}_{\text{sub}_2}$ = Add ((Add ((Xcos ((Xlog ((Xcos ((Sub ((Xlog ((($u_2(1)$·x_{t-4})) , ($u_2(2)$))) , ($v_2(1)$·-3.5))) , ($u_2(3)$))) , ($u_2(4)$))) , ($u_2(5)$))) , (Xcos ((Xlog ((Xcos ((Sub (($v_2(2)$·-7) , (Add ((Add ((Xcos ((Div ((Mul ((($u_2(6)$·x_{t-4})) , ($v_2(3)$·8.0))) , ($v_2(4)$·-3.5))) , ($u_2(7)$))) , (Xlog ((Xcos ((Sub ((Div ((Div ((($u_2(8)$·x_{t-4})) , ($v_2(5)$·8.0))) , ($v_2(6)$·-3.5))) , (Sub ((Div ((($u_2(9)$·x_{t-4})) , ($v_2(7)$·8.0))) , ($v_2(8)$·-7))))) , ($u_2(10)$))) , ($u_2(11)$))))) , (Add ((Xlog ((Xcos ((Sub ((Xcos ((Sub ((Xlog ((($u_2(12)$·x_{t-4})) , ($u_2(13)$))) , ($v_2(9)$·-3.5))) , ($u_2(14)$))) , (Sub (($v_2(10)$·-7) , ($v_2(11)$·-7))))) , ($u_2(15)$))) , ($u_2(16)$))) , (Xlog ((Xcos ((Div ((Div ((($u_2(17)$·x_{t-4})) , ($v_2(12)$·8.0))) , ($v_2(13)$·-7))) , ($u_2(18)$))) , ($u_2(19)$)))))))))) , ($u_2(20)$))) , ($u_2(21)$))) , ($u_2(22)$))))) , (Add ((Add ((Xcos ((Div ((Div ((($u_2(23)$·x_{t-4})) , ($v_2(14)$·8.0))) , ($v_2(15)$·-3.5))) , ($u_2(24)$))) , (Add ((Xcos ((Xlog ((Xcos ((Sub ((Xcos ((Sub ((Xlog ((Xcos ((Sub ((Xlog ((($u_2(25)$·x_{t-4})) , ($u_2(26)$))) , ($v_2(16)$·-3.5))) , ($u_2(27)$))) , ($u_2(28)$))) , ($v_2(17)$·-7))) , ($u_2(29)$))) , (Sub ((Div ((($u_2(30)$·x_{t-4})) , ($v_2(18)$·8.0))) , ($v_2(19)$·-7))))) , ($u_2(31)$))) , ($u_2(32)$))) , ($u_2(33)$))) , (Xlog ((Xcos ((Sub ((Xlog ((($u_2(34)$·x_{t-4})) , ($u_2(35)$))) , ($v_2(20)$·-3.5))) , ($u_2(36)$))) , ($u_2(37)$))))))) , (Add ((Add ((Add ((Xcos ((Div ((Div ((($u_2(38)$·x_{t-4})) , ($v_2(21)$·8.0))) , ($v_2(22)$·-3.5))) , ($u_2(39)$))) , (Add ((Xcos ((Div ((Mul ((($u_2(40)$·x_{t-4})) , ($v_2(23)$·8.0))) , ($v_2(24)$·-3.5))) , ($u_2(41)$))) , (Xlog ((Xcos ((Sub ((Xcos ((Sub ((Xlog ((($u_2(42)$·x_{t-4})) , ($u_2(43)$))) , ($v_2(25)$·-7))) , ($u_2(44)$))) , (Sub ((Div ((($u_2(45)$·x_{t-4})) , ($v_2(26)$·8.0))) , ($v_2(27)$·-7))))) , ($u_2(46)$))) , ($u_2(47)$))))))) , (Add ((Add ((Div ((Div ((($u_2(48)$·x_{t-4})) , ($v_2(28)$·8.0))) , ($v_2(29)$·-7))) , (Add ((Xcos ((Div ((Div ((($u_2(49)$·x_{t-4})) , ($v_2(30)$·8.0))) , ($v_2(31)$·-7))) , ($u_2(50)$))) , (Add ((Xcos ((Div ((Mul ((($u_2(51)$·x_{t-4})) , ($v_2(32)$·8.0))) , ($v_2(33)$·-3.5))) , ($u_2(52)$))) , (Xlog ((Xcos ((Sub ((Div ((Div ((($u_2(53)$·x_{t-4})) , ($v_2(34)$·8.0))) , ($v_2(35)$·-3.5))) , (Sub ((Div ((($u_2(54)$·x_{t-4})) , ($v_2(36)$·8.0))) , ($v_2(37)$·-7))))) , ($u_2(55)$))) , ($u_2(56)$))))))))) , (Add ((Xcos (($v_2(38)$·-7) , ($u_2(57)$))) , (Add ((Xcos ((Div ((Div ((($u_2(58)$·x_{t-4})) , ($v_2(39)$·8.0))) , ($v_2(40)$·-7))) , ($u_2(59)$))) , (Add ((Xcos ((Div ((Mul ((($u_2(60)$·x_{t-4})) , ($v_2(41)$·8.0))) , ($v_2(42)$·-3.5))) , ($u_2(61)$))) , (Xlog ((Xcos ((Add ((Div ((Div ((($u_2(62)$·x_{t-4})) , ($v_2(43)$·8.0))) , ($v_2(44)$·-3.5))) , (Sub ((Div ((($u_2(63)$·x_{t-4})) , ($v_2(45)$·8.0))) , ($v_2(46)$·-7))))) , ($u_2(64)$))) , ($u_2(65)$))))))))))))) , (Xlog ((Xcos ((Sub ((Xcos ((Xlog ((Xsin ((Sub ((Xcos ((Xlog ((($u_2(66)$·x_{t-4})) , ($u_2(67)$))) , ($u_2(68)$))) , (Sub ((Div ((($u_2(69)$·x_{t-4})) , ($v_2(47)$·8.0))) , ($v_2(48)$·-7))))) , ($u_2(70)$))) , ($u_2(71)$))) , ($u_2(72)$))) , (Xcos ((Sub (($v_2(49)$·-3.5) , (Div ((($u_2(73)$·x_{t-4})) , ($v_2(50)$·8.0)))) , ($u_2(74)$))))) , ($u_2(75)$))) , ($u_2(76)$))))))))

3. Rekursionsstufe

$v_3 = \{$1.0, 1.0, 1.0, 1.0, 1.0, 1.00000001490116, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0$\}$

$u_3 = \{$1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0$\}$

$\hat{f}_{\mathrm{sub}_3}$ = Add ((Add ((Xlog ((Sub ((Sub ((Xcos ((Sub (($(u_3(1)\cdot x_{t-1})$) , $(v_3(1)\cdot -5.0)$)) , $(u_3(2))$))) , (Sub ((Xcos ((Add (($(u_3(3)\cdot x_{t-1})$) , $(v_3(2)\cdot -5.0)$)) , $(u_3(4))$))) , (Mul ((Xexp ($(v_3(3)\cdot -6.0)$, $(u_3(5))$))) , (Xlog ($(v_3(4)\cdot 6.0)$, $(u_3(6))$))))))))) , (Xsin ($(v_3(5)\cdot 6.0)$, $(u_3(7))$))))) , $(u_3(8))$))) , (Add ((Add ((Xlog ((Sub ((Xcos ((Add (($(u_3(9)\cdot x_{t-1})$) , $(v_3(6)\cdot -5.0)$)) , $(u_3(10))$))) , (Mul ((Xexp ($(v_3(7)\cdot -6.0)$, $(u_3(11))$))) , (Xexp ((Mul ((Xsin ($(v_3(8)\cdot 6.0)$, $(u_3(12))$))) , (Mul ((Sub ((Xcos ((Sub (($(u_3(13)\cdot x_{t-1})$) , $(v_3(9)\cdot -5.0)$)) , $(u_3(14))$))) , (Mul ((Xlog ($(v_3(10)\cdot 3.0)$, $(u_3(15))$))) , (Xsin ($(v_3(11)\cdot 6.0)$, $(u_3(16))$))))))) , (Add (($(u_3(17)\cdot x_{t-1})$) , $(v_3(12)\cdot -5.0)$)))))) , $(u_3(18))$))))))) , $(u_3(19))$))) , (Sub ((Xlog ((Sub ((Xcos ((Xlog ((Sub ((Xcos ((Mul ((Add (($(u_3(20)\cdot x_{t-1})$) , $(v_3(13)\cdot -5.0)$)) , (Xsin ($(v_3(14)\cdot 6.0)$, $(u_3(21))$))))) , $(u_3(22))$))) , (Sub ((Mul ((Xlog ($(v_3(15)\cdot 3.0)$, $(u_3(23))$))) , (Xsin ($(v_3(16)\cdot 6.0)$, $(u_3(24))$))))) , (Mul ((Add (($(u_3(25)\cdot x_{t-1})$) , $(v_3(17)\cdot -5.0)$)) , (Xsin ($(v_3(18)\cdot 6.0)$, $(u_3(26))$))))))))) , $(u_3(27))$))) , $(u_3(28))$))) , (Mul ((Xcos ((Xlog ((Sub ((Xcos ((Sub (($(u_3(29)\cdot x_{t-1})$) , $(v_3(19)\cdot -5.0)$)) , $(u_3(30))$))) , (Mul ((Xlog ($(v_3(20)\cdot 3.0)$, $(u_3(31))$))) , (Xsin ($(v_3(21)\cdot 6.0)$, $(u_3(32))$))))))) , $(u_3(33))$))) , $(u_3(34))$))) , (Xsin ($(v_3(22)\cdot 6.0)$, $(u_3(35))$))))))) , $(u_3(36))$))) , (Xcos ((Sub (($(u_3(37)\cdot x_{t-1})$) , $(v_3(23)\cdot -5.0)$)) , $(u_3(38))$))))))) , (Sub ((Xcos ((Xlog ((Sub ((Sub ((Xcos ((Xlog ((Sub ((Xcos ((Xsin ($(v_3(24)\cdot 6.0)$, $(u_3(39))$))) , $(u_3(40))$))) , (Mul ((Mul ((Add (($(u_3(41)\cdot x_{t-1})$) , $(v_3(25)\cdot -5.0)$)) , (Xsin ($(v_3(26)\cdot 6.0)$, $(u_3(42))$))))) , (Xsin ($(v_3(27)\cdot 6.0)$, $(u_3(43))$))))))) , $(u_3(44))$))) , $(u_3(45))$))) , (Mul ((Xlog ($(v_3(28)\cdot 3.0)$, $(u_3(46))$))) , (Xsin ($(v_3(29)\cdot 6.0)$, $(u_3(47))$))))))) , (Xsin ($(v_3(30)\cdot 6.0)$, $(u_3(48))$))))) , $(u_3(49))$))) , $(u_3(50))$))) , (Xsin ($(v_3(31)\cdot 6.0)$, $(u_3(51))$))))))))) , (Sub ((Sub ((Xlog ((Sub ((Xcos ((Xlog ((Sub ((Xcos ((Sub (($(u_3(52)\cdot x_{t-1})$) , $(v_3(32)\cdot -5.0)$)) , $(u_3(53))$))) , (Sub ((Sub ((Mul ((Xlog ($(v_3(33)\cdot 3.0)$, $(u_3(54))$))) , (Xsin ($(v_3(34)\cdot 6.0)$, $(u_3(55))$))))) , (Mul ((Add (($(u_3(56)\cdot x_{t-1})$) , $(v_3(35)\cdot -5.0)$)) , (Xsin ($(v_3(36)\cdot 6.0)$, $(u_3(57))$))))))) , (Mul ((Add (($(u_3(58)\cdot x_{t-1})$) , $(v_3(37)\cdot -5.0)$)) , (Xsin ($(v_3(38)\cdot 6.0)$, $(u_3(59))$))))))))) , $(u_3(60))$))) , $(u_3(61))$))) , (Xsin ($(v_3(39)\cdot 6.0)$, $(u_3(62))$))))) , $(u_3(63))$))) , (Xcos ((Sub ((Xexp ((Sub (($(u_3(64)\cdot x_{t-1})$) , $(v_3(40)\cdot -5.0)$)) , $(u_3(65))$))) , (Div ((Sub ((Xcos ((Sub (($(u_3(66)\cdot x_{t-1})$) , $(v_3(41)\cdot -5.0)$)) , $(u_3(67))$))) , (Mul ((Xlog ($(v_3(42)\cdot 3.0)$, $(u_3(68))$))) , (Xsin ($(v_3(43)\cdot 6.0)$, $(u_3(69))$))))))) , (Xcos ((Sub (($(u_3(70)\cdot x_{t-1})$) , $(v_3(44)\cdot -5.0)$)) , $(u_3(71))$))))))) , $(u_3(72))$))))) , (Xsin ($(v_3(45)\cdot 6.0)$, $(u_3(73))$))))))

4. Rekursionsstufe

v_4 = {0.99999998789599, 0.99999977518993, 0.99999977518993, 0.99999975750456, 1.00180120469587, 1.00008762996818, 1.00005595943645, 0.99999252572949, 1.00034893722543, 0.99991507478099, 1.00006705070728, 0.99987387865056, 1.00000000564634, 0.99999925968514, 1.00491828410373, 0.99997196664542, 0.99999999942898, 1.00001194172303, 1.00000000441266, 1.00001393103185, 1.00074759690047, 1.00000000441266, 1.00001393103181, 1.00074759690035, 1.00122247507093, 0.99980984537948, 0.99980984537857, 0.99974899749181, 0.99882576229466, 1.00000001204218, 1.00000000859480, 1.00001546665950, 1.00233841521799}

u_4 = {6.37698947658787, -3.7099364757605, -7.1299887489297, 2.62015195145800, -1.9598058678632, 8.22503872940791, 2.55286992039664, 3.67084966596579, 4.58223223139043, 4.32705582551846, 2.99405744301743, 7.13832140738793, -1.8812149653334, -4.5292786748133, -0.8983106098085, -6.0460522602869, -4.7975628445023, -7.5671164460172, 4.18023896765384, -3.4870501954035, -9.6475851055299, 7.26668632245031, 10.0000000000000, 6.16730553543461, 6.98239826580822, -9.1189186464466, -4.4530392371600, -5.1730738002586, 1.63132296363356, -10.000000000000, 4.26895975860448, 5.54266630005924, 2.67123964897369, -8.8359780730261, 0.46166075923463, -2.4721660554733, 8.61549581307597, -2.7265696913759, -10.000000000000, -3.6215403002873, -9.4983461020357, 10.000000000000, -0.5443870036212, -0.9005783674497, 6.91618958969383, -0.5904907593788, 4.62190633496551, -4.7209043496581, -7.5502129690657, 8.55226693065693, 4.82114731825106, -10.000000000000, 3.23451398441008, 0.86286625431341, 0.64992636470026, -4.4419214682507, -0.0851930420683, -4.2173877387221, -8.8631371439562, -2.7465016894201, -8.4372833984010, 2.66121677658299, -10.000000000000, 3.04155289100017, 8.13464109553862, -1.9732356423380, 8.88278131675640, -7.3832505032091, 9.05240029015084, 6.05003712715443, 7.53731041023295, -5.6852219179246, -9.9565533136607, 5.05560295299724, 0.64144306862378, -6.1186132490685, 3.05158845709120, -5.8629679898100, -4.2053165180877, 1.64792842216370, -1.4878824370289, 0.57419270681461, -0.6622493634491, -9.5088072588756, 5.39381559564589, 5.76348153311161, -4.7475715979582, -2.1407888475456, 10.0000000000000, -7.8876003403849, -5.0167933787246, -4.6912896746320, 4.41192823539781, 1.79975373533153, -6.7764791279149, 1.49607402098665, -6.5558099459291, 9.97060245072671, 0.59788652714931}

$\hat{f}_{\mathrm{sub}_4}$ = Add ((Add ((Xcos ((Add ((Div ((Xlog ((Sub (($(u_4(1)\cdot x_{t-8})$) , $(v_4(1)\cdot 8.5)$)) , $(u_4(2)$))) , (Add ((Xcos ((Div (($(u_4(3)\cdot x_{t-8})$) , $(v_4(2)\cdot -9.0)$)) , $(u_4(4)$))) , (Add ((Xcos ((Div (($(u_4(5)\cdot x_{t-8})$) , $(v_4(3)\cdot -9.0)$)) , $(u_4(6)$))) , (Xsin ((Xsin ((Div (($(u_4(7)\cdot x_{t-8})$) , $(v_4(4)\cdot -9.0)$

)) , (u_4(8)))) , (u_4(9)))))))))) , (Xsin ((Xexp ((Sub (((u_4(10)·x_{t-8})) , (v_4(5)·8.5))) , (u_4(11)))) , (u_4(12)))))) , (u_4(13)))) , (Add ((Add ((Add ((Xcos ((Add ((Xcos ((Mul (((u_4(14)·x_{t-8})) , (v_4(6)·-9.0))) , (u_4(15)))) , (Add ((Xcos ((Add (((u_4(16)·x_{t-8})) , (v_4(7)·-9.0))) , (u_4(17)))) , (Xsin ((Sub (((u_4(18)·x_{t-8})) , (v_4(8)·8.5))) , (u_4(19)))))))) , (u_4(20)))) , (Xsin ((Add ((Add ((Xcos ((Add ((Xcos ((Mul (((u_4(21)·x_{t-8})) , (v_4(9)·-9.0))) , (u_4(22)))) , (Add ((Sub (((u_4(23)·x_{t-8})) , (v_4(10)·8.5))) , (Xsin ((Sub (((u_4(24)·x_{t-8})) , (v_4(11)·8.5))) , (u_4(25)))))))) , (u_4(26)))) , (Div (((u_4(27)·x_{t-8})) , (v_4(12)·-9.0))))) , (Xsin ((Xcos ((Add ((Div ((Xlog ((Sub (((u_4(28)·x_{t-8})) , (v_4(13)·8.5))) , (u_4(29)))) , (Xcos ((Xlog ((Xsin ((v_4(14)·-0.5) , (u_4(30)))) , (u_4(31)))) , (u_4(32)))))) , (Xsin ((Xexp ((Sub (((u_4(33)·x_{t-8})) , (v_4(15)·8.5))) , (u_4(34)))) , (u_4(35)))))) , (u_4(36)))) , (u_4(37)))))) , (u_4(38)))))) , (Add ((Add ((Xcos ((Add ((Xcos ((Mul (((u_4(39)·x_{t-8})) , (v_4(16)·-9.0))) , (u_4(40)))) , (Div ((Xlog ((Sub (((u_4(41)·x_{t-8})) , (v_4(17)·8.5))) , (u_4(42)))) , (Xcos ((Xlog ((Xsin ((v_4(18)·-0.5) , (u_4(43)))) , (u_4(44)))) , (u_4(45)))))))) , (u_4(46)))) , (Xsin ((Xcos ((Add ((Div ((Xlog ((Sub (((u_4(47)·x_{t-8})) , (v_4(19)·8.5))) , (u_4(48)))) , (Xcos ((Xlog ((Xsin ((v_4(20)·-0.5) , (u_4(49)))) , (u_4(50)))) , (u_4(51)))))) , (Xsin ((Xexp ((Sub (((u_4(52)·x_{t-8})) , (v_4(21)·8.5))) , (u_4(53)))) , (u_4(54)))))) , (u_4(55)))) , (u_4(56)))))) , (Xsin ((Xcos ((Add ((Div ((Xlog ((Sub (((u_4(57)·x_{t-8})) , (v_4(22)·8.5))) , (u_4(58)))) , (Xcos ((Xlog ((Xsin ((v_4(23)·-0.5) , (u_4(59)))) , (u_4(60)))) , (u_4(61)))))) , (Xsin ((Xexp ((Sub (((u_4(62)·x_{t-8})) , (v_4(24)·8.5))) , (u_4(63)))) , (u_4(64)))))) , (u_4(65)))) , (u_4(66)))))))) , (Xsin ((Xsin ((Add ((Xcos ((Xsin ((Xcos ((Xsin ((Sub (((u_4(67)·x_{t-8})) , (v_4(25)·8.5))) , (u_4(68)))) , (u_4(69)))) , (u_4(70)))) , (u_4(71)))) , (Add ((Add ((Xcos ((Div (((u_4(72)·x_{t-8})) , (v_4(26)·-9.0))) , (u_4(73)))) , (Add ((Xcos ((Div (((u_4(74)·x_{t-8})) , (v_4(27)·-9.0))) , (u_4(75)))) , (Xsin ((Xcos ((Mul (((u_4(76)·x_{t-8})) , (v_4(28)·-9.0))) , (u_4(77)))) , (u_4(78)))))))) , (Xsin ((Xexp ((Sub (((u_4(79)·x_{t-8})) , (v_4(29)·8.5))) , (u_4(80)))) , (u_4(81)))))))) , (u_4(82)))) , (u_4(83)))))))) , (Xsin ((Xsin ((Add ((Xcos ((Xlog ((Sub (((u_4(84)·x_{t-8})) , (v_4(30)·8.5))) , (u_4(85)))) , (u_4(86)))) , (Xsin ((Xsin ((Xcos ((Add ((Div ((Xlog ((Sub (((u_4(87)·x_{t-8})) , (v_4(31)·8.5))) , (u_4(88)))) , (Xcos ((Xlog ((Xsin ((v_4(32)·-0.5) , (u_4(89)))) , (u_4(90)))) , (u_4(91)))))) , (Xsin ((Xexp ((Sub (((u_4(92)·x_{t-8})) , (v_4(33)·8.5))) , (u_4(93)))) , (u_4(94)))))) , (u_4(95)))) , (u_4(96)))) , (u_4(97)))))) , (u_4(98)))) , (u_4(99)))))

5. Rekursionsstufe

v_5 = {-0.3430005947916, 2.03602157354683, 2.58545043840065, 6.24743883884601, 8.95787985541251, 2.84686299117860, -5.8010330164780, -1.8384343923496, 2.87736827947344, -8.8542513506124, -3.4541744562685, -5.5294898092652, 0.88966282758378, 9.53826199631463, -0.9842501048845, -8.2107281035111, -7.5269935897144}

u_5 = {-7.5296130871936, -7.0650675922424, -9.5932160680759, -0.1578434157133, -6.1623566616104, 0.74854986278193, 2.83368587592770, -0.8653032607290, 1.32166374874688, -1.0229819464766, -2.0747399119541, 10.0000000000000,

4.97750979358148, 7.36513236188100, 4.06065719089948, 3.15080609421839}

$\hat{f}_{\text{sub}_5}$ = Add ((Mul ((Xcos ((Mul ((Mul ((Mul (($v_5(1)$·5.0) , (Div ((($u_5(1)$·x_{t-1})) , (($v_5(2)$·6.5)))))) , (Add ((v_5(3)·-5.0) , (Xexp ((($u_5(4)$·x_{t-2})) , (u_5(5)))))))) , (Mul ((Add ((Xcos ((($v_5(4)$·-8.0)) , (u_5(6)))) , (v_5(5)·-2.5))) , (Div ((Xsin ((($v_5(6)$·5.5)) , (u_5(7)))) , (v_5(7)·0.5))))))) , (u_5(8)))) , (Mul ((Add ((Xcos ((($v_5(8)$·-8.0)) , (u_5(8)))) , (v_5(9)·-2.5))) , (Div ((Xsin ((($v_5(10)$·5.5)) , (u_5(10)))) , (v_5(11)·0.5))))))) , (Xcos ((Xcos ((Add ((Sub ((Mul ((Mul ((($u_5(11)$·x_{t-8})) , (($v_5(12)$·1.5)))) , (Mul ((($v_5(13)$·10.0)) , (($u_5(12)$·x_{t-4})))))) , (Add (($v_5(14)$·-5.0) , (Xexp ((($v_5(15)$·-4.0)) , (u_5(13))))))))) , (Xexp ((Sub (($v_5(16)$·(10/9.5)) , ($v_5(17)$·10.5))) , (u_5(14)))))) , (u_5(15)))) , (u_5(16)))))

B.4.2 Lernen mit den gefilterten Trainingsdaten

$a_0 = \{1.07727324655168\}$

$a_1 = \{1.00893834657706\}$

$a_2 = \{0.97979574945797\}$

$$\hat{f}_{\text{sum}_n} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2})$$

1. Rekursionsstufe

v_1 = {2.30766158010548, -1.1646660088578, -6.0514209365529, -2.3705129541858, -0.3158597478346, -5.7534601591981, -3.7092503011697, 0.98286744648103, -8.7885808933856, -9.2931215306734, -10.000000000000, -2.0783958889639, 7.73857249278220, -8.9592976095439, -8.1253954050211, 5.94955710537757, -10.000000000000, -4.2876054883862, 5.33703161651968, -4.5361903616941, -4.4275180596064, -8.3503430259878, 0.53582393919567, 2.10387094413300}

u_1 = {2.53982761009740, 0.66812309686960, -0.9463330501592, -0.0124203410077, 0.31683404960885, -8.9344314047824, -9.9999999999805, 9.73244658124166, -5.8083773463702, 0.06215475006789, 0.07705491371897, -4.0879593001345, 0.82132454284417, 9.98344028386029, 0.77762972142234, 2.20630693198091, 0.65990475313382, 0.82150781735045, 2.76316140391154, -0.7749713363307, -4.3944540564576, -3.5188041021548, -9.9999371098004, 1.23295561237161}

$\hat{f}_{\text{sub}_1}$ = Sub ((($u_1(1)\cdot x_{t-1}$)) , (Sub ((Add ((($u_1(2)\cdot x_{t-2}$)) , (($v_1(1)$·-1.5)))) , (Sub ((($u_1(3)\cdot x_{t-1}$)) , (Xlog ((Mul ((Mul ((Div ((($u_1(4)\cdot x_{t-1}$)) , (Mul ((($u_1(5)\cdot x_{t-4}$)) , (Div ((Sub ((($u_1(6)\cdot x_{t-1}$)) , (Div ((Sub ((($u_1(7)\cdot x_{t-1}$)) , (Add ((($u_1(8)\cdot x_{t-4}$)) , (Div ((Mul ((Add ((($v_1(2)\cdot 10.0$)) , (($v_1(3)$·-9.5)))) , (Add ((($u_1(9)\cdot x_{t-2}$)) , (($v_1(4)$·-1.5)))))) , (Add ((($u_1(10)\cdot x_{t-2}$)) , (($v_1(5)$·-1.5))))))))) , (Add ((($v_1(6)\cdot 6.0$)) , (($v_1(7)\cdot 4.0$)))))))) , (Add ((($u_1(11)\cdot x_{t-2}$)) , (($v_1(8)$·-1.5))))))))) , (Xexp ((Div ((Sub ((($u_1(12)\cdot x_{t-1}$)) , (Add ((($u_1(13)\cdot x_{t-4}$)) , (Div ((Mul ((Add ((($v_1(9)\cdot 10.0$)) , (($v_1(10)$·-9.5)))) , (Add ((($u_1(14)\cdot x_{t-2}$)) , (($v_1(11)$·-1.5)))))) , (Xsin ((($v_1(12)$·-8.0)) , ($u_1(15)$))))))))) , (Add ((($v_1(13)\cdot 6.0$)) , (($v_1(14)\cdot 4.0$)))))) , ($u_1(16)$))))) , (Xexp ((Div ((Sub ((($u_1(17)\cdot x_{t-1}$)) , (Add ((($u_1(18)\cdot x_{t-4}$)) , (Div ((Mul ((Add ((($v_1(15)\cdot 10.0$)) , (($v_1(16)$·-9.5)))) , (Add ((($u_1(19)\cdot x_{t-2}$)) , (($v_1(17)$·-1.5)))))) , (Sub ((($u_1(20)\cdot x_{t-1}$)) , (Sub ((($u_1(21)\cdot x_{t-4}$)) , (Div ((Mul ((Add ((($v_1(18)\cdot 10.0$)) , (($v_1(19)$·-9.5)))) , (Add ((($v_1(20)\cdot 6.0$)) , (($v_1(21)\cdot 4.0$)))))) , (Xsin ((($v_1(22)$·-8.0)) , ($u_1(22)$))))))))))))))) , (Add ((($v_1(23)\cdot 6.0$)) , (($v_1(24)\cdot 4.0$)))))) , ($u_1(23)$))))) , ($u_1(24)$))))))))

2. Rekursionsstufe

v_2 = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.00000001490116, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0}

u_2 = {-4.5887944374249, -9.7531675984699, 4.54440432564415, -3.9934248332591, -8.9968587764721, 1.95512674210417, 4.24241532449856, -2.3509406806777, -4.6299221765764, 5.31679037653083, 0.29636742182464, 0.89648428612279, 3.73434811813993, -1.3499203451096, -8.7347942794417, 8.30360589407968, 1.52567675302215, 2.59181588827248, -3.3608708629561, 4.95662476410225, 5.34246494907765, 10.0000000000000, -7.9660465633773, 9.76929848805559, 3.98749936596338, -4.2539055425190, -5.9416758079469, 10.0000000000000, 10.0000000000000, -2.9549775013329, -0.6270475539469, -8.2108315537118, 4.08854181061630, -0.8558970541742, -9.2810106221376, 8.16478959333641, -6.9629490440608, -7.3357990262523, -5.2186707905400, 8.79106787867287, 8.51217339125349, -0.9729386144975, 5.66875510750841, -0.9970736817987, 2.71545064614806, -9.1662992932563, -7.5887209307309, 4.17640907884968, 3.18756410906669, 0.60070014409398, -8.7709268550219, 6.36920961237808, 2.06429538915439, -4.5573647679291, -9.7907088668027, -10.000000000000, -6.2857122718428, 10.0000000000000, -9.0782546422691, -2.7193691658406, -5.1528766550975, -3.3131845215100, -10.000000000000, 5.27434676860626, -7.0663456326212, 1.12145364359789, -5.6200860009681, 2.15765696076500, -3.0629934064493, 4.64140239690645, -9.9127287969782, 4.58926338345898, -0.0495884726617, 6.37753460538947, -7.0250996156184, -7.4475679558883, 9.27497064750845, 1.97365047492014, 8.19308737661311, -4.0449506547888, 0.49363165488699, 5.42505866479746, -2.7965272742324, -2.4450087007100,

-7.0480233366372, -10.000000000000, 10.000000000000, 2.06740542737733, 0.24177656008045, -9.6454933289434, -9.5126888472843, -1.1683458538632}

$\hat{f}_{\text{sub}_2}$ = Sub ((Xsin ((Div ((Xsin ((Div ((Xsin ((Div ((Xsin ((Add ((Xsin ((($u_2(1)$·x_{t-1})) , ($u_2(2)$))) , (Xlog ((Xcos ((Xcos ((Xcos ((($v_2(1)$·4.5)) , ($u_2(3)$))) , ($u_2(4)$))) , ($u_2(5)$))) , ($u_2(6)$))))) , ($u_2(7)$))) , (Div ((Xsin ((Xlog ((Xcos ((Mul ((Mul ((($v_2(2)$·6.0)) , (($v_2(3)$·7.0)))) , (Xsin ((($u_2(8)$·x_{t-1})) , ($u_2(9)$))))) , ($u_2(10)$))) , ($u_2(11)$))) , ($u_2(12)$))) , (Add ((Xcos ((Xcos ((Mul ((($v_2(4)$·-4.0)) , (Add ((($v_2(5)$·-6.0)) , (($v_2(6)$·8.5)))))) , ($u_2(13)$))) , ($u_2(14)$))) , (($v_2(7)$·-7.0))))))) , ($u_2(15)$))) , (Sub ((($v_2(8)$·4.0)) , (Add ((Div ((Div ((($v_2(9)$·2.5)) , (($v_2(10)$·-4.5)))) , (Add ((Xcos ((Xcos ((Mul ((($v_2(11)$·-4.0)) , (Xcos ((Mul ((Xsin ((($u_2(16)$·x_{t-1})) , ($u_2(17)$))) , (Xsin ((($u_2(18)$·x_{t-1})) , ($u_2(19)$))))) , ($u_2(20)$))))) , ($u_2(21)$))) , ($u_2(22)$))) , (($v_2(12)$·-7.0)))))) , (($v_2(13)$·5.0)))))))) , ($u_2(23)$))) , (Sub ((Sub ((Xexp ((Xcos ((Xexp ((($u_2(24)$·x_{t-10})) , ($u_2(25)$))) , ($u_2(26)$))) , ($u_2(27)$))) , (Xcos ((Mul ((Mul ((($v_2(14)$·6.0)) , (($v_2(15)$·7.0)))) , (Xsin ((($u_2(28)$·x_{t-1})) , ($u_2(29)$))))) , ($u_2(30)$))))) , (Xlog ((Div ((Xsin ((Div ((Xsin ((Add ((Add ((Add ((Sub ((Sub ((($v_2(16)$·4.0)) , (($v_2(17)$·-2.5)))) , (Mul ((($v_2(18)$·-8.5)) , (($v_2(19)$·-9.0)))))) , (Sub ((Xcos ((($v_2(20)$·8.0)) , ($u_2(31)$))) , (Sub ((($u_2(32)$·x_{t-12})) , ((0.0))))))))) , (Mul ((Xsin ((Xexp ((($v_2(21)$·2.0)) , ($u_2(33)$))) , ($u_2(34)$))) , (Xlog ((Xexp ((($v_2(22)$·5.5)) , ($u_2(35)$))) , ($u_2(36)$))))))) , (Xlog ((Xcos ((Xcos ((Xcos ((Mul ((Mul ((($v_2(23)$·6.0)) , (($v_2(24)$·7.0)))) , (Xsin ((($u_2(37)$·x_{t-1})) , ($u_2(38)$))))) , ($u_2(39)$))) , ($u_2(40)$))) , ($u_2(41)$))) , ($u_2(42)$))))) , ($u_2(43)$))) , (Xexp ((Xcos ((Xexp ((($u_2(44)$·x_{t-10})) , ($u_2(45)$))) , ($u_2(46)$))) , ($u_2(47)$))))) , ($u_2(48)$))) , (Sub ((($v_2(25)$·4.0)) , (Add ((Div ((Xcos ((Xcos ((Xcos ((($v_2(26)$·4.5)) , ($u_2(49)$))) , ($u_2(50)$))) , ($u_2(51)$))) , (Add ((Xcos ((Xcos ((Xcos ((Xcos ((Mul ((($v_2(27)$·6.0)) , (Xsin ((($u_2(52)$·x_{t-1})) , ($u_2(53)$))))) , ($u_2(54)$))) , ($u_2(55)$))) , ($u_2(56)$))) , ($u_2(57)$))) , (($v_2(28)$·-7.0)))))) , (($v_2(29)$·5.0)))))))) , ($u_2(58)$))))))) , ($u_2(59)$))) , (Sub ((Sub ((Xexp ((Xcos ((Xexp ((($u_2(60)$·x_{t-10})) , ($u_2(61)$))) , ($u_2(62)$))) , ($u_2(63)$))) , (Xcos ((Mul ((Mul ((($v_2(30)$·6.0)) , (($v_2(31)$·7.0)))) , (Xsin ((($u_2(64)$·x_{t-1})) , ($u_2(65)$))))) , ($u_2(66)$))))) , (Xlog ((Xsin ((Div ((Xsin ((Div ((Xsin ((Add ((Add ((Add ((Sub ((Sub ((($v_2(32)$·4.0)) , (($v_2(33)$·-2.5)))) , (Mul ((($v_2(34)$·-8.5)) , (($v_2(35)$·-9.0)))))) , (Sub ((Xcos ((($v_2(36)$·8.0)) , ($u_2(67)$))) , (Sub ((($u_2(68)$·x_{t-12})) , ((0.0)))))))) , (Mul ((Xsin ((Xexp ((($v_2(37)$·2.0)) , ($u_2(69)$))) , ($u_2(70)$))) , (Xlog ((Xexp ((($v_2(38)$·5.5)) , ($u_2(71)$))) , ($u_2(72)$))))))) , (Xlog ((Xcos ((Xcos ((Xcos ((Mul ((Mul ((($v_2(39)$·6.0)) , (($v_2(40)$·7.0)))) , (Xsin ((($u_2(73)$·x_{t-1})) , ($u_2(74)$))))) , ($u_2(75)$))) , ($u_2(76)$))) , ($u_2(77)$))) , ($u_2(78)$))))) , ($u_2(79)$))) , (Xcos ((Div ((Add ((($v_2(41)$·6.0)) , (($v_2(42)$·7.0)))) , (Div ((Xsin ((($u_2(80)$·x_{t-1})) , ($u_2(81)$))) , (($v_2(43)$·-6.5)))))) , ($u_2(82)$))))) , ($u_2(83)$))) , (Sub ((($v_2(44)$·4.0)) , (Add ((Div ((Xsin ((Xlog ((Xcos ((Mul ((Mul ((($v_2(45)$·6.0)) , (($v_2(46)$·7.0)))) , (Xcos ((($v_2(47)$·8.0)) , ($u_2(84)$)))))) , ($u_2(85)$))) , ($u_2(86)$))) , ($u_2(87)$))) , (Add ((Xcos ((Xexp ((($u_2(88)$·x_{t-10})) , ($u_2(89)$))) , ($u_2(90)$))) , (($v_2(48)$·-7.0)))))) , (($v_2(49)$·5.0)))))))) , ($u_2(91)$))) , ($u_2(92)$))))))

B.5 Wechselkurs zwischen Euro und Baht

B.5.1 Lernen mit den originalen Trainingsdaten

$a_0 = \{1.30604510814502\}$

$a_1 = \{0.96646858467516\}$

$a_2 = \{0.98524464536535\}$

$a_3 = \{1.22624554661084\}$

$a_4 = \{1.0\}$

$a_5 = \{1.0\}$

$$\hat{f}_{\text{sum}_n} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) + ... + (a_5 \cdot \hat{f}_{sub_5})$$

1. Rekursionsstufe

$u_1 = \{$-1.96507508573071, -1.96319456238003, -0.57804033823713, 1.06003833213003, 0.99921648382348$\}$

$\hat{f}_{\text{sub}_1}$ = Add ((Xsin ((Xsin ((Sub ((($u_1(1)$·x_{t-1})) , (($u_1(2)$·x_{t-2})))) , ($u_1(3)$))) , ($u_1(4)$))) , (($u_1(5)$·x_{t-2})))

2. Rekursionsstufe

$v_2 = \{$1.84574427494441, -0.5729915597166, -1.2780601767133, 1.00759579362657, -1.2048774971967, -1.5918591983156, 0.32940385120571, -0.0248670207901, 2.85169016772381, -0.4094223213248, -0.3131185523668$\}$

$u_2 = \{$1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.00000001490116, 1.0, 1.0$\}$

$\hat{f}_{\text{sub}_2}$ = Mul ((Xsin ((Xcos ((Add ((Xexp ((Xexp ((Div ((($v_2(1)$·-1.0)) , (($v_2(2)$·2.0)))) , ($u_2(1)$))) , ($u_2(2)$))) , (Div ((Xsin ((Add ((Div ((($v_2(3)$·-1.0)) , (($v_2(4)$·2.0)))) , (Xexp ((($v_2(5)$·-3.0)) , ($u_2(3)$)))))) , ($u_2(4)$))) , (Xexp ((Xcos ((Div ((($v_2(6)$·2.5)) , (($v_2(7)$·-10.0)))) , ($u_2(5)$))) , ($u_2(6)$))))))) , ($u_2(7)$))) , ($u_2(8)$))) , (Xsin ((Div ((Add ((($u_2(9)$·x_{t-20})) , (($v_2(8)$·6.0)))) , (Xexp ((($v_2(9)$·-3.0)) , ($u_2(10)$))))) , ($u_2(11)$))))

3. Rekursionsstufe

v_3 = {0.76864396791004, 5.99051997888494, 9.72639890876945, -8.0915110491606, -0.1658217955084, -9.1300569122680, 9.99999902948681, 6.45704173012868, 2.49554504637781, 3.92552795623577, 9.99999999999996, -9.4893725993735, -9.9999999999645, -9.5168063613571, 5.43291863396495, -1.1194921433871, -9.9999999999841, -5.6180907414214, -9.1476001070109, -8.1984915180003, -5.5252229151796, 1.00279233516262, -2.3406060682203, -0.8079178846087, -7.5561932638676, -0.8561042755422, 9.10430357866637, 8.94840744078938, 0.19660909604439, 1.02205442485750, 9.51583106888115, 4.53929387025707, -9.9999999999999, 9.99999999999836, 5.77842596760039, -9.9999999999999, 2.22650336183910, -7.1911984363422, 9.56753456366317, 3.22858993764042, -6.5875139393011, -2.1560938151446, 9.95139870459944, 1.52948583947030, 9.66173572731235}

u_3 = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0}

$\hat{f}_{\mathrm{sub}_3}$ = Div ((Sub ((Xsin ((Xexp ((Xexp ((Sub (((v_3(1)·-1.0)) , ((0.0)))) , (u_3(1)))) , (u_3(2)))) , (u_3(3)))) , (Xexp ((Sub ((Xcos ((Div ((Mul ((Div (((v_3(2)·6.0)) , ((v_3(3)·7.5)))) , (Add (((u_3(4)·x_{t-12})) , ((v_3(4)·7.5)))))) , (Xlog ((Sub ((Xsin ((Xsin (((v_3(5)·-7.0)) , (u_3(5)))) , (u_3(6)))) , (Xsin ((Xlog (((v_3(6)·4.0)) , (u_3(7)))) , (u_3(8)))))) , (u_3(9))))) , (u_3(10)))) , (Add ((Xsin ((Div ((Xsin (((v_3(7)·-10.0)) , (u_3(11)))) , (Div ((Xlog ((Mul ((Xlog ((Div ((Div ((Sub ((Xsin (((v_3(8)·-7.0)) , (u_3(12)))) , (Mul (((v_3(9)·-7.0)) , ((v_3(10)·-9.5)))))) , (Xsin (((v_3(11)·-7.0)) , (u_3(13)))))) , (Mul ((Xsin ((Mul (((v_3(12)·3.0)) , ((u_3(14)·x_{t-2})))) , (u_3(15)))) , (Sub ((Div (((v_3(13)·-8.5)) , ((v_3(14)·4.0)))) , (Add ((Sub (((v_3(15)·-9.5)) , ((v_3(16)·7.5)))) , (Xlog (((v_3(17)·-9.5)) , (u_3(16))))))))))) , (u_3(17)))) , (Sub ((Div (((v_3(18)·-8.5)) , ((v_3(19)·4.0)))) , (Sub ((Div ((Add (((v_3(20)·6.5)) , ((v_3(21)·9.0)))) , (Add (((u_3(18)·x_{t-10})) , ((v_3(22)·-7.0)))))) , (Xsin (((v_3(23)·-10.0)) , (u_3(19)))))))))) , (u_3(20)))) , (Mul ((Xsin ((Mul (((v_3(24)·8.0)) , ((u_3(21)·x_{t-18})))) , (u_3(22)))) , (Sub ((Div (((v_3(25)·-8.5)) , ((v_3(26)·4.0)))) , (Add ((Sub (((v_3(27)·-9.5)) , ((v_3(28)·7.5)))) , (Xlog (((v_3(29)·-9.5)) , (u_3(23)))))))))))) , (u_3(24)))) , (Xsin ((Mul (((v_3(30)·8.0)) , ((u_3(25)·x_{t-18})))) , (u_3(26)))))))) , (u_3(27)))))) , (Add ((Add ((Mul ((Xlog ((Add ((Xsin (((v_3(31)·9.5)) , (u_3(28)))) , (Xlog ((Sub ((Xsin ((Xexp (((v_3(32)·-7.0)) , (u_3(29)))) , (u_3(30)))) , (Div ((Xcos (((v_3(33)·-4.0)) , (u_3(31)))) , (Xlog ((Xlog ((Add (((v_3(34)·-5.0)) , ((v_3(35)·-3.0)))) , (u_3(32)))) , (u_3(33)))))))) , (u_3(34)))))) , (u_3(35)))) , (Sub ((Xsin ((Xsin (((v_3(36)·-7.0)) , (u_3(36)))) , (u_3(37)))) , (Xsin ((Xexp ((Xlog (((v_3(37)·-2.0)) , (u_3(38)))) , (u_3(39)))) , (u_3(40))))))) , (Add (((u_3(41)·x_{t-6})) , (Sub (((v_3(38)·-10.0)) , ((v_3(39)·5.5)))))))) , (Sub ((Div ((Mul (((v_3(40)·6.5)) , ((v_3(41)·9.0)))) , (Add (((u_3(42)·x_{t-10})) , ((v_3(42)·-7.0))))

)) , (Mul ((Div (((v_3(43)·-2.0)) , ((v_3(44)·10.0)))) , (Xsin (((v_3(45)·-10.0)) , (u_3(43))))))))))

4. Rekursionsstufe

v_4 = {0.07748965343560, -3.9937971746014, 4.10533184633024, -4.0272934449074, 3.05955187211295, -2.2622902180589, 4.99609044348103, -4.9959670021497, -1.1157363842249, 2.50286582261069, 4.00292581906747, -2.2938569594828, 3.20822843689664, 0.87486313638309, 2.05990786616210, 3.81044357251513, 4.68271942144782, 2.72692601089766, 3.94048373554039, 2.82911881506963, -1.7928919688601, 0.28020555113809, -4.9971901931084, -2.9279143251442, -3.8112198646138, -0.1386010839000, 4.63586904144759, 0.76493502205781, 0.64882635007175, 4.03114091008439, -4.1285337189519, 4.93112095709472, -2.1751518937574, 2.90925234427109}

u_4 = {4.72829351609376, 0.03083848658670, 1.92269057918886, -4.9999999746068, -3.9684572400342, -2.2329094086965, 2.78640498899035, -4.9999999999992, -2.7201496321013, -0.0169863149583, 4.54934020095951, -2.6627684223988, 3.54564854589489, 0.80091201837841, 3.72795640375423, 3.13438233332844, -4.9999999999995, -0.8446069432972, -2.3708930488313, -2.8994430814428, -0.1127051184992, -3.8558591045555, -1.4183741691357, -3.3788728635433, 4.68866497410826, 4.99999999999970, 2.79479306229758, 4.88614368310568, -4.9999999999999, 0.42877620059923, -1.9590321845747, 1.81528258315439, -3.9487972019082, 4.99999999999998, -4.7030492715575, -1.6464379316748, -0.9444205267946, -4.9020702113939, -4.5129096722756, 2.71564959181239, , 2.19538626301907, 4.99999999999998, 1.96335225048477, 4.48079334269342, -4.9999999999999, 1.00659992603613, 3.23227310651384, 4.40023847951583, 1.82983240488326, 4.65842618569343}

$\hat{f}_{\text{sub}_4}$ = Div ((Xlog ((Xcos ((Sub ((Xcos ((Xcos ((Sub ((Xcos ((Mul ((Xlog (((v_4(1)·-1.0)) , (u_4(1)))) , (Div ((Mul (((v_4(2)·-8.0)) , ((v_4(3)·8.5)))) , (Add (((v_4(4)·-1.0)) , ((v_4(5)·-10.0)))))))) , (u_4(2)))) , (Sub ((Mul ((Div ((Div (((u_4(3)·x_{t-10})) , ((v_4(6)·-4.5)))) , (Xsin (((u_4(4)·x_{t-1})) , (u_4(5))))))) , (Xcos ((Xlog (((u_4(6)·x_{t-2})) , (u_4(7)))) , (u_4(8)))))) , (Mul (((v_4(7)·-4.5)) , ((v_4(8)·-3.5)))))))) , (u_4(9)))) , (u_4(10)))) , (Sub ((Mul ((Div ((Div (((u_4(11)·x_{t-10})) , ((v_4(9)·-4.5)))) , (Xcos (((v_4(10)·8.0)) , (u_4(12))))))) , (Xcos ((Xlog (((u_4(13)·x_{t-2})) , (u_4(14)))) , (u_4(15)))))) , (Mul (((v_4(11)·-4.5)) , ((v_4(12)·-3.5)))))))) , (u_4(16)))) , (u_4(17)))) , (Div ((Sub ((Div ((Xcos (((v_4(13)·-1.5)) , (u_4(18)))) , (Add (((v_4(14)·2.5)) , ((v_4(15)·-0.5)))))) , (Xsin ((Mul ((Sub ((Mul (((v_4(16)·-8.0)) , ((v_4(17)·8.5)))) , (Xsin (((u_4(19)·x_{t-1})) , (u_4(20))))))) , (Div ((Xlog (((u_4(21)·x_{t-2})) , (u_4(22)))) , (Xlog (((v_4(18)·-7.5)) , (u_4(23)))))))) , (u_4(24)))))) , (Div ((Sub ((Sub ((Xexp ((Add (((v_4(19)·6.0)) , ((v_4(20)·-7.5)))) , (u_4(25)))) , (Sub

((Mul ((($v_4(21)$·-8.0)) , (($v_4(22)$·8.5)))) , (Add ((Add ((($u_4(26)$·x_{t-18})) , (($v_4(23)$·-3.0)))) , (Xcos ((($u_4(27)$·x_{t-8})) , ($u_4(28)$)))))))) , (Xsin ((Add ((Add ((($u_4(29)$·x_{t-18})) , (($v_4(24)$·-3.0)))) , (Xcos ((($u_4(30)$·x_{t-8})) , ($u_4(31)$)))))) , ($u_4(32)$))))) , (Sub ((Xsin ((Add ((Add ((Div ((Xlog ((($u_4(33)$·x_{t-2})) , ($u_4(34)$))) , (Xlog ((($v_4(25)$·-7.5)) , ($u_4(35)$)))))) , (Xlog ((($v_4(26)$·2.0)) , ($u_4(36)$)))))) , (Xcos ((Sub ((Xcos ((Mul ((Xlog ((($v_4(27)$·-1.0)) , ($u_4(37)$))) , (Mul ((Xcos ((($u_4(38)$·x_{t-10})) , ($u_4(39)$))) , (($v_4(28)$·-2.0)))))) , ($u_4(40)$))) , (Sub ((Mul ((Div ((Div ((($u_4(41)$·x_{t-10})) , (($v_4(29)$·-4.5)))) , (Xcos ((($v_4(30)$·8.0)) , ($u_4(42)$)))))) , (Xcos ((Xlog ((($u_4(43)$·x_{t-2})) , ($u_4(44)$))) , ($u_4(45)$))))) , (Mul ((($v_4(31)$·-4.5)) , (($v_4(32)$·-3.5)))))))) , ($u_4(46)$)))))) , ($u_4(47)$))) , (Xexp ((Add ((Add ((($u_4(48)$·x_{t-18})) , (($v_4(33)$·-3.0)))) , (Xlog ((($v_4(34)$·2.0)) , ($u_4(49)$)))))) , ($u_4(50)$))))))))))

5. Rekursionsstufe

$v_5 = \{1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.00000001490116\}$

$u_5 = \{$-3.6704619926401, -4.1166259175897, -4.8593554556441, 5.00000001490093, -4.9996705710760, -4.9950638185126, 0.16645380132959, 1.16534809907377, 0.80106525946271, -2.3426576368809, -0.9082182041046$\}$

$\hat{f}_{\text{sub}_5}$ = Div ((Mul ((Sub ((Mul ((Xsin ((($u_5(1)$·x_{t-20})) , ($u_5(2)$))) , (Mul ((($v_5(1)$·-8.5)) , (($v_5(2)$·4.5))))))) , (Xexp ((Add ((Xsin ((Xsin ((($u_5(3)$·x_{t-20})) , ($u_5(4)$))) , ($u_5(5)$))) , (Xsin ((Xlog ((($v_5(3)$·-2.0)) , ($u_5(6)$))) , ($u_5(7)$)))))) , ($u_5(8)$))))) , (Xexp ((($v_5(4)$·-9.5)) , ($u_5(9)$))))) , (Xcos ((Add ((($u_5(10)$·x_{t-14})) , (($v_5(5)$·-0.5)))) , ($u_5(11)$))))

B.5.2 Lernen mit den gefilterten Trainingsdaten

$a_0 = \{0.00074546585596\}$

$a_1 = \{0.99997086504890\}$

$a_2 = \{1.02909450569510\}$

$a_3 = \{1.04827634895412\}$

$$\hat{f}_{\text{sum}_n} = a_0 + (a_1 \cdot \hat{f}_{\text{sub}_1}) + (a_2 \cdot \hat{f}_{\text{sub}_2}) + (a_3 \cdot \hat{f}_{\text{sub}_3})$$

1. Rekursionsstufe

v_1 = {0.84060960397770, -0.5929501008869, -2.2266778040744, -3.4335954287508, -1.2203447179501, 2.73155382980174, -0.9723835092792, -0.1067485033792, 2.42238448760756, -0.6413078938107, -0.9881627144895, -3.2934976490422, 0.22052067134778, 2.72690515702388, 0.85472622189867, 1.73329660776175, 3.56473069045426, -3.9048718419455, 0.59961197645602, -1.3805483099472, -4.0000000000000, 3.97282940882149, 0.41451115026605, -0.9155191019739, 1.70661113026722, -1.5545566583229, 2.84863742139472, -2.3345315546636, -4.0000000000000, 3.67309329546542, 2.37795959446336, 1.28704255703344, 1.27042915763396, -3.9997289993974, -4.0000000000000, -1.8983741936890, -0.2942897196693, -3.9999876263008, -1.0908328058182, -2.9550659454209, 4.00000000000000, 4.00000000000000, -3.7929210985691, -2.1368690363948, -1.1474670864705, 3.75820699933888, -4.0000000000000, -0.6415363125361, 2.49860600521124, 4.00000000000000, 2.76728708396434, -0.7780843577093}

u_1 = {1.00100064657082, 1.00198259338088, 1.00100064760186, 0.99970268920222, 1.13710196496755, 1.22865004450867, 1.00280917755179, 0.99819581701196, 0.99415664779681, 1.01174537085058, 1.00753634106769, 0.98782941205671, 1.01582615373315, 1.00890472188573, 0.98202856695684, 0.77856268483247, 0.95420235899835, 0.90351399343835, 0.99973775645400, 1.03602480474752, 1.00134332422170, 0.90266168246142, 1.06129814876085, -0.0823553194252, 0.56531348096334, 1.00054097822948, 0.97675714900236, 0.99976141646642, 1.00013152283720, 1.38371856856503, 0.99886884985215, 0.96753390885130, 1.00350628274371, 1.00302888986940, 0.97324462675366, 1.00322602121886, 0.97017715947124, 0.99738363639658, 0.99967950728643, 0.99955578041760, 0.99967958485671, 0.97324462675366, 0.52345735259537, 2.88543898840597, 1.31939191760981, 1.01847886985857, 1.11620602260714, 1.00123749819683, 0.70862260686039, 9.65440366861464, 9.65440366861464, 0.45325942217605, 0.45726676113703, 0.98117889448261, 0.82798689354572, 1.11641602342868, 1.48911768488088, 0.12947474587453, -0.6570869433580, -0.1900582265883, 1.15715398626601}

$\hat{f}_{\text{sub}_1}$ = Sub ((($u_1(1)$·x_{t-1})) , (Sub ((($u_1(2)$·x_{t-2})) , (Sub ((($u_1(3)$·x_{t-1})) , (Div ((Xlog ((Xlog ((Xlog ((Sub ((Div ((Xlog ((Div ((Xcos ((Xcos ((Xcos ((Sub ((Mul ((($u_1(4)$·x_{t-2})) , (($v_1(1)$·1.0)))) , (Add ((($v_1(2)$·6.5)) , (($v_1(3)$·5.0))))))) , ($u_1(5)$))) , ($u_1(6)$))) , ($u_1(7)$))) , (Sub ((($u_1(8)$·x_{t-1})) , (Div ((($u_1(9)$·x_{t-14})) , (($v_1(4)$·-7.5)))))))) , ($u_1(10)$))) , (Sub ((($u_1(11)$·x_{t-1})) , (Add ((($u_1(12)$·x_{t-14})) , (($v_1(5)$·-7.5)))))))) , (Xlog ((($v_1(6)$·2.5)) , ($u_1(13)$)))))) , ($u_1(14)$))) , ($u_1(15)$))) , ($u_1(16)$))) , (Sub ((Add ((($u_1(17)$·x_{t-14})) , (($v_1(7)$·-7.5)))) , (Xlog ((Div ((Div ((Sub ((Div ((Div ((Add ((($v_1(8)$·6.5)) , (($v_1(9)$·5.0)))) , (Sub ((Mul ((($u_1(18)$·x_{t-2})) , (($v_1(10)$·1.0)))) , (Add ((($v_1(11)$·6.5)) , (($v_1(12)$·5.0))))

)))) , (Xlog ((Xcos ((Sub ((Xcos ((Sub ((Mul ((($u_1(19)\cdot x_{t-2}$)) , (($v_1(13)\cdot 1.0$)))) , (Add ((($v_1(14)\cdot 6.5$)) , (($v_1(15)\cdot 5.0$)))))) , ($u_1(20)$))) , (Add ((($v_1(16)\cdot 6.5$)) , (($v_1(17)\cdot 5.0$)))))) , ($u_1(21)$))) , ($u_1(22)$))))) , (Div ((Xcos ((Sub ((Xcos ((Sub ((Mul ((($u_1(23)\cdot x_{t-2}$)) , (($v_1(18)\cdot 1.0$)))) , (Add ((($v_1(19)\cdot 6.5$)) , (($v_1(20)\cdot 5.0$)))))) , ($u_1(24)$))) , (Add ((($v_1(21)\cdot 6.5$)) , (($v_1(22)\cdot 5.0$)))))) , ($u_1(25)$))) , (Xlog ((Xcos ((Sub ((Sub ((Xcos ((Mul ((($u_1(26)\cdot x_{t-2}$)) , (($v_1(23)\cdot 1.0$)))) , ($u_1(27)$))) , (Mul ((($u_1(28)\cdot x_{t-2}$)) , (($v_1(24)\cdot 1.0$)))))) , (Add ((($v_1(25)\cdot 6.5$)) , (($v_1(26)\cdot 5.0$)))))) , ($u_1(29)$))) , ($u_1(30)$)))))))) , (Xlog ((Xcos ((Sub ((Xcos ((Mul ((($u_1(31)\cdot x_{t-2}$)) , (($v_1(27)\cdot 1.0$)))) , ($u_1(32)$))) , (Mul ((($u_1(33)\cdot x_{t-2}$)) , (($v_1(28)\cdot 1.0$)))))) , ($u_1(34)$))) , ($u_1(35)$))))) , (Div ((Xlog ((Xlog ((Xexp ((Xcos ((Sub ((Xcos ((Mul ((($u_1(36)\cdot x_{t-2}$)) , (($v_1(29)\cdot 1.0$)))) , ($u_1(37)$))) , (Mul ((($u_1(38)\cdot x_{t-2}$)) , (($v_1(30)\cdot 1.0$)))))) , ($u_1(39)$))) , ($u_1(40)$))) , ($u_1(41)$))) , ($u_1(42)$))) , (Sub ((Div ((Xcos ((Sub ((Xcos ((Sub ((Mul ((($u_1(43)\cdot x_{t-2}$)) , (($v_1(31)\cdot 1.0$)))) , (Add ((($v_1(32)\cdot 6.5$)) , (($v_1(33)\cdot 5.0$)))))) , ($u_1(44)$))) , (Add ((($v_1(34)\cdot 6.5$)) , (($v_1(35)\cdot 5.0$)))))) , ($u_1(45)$))) , (Xlog ((Xcos ((Sub ((Xcos ((Sub ((Mul ((($u_1(46)\cdot x_{t-2}$)) , (($v_1(36)\cdot 1.0$)))) , (Add ((($v_1(37)\cdot 6.5$)) , (($v_1(38)\cdot 5.0$)))))) , ($u_1(47)$))) , (Add ((($v_1(39)\cdot 6.5$)) , (($v_1(40)\cdot 5.0$)))))) , ($u_1(48)$))) , ($u_1(49)$))))) , (Sub ((Div ((Xcos ((Mul ((Xcos ((Sub ((Mul ((($u_1(50)\cdot x_{t-2}$)) , (($v_1(41)\cdot 1.0$)))) , (Mul ((($u_1(51)\cdot x_{t-2}$)) , (($v_1(42)\cdot 1.0$)))))) , ($u_1(52)$))) , (Add ((($v_1(43)\cdot 6.5$)) , (($v_1(44)\cdot 5.0$)))))) , ($u_1(53)$))) , (Xlog ((Xcos ((Sub ((Xcos ((Mul ((($u_1(54)\cdot x_{t-2}$)) , (($v_1(45)\cdot 1.0$)))) , ($u_1(55)$))) , (Add ((($v_1(46)\cdot 6.5$)) , (($v_1(47)\cdot 5.0$)))))) , ($u_1(56)$))) , ($u_1(57)$))))) , (Xcos ((Sub ((Xcos ((Sub ((Mul ((($u_1(58)\cdot x_{t-2}$)) , (($v_1(48)\cdot 1.0$)))) , (Add ((($v_1(49)\cdot 6.5$)) , (($v_1(50)\cdot 5.0$)))))) , ($u_1(59)$))) , (Add ((($v_1(51)\cdot 6.5$)) , (($v_1(52)\cdot 5.0$)))))) , ($u_1(60)$))))))))))) , ($u_1(61)$))))))))))))))

2. Rekursionsstufe

v_2 = {-0.6286528226003, -0.4598752965517, 5.00000000000000, 5.00000000000000, 1.08994799121375, -5.0000000000000, 0.51436269558011, -3.9825503946870, 1.05253446107211, -2.9138422358051, -3.6611680303809, 3.66703645908915, 1.00530160009538, 2.28733932548029, -4.1710897974118, 0.85568527646039, -0.1189754076530, 1.21695173102629, 4.99989613904175, 0.20759924400771, -4.6133046362495, 0.85077562905555, -2.5826299701791, -4.9999587231062, -0.5032350512989, 0.11704179020930, 0.00575494077662, 0.23753967162593, 0.68430124502012, 1.76057585783106, -2.7120162756495, -4.6716348028067, 3.10987638690525, -1.9863251437897, 2.25662638646093, -1.1176018936484, 4.38986618199912, 2.16089450141434, 1.08885636793124, -1.8097775415831, -4.9198608510501, -4.5600874573461, 4.92748695906277, 5.00000000000000, 4.94495829565401, -0.7101658641551, 2.97487235456398, -4.3370911380178, -0.2875186809798, -4.1753546862013, -4.4739190361324, 3.01106942117222, -1.0350794043887, 3.45825170022315}

u_2 = {0.99991813021954, 1.00009077684425, 1.00009077684425, 1.00009014767177, 0.99991806507308, 1.00009032008368, 0.99991496101030, 1.00008808502361,

1.00009052786523, 1.00100742376617, 1.00051105066016, 0.99988880990653,
0.99979772097120, 0.99991806507308, 0.99940893640478, 0.99939824133145,
0.99701106364080, 0.99991806507308, 0.99977244784852, 0.99977693488294,
0.99991806507308, 1.00009936638582, 0.99901652607682, 1.00117490959092,
1.00136108319144, 0.99844802577793, 1.00003958597942, 1.00003958067691,
1.00004391262238, 0.99908913887184, 0.99908947851523, 1.00008823219927,
0.99972204312038, 0.99975742640787, 0.99975805836202, 0.99992907465098,
0.99996779027060, 0.99992547983162, 1.00000094559774, 1.00000094502997,
1.00039235006226, 0.99997279569269, 0.99996776748735, 1.00009699414076,
0.99995500767076, 1.00003847007602, 1.00003786231112, 1.00003847007602,
1.00002541562413, 1.00003824366976, 1.00003677245301, 1.00003847007602,
1.00009276314060, 1.00006723638910, 0.99912760581978, 0.99866290807492,
0.99995198733095, 0.99995196547039, 0.99999277623187, 0.99997968127074,
0.99997196879248, 0.99996594413010, 0.99996413733830, 0.99021431188754,
0.99986990059801, 0.99986991083261, 0.99732376449972, 0.99655676604912,
1.00000000000000, 1.00000000000000, 1.00000000000000, 1.00242066649043,
0.99851396157849, 1.00076077815012, 1.00076079029687, 1.00046945743072,
1.00047143660287, 0.99957380834855, 0.99955696867862, 0.99957380834855,
1.00236559806494, 1.00240204828707, 0.99780366910736, 0.99782060390257,
0.99733164470345, 1.00231367306936, 1.00234532635704, 1.00000000000000,
1.00000000000000, 1.00000000000000, 1.00000000000000, 1.00000000000000,
1.00217754351725, 1.00234532635704, 0.99608551914379, 1.00347135150207,
1.00347135150207, 1.00351096409056, 1.00349826290456, 1.00357957258593,
0.99783715457398, 1.00183416991077, 1.00169460773815, 0.99787881819590,
1.00003371839586, 1.00012616161839, 0.99999587054941, 0.99999587054941,
0.99988184225928, 1.00003175705973, 0.99886670151250, 1.00027106951575,
1.00024562383392, 1.00377476960259, 0.99940163469520, 1.00018902142782,
1.00020309364427, 0.99982246192495, 1.00024562383392, 0.99982141658182}

$\hat{f}_{\text{sub}_2}$ = Div ((Xsin ((Add ((Xlog ((Xsin ((Sub ((Div ((Xlog ((Xsin ((Mul ((Xexp ((Xcos (((v_2(1)·-3.5)) , (u_2(1)))) , (u_2(2)))) , (Xlog (((v_2(2)·-6.5)) , (u_2(3)))))) , (u_2(4)))) , (u_2(5)))) , (Div ((Div ((Xlog ((Xlog ((Xcos ((Mul ((Xlog (((v_2(3)·10.0)) , (u_2(6)))) , (Sub (((v_2(4)·3.0)) , ((v_2(5)·-4.5)))))) , (u_2(7)))) , (u_2(8)))) , (u_2(9)))) , (Xlog ((Sub ((Xexp ((Div (((v_2(6)·-9.0)) , ((u_2(10)·x_{t-1})))) , (u_2(11)))) , (Add ((Sub (((u_2(12)·x_{t-8})) , ((v_2(7)·-2.5)))) , (Xcos (((v_2(8)·-0.5)) , (u_2(13)))))))) , (u_2(14))))) , (Mul ((Xexp ((Xcos ((Mul ((Xcos (((v_2(9)·-0.5)) , (u_2(15)))) , (Add (((u_2(16)·x_{t-16})) , ((v_2(10)·1.5)))))) , (u_2(17)))) , (u_2(18)))) , (Xcos ((Mul ((Xcos (((v_2(11)·-0.5)) , (u_2(19)))) , (Add (((u_2(20)·x_{t-16})) , ((v_2(12)·1.5)))))) , (u_2(21)))))))))) , (Mul (((v_2(13)·-1.5)) , ((u_2(22)·x_{t-1}))))))) , (u_2(23)))) , (u_2(24)))) , (Xlog ((Add ((Xcos (((v_2(14)·-3.5)) , (u_2(25)))) , (Add ((Sub (((u_2(26)·x_{t-8})) , ((v_2(15)·-2.5)))) , (Div ((Xexp ((Add ((Xcos ((Mul ((Xsin ((Sub ((Xexp ((Xlog (((v_2(16)·-9.5)) , (u_2(27)))) , (u_2(28)))) , (Mul (((v_2(17)·-1.5)) , ((u_2(29)·x_{t-1})))))) , (u_2(30)))) , (Add (((u_2(31)·x_{t-16})) , ((v_2(18)·1.5)))))) , (u_2(32)))) , (Xexp ((Xsin ((Sub ((Mul (((v_2(19)·-1.5)) , ((u_2(33)·x_{t-1}

)))) , (Mul ((Xcos ((($v_2(20)\cdot-0.5$)) , ($u_2(34)$))) , (Add ((($u_2(35)\cdot x_{t-16}$)) , (($v_2(21)\cdot1.5$)))))))) , ($u_2(36)$))) , ($u_2(37)$))))) , ($u_2(38)$))) , (Xexp ((Add ((Xcos ((Mul ((Xexp ((($v_2(22)\cdot8.0$)) , ($u_2(39)$))) , (Add ((($u_2(40)\cdot x_{t-16}$)) , (($v_2(23)\cdot1.5$)))))) , ($u_2(41)$))) , (Xexp ((Div ((Div ((Xlog ((Xcos ((Mul ((Xcos ((($v_2(24)\cdot-0.5$)) , ($u_2(42)$))) , (Add ((($u_2(43)\cdot x_{t-16}$)) , (($v_2(25)\cdot1.5$)))))) , ($u_2(44)$))) , ($u_2(45)$))) , (Xlog ((Xexp ((Xlog ((($v_2(26)\cdot-9.5$)) , ($u_2(46)$))) , ($u_2(47)$))) , ($u_2(48)$))))) , (Xlog ((Add ((Xlog ((($v_2(27)\cdot-6.5$)) , ($u_2(49)$))) , (Xexp ((Xlog ((($v_2(28)\cdot-9.5$)) , ($u_2(50)$))) , ($u_2(51)$))))) , ($u_2(52)$))))) , ($u_2(53)$))))) , ($u_2(54)$))))))))) , ($u_2(55)$))))) , ($u_2(56)$))) , (Sub ((Div ((Xexp ((Add ((Xcos ((Mul ((Xlog ((Xexp ((($v_2(29)\cdot8.0$)) , ($u_2(57)$))) , ($u_2(58)$))) , (Add ((Sub ((($u_2(59)\cdot x_{t-8}$)) , (($v_2(30)\cdot-2.5$)))) , (Div ((Xexp ((Add ((Xcos ((Mul ((Xlog ((Xlog ((Div ((($v_2(31)\cdot1.0$)) , (($u_2(60)\cdot x_{t-6}$)))) , ($u_2(61)$))) , ($u_2(62)$))) , (Add ((($u_2(63)\cdot x_{t-16}$)) , (($v_2(32)\cdot1.5$)))))) , ($u_2(64)$))) , (Xexp ((Xlog ((($v_2(33)\cdot-9.5$)) , ($u_2(65)$))) , ($u_2(66)$))))) , ($u_2(67)$))) , (Sub ((Xlog ((($v_2(34)\cdot10.0$)) , ($u_2(68)$))) , (Xexp ((Xexp ((Xexp ((Xlog ((($v_2(35)\cdot-9.5$)) , ($u_2(69)$))) , ($u_2(70)$))) , ($u_2(71)$))) , ($u_2(72)$)))))))))) , ($u_2(73)$))) , (Xexp ((Xlog ((($v_2(36)\cdot-9.5$)) , ($u_2(74)$))) , ($u_2(75)$))))) , ($u_2(76)$))) , (Sub ((Xlog ((Xexp ((($v_2(37)\cdot-8.0$)) , ($u_2(77)$))) , ($u_2(78)$))) , (Xlog ((Xexp ((($v_2(38)\cdot8.0$)) , ($u_2(79)$))) , ($u_2(80)$))))))) , (Div ((Xexp ((Xcos ((Xlog ((Xsin ((Xexp ((Mul ((Div ((Div ((Sub ((Xlog ((Mul ((($v_2(39)\cdot-1.5$)) , (($u_2(81)\cdot x_{t-1}$))))) , ($u_2(82)$))) , (Xlog ((Xexp ((($v_2(40)\cdot8.0$)) , ($u_2(83)$))) , ($u_2(84)$))))) , (Xlog ((Sub ((Mul ((Xcos ((($v_2(41)\cdot-0.5$)) , ($u_2(85)$))) , (Sub ((($v_2(42)\cdot3.0$)) , (($v_2(43)\cdot-4.5$)))))) , (Add ((($u_2(86)\cdot x_{t-16}$)) , (($v_2(44)\cdot1.5$)))))) , ($u_2(87)$))))) , (Xlog ((Add ((Mul ((Xexp ((Xcos ((($v_2(45)\cdot-3.5$)) , ($u_2(88)$))) , ($u_2(89)$))) , (Xlog ((($v_2(46)\cdot-6.5$)) , ($u_2(90)$))))) , (Xexp ((Mul ((Xcos ((($v_2(47)\cdot-0.5$)) , ($u_2(91)$))) , (Add ((($u_2(92)\cdot x_{t-16}$)) , (($v_2(48)\cdot1.5$)))))) , ($u_2(93)$))))) , ($u_2(94)$))))) , (Xsin ((Add ((Xexp ((Xsin ((Mul ((Xexp ((Xcos ((($v_2(49)\cdot-3.5$)) , ($u_2(95)$))) , ($u_2(96)$))) , (Xlog ((($v_2(50)\cdot-6.5$)) , ($u_2(97)$))))) , ($u_2(98)$))) , ($u_2(99)$))) , (Sub ((($u_2(100)\cdot x_{t-8}$)) , (($v_2(51)\cdot-2.5$)))))) , ($u_2(101)$))))) , ($u_2(102)$))) , ($u_2(103)$))) , ($u_2(104)$))) , ($u_2(105)$))) , ($u_2(106)$))) , (Add ((Xlog ((Xcos ((Xlog ((Xcos ((Xlog ((Xsin ((Add ((Sub ((($u_2(107)\cdot x_{t-8}$)) , (($v_2(52)\cdot-2.5$)))) , (Sub ((($u_2(108)\cdot x_{t-8}$)) , (($v_2(53)\cdot-2.5$)))))) , ($u_2(109)$))) , ($u_2(110)$))) , ($u_2(111)$))) , ($u_2(112)$))) , ($u_2(113)$))) , ($u_2(114)$))) , (Xlog ((Xcos ((Xlog ((Xsin ((Xexp ((Div ((($v_2(54)\cdot1.0$)) , (($u_2(115)\cdot x_{t-6}$)))) , ($u_2(116)$))) , ($u_2(117)$))) , ($u_2(118)$))) , ($u_2(119)$))) , ($u_2(120)$))))))))))

3. Rekursionsstufe

v_3 = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0}

u_3 = {-0.6925938246634, -0.8450701487198, 0.82884625622811, -3.7461159230467, 2.96180101219058, -3.1639398387199, -2.1042980035687, -0.2796697065060, 3.07590709472504, -2.0559887318618, 3.51604329090463, -1.9700562654818, 3.47943951672410, 0.35616333828805, -3.4729682327639, 3.20612074446293, -2.4048540211591, 4.99999999999999, 3.84452443549387, 2.83288074947560,

-3.9795608293284, -2.3743391715058, 3.90747559233183, 4.28350626943602, 4.84484107684578, 0.95502978260760, 0.87041936417434, -4.1076383793474, 3.64196617000498, -0.0903204694173, -4.6820797377143, 2.13744222828269, -4.9999999999999, 2.62242418607705, -3.6821325431272, 3.94012828421573, -0.7995623778820, 1.64712593798068, 1.61721293787298, -2.3803225243172, 3.08151015637154, -1.8758423077750, -1.2774702942939, -2.1838565871711, 4.07632008419133, -4.9999999999999, -4.9984425350323, 3.44601051414802, -4.7585171874685, -0.3854939281340, 4.19497430600511, -3.1057461103875, -0.9027393106217, -3.6526629726753, -4.8541628760964, -4.9902459627107, -1.9597449035534, -4.0926280643825, -0.1405590463555, 4.99999910592657, 0.40692206246545, -3.1578628842157, -0.3050462490276, 3.37476264593342, 2.97477296738194, -2.4908065371908, -3.9309489925775, -1.6362927571172, -3.4181017384553, -1.0447647646341, -2.1004435128160, 3.76597740085965, -3.3342816250352, -0.2859068214010, -0.7229528282265, -1.0430222818813, 0.24387712526006, -0.9011900098067, -4.4025476358988, -2.8794896782852, -4.5639587014128, 1.46002530283118, -4.9999988023039, -4.0641841522176, -4.1387098223190, -0.3218282447305, 4.99999990963398, 0.49966463985882, 4.25372290783471, 2.83052426119797, 4.25738818563115, 3.98419360596655, 3.18298804723164, 3.71372489361805, -2.4356723365264, -4.9568155917174, 2.55689950428139, -4.7326305755791, -1.5980132695681, 4.99999999999999, 0.69178801495283, -2.2167976444844, 4.96940972848017, -4.4000336779114, 1.79087507835009, -0.3438420727232, -1.4538553582952, -1.6148949461258, -1.5521973644060, 3.62313825166084, 2.17326336773601, -1.3848763402679, 2.37305410861186, 0.80884303368868, 3.90793757390493, -2.8321154733176, -1.8186786523208, 3.57646903588253, -4.8691376635807, -3.8287192201669, -0.9819271964792, 3.63065353643314, -4.5458059212017, -1.8337583166134, 4.99999999999999, -2.6177331723131, 1.46296721250282, -3.1936937626653, -4.9999984877716, 4.99999999306399, -0.9055849164287, -3.2206214504671, 0.11535680044777, 0.29477253318107, -3.7456175086865, 3.26334519331538, 2.12083120789570, -4.6241992150537, 3.91295596914832, 3.41825679748413, -2.8926670964753, -4.7903496666792, 2.11985264627988, 4.88263741086410, 1.98042874077456, 3.12301581475963, -0.7998085054148, 1.06489649268744, 1.31093496687747, -4.0800813571933, 0.25922263013010, 0.12602522372451, -4.6562503007705, 1.42223625908141, 3.66677131561901, -1.7385697309629}

$\hat{f}_{\text{sub}_3}$ = Xsin ((Div ((Xlog ((Add ((Xexp ((Sub ((Mul ((Sub ((Xcos ((($u_3(1)$·x_{t-12})) , ($u_3(2)$))) , (Mul ((Sub ((($v_3(1)$·8.5)) , (($v_3(2)$·-10.0)))) , (Mul ((Xcos ((Add ((Xexp ((Add ((Xcos ((Add ((($v_3(3)$·-3.5)) , (($u_3(3)$·x_{t-4})))) , ($u_3(4)$))) , (Xcos ((($u_3(5)$·x_{t-12})) , ($u_3(6)$))))) , ($u_3(7)$))) , (Add ((Sub ((($u_3(8)$·x_{t-12})) , (($u_3(9)$·x_{t-18})))) , (Xcos ((($u_3(10)$·x_{t-12})) , ($u_3(11)$)))))))) , ($u_3(12)$))) , (Add ((Sub ((($u_3(13)$·x_{t-12})) , (($u_3(14)$·x_{t-18})))) , (Add ((Sub ((($u_3(15)$·x_{t-12})) , (($u_3(16)$·x_{t-18})))) , (Xcos ((($u_3(17)$·x_{t-12})) , ($u_3(18)$)))))))))))))) , (Xcos ((($u_3(19)$·x_{t-12})) , ($u_3(20)$))))) , (Mul ((Sub ((($v_3(4)$·8.5)) , (($v_3(5)$·-10.0)))) , (Mul ((Xcos ((Add ((Xexp ((Add ((Xcos ((($v_3(6)$·8.0)) , ($u_3(21)$)))) , (Mul ((Add ((($v_3(7)$·-3.5)) , (($u_3(22)$·x_{t-4})))) , (Xlog ((($v_3(8)$·1.0)) , ($u_3(23)$)))))))) , ($u_3(24)$))) , (Add ((Sub ((($u_3(25)$·x_{t-12})) , (($u_3(26)$·x_{t-18})))) , (Xcos ((($u_3(27)$·x_{t-12})

) , (u_3(28)))))))) , (u_3(29)))) , (Add ((Xcos ((Xcos ((Add ((Xexp ((Add ((Xcos ((Add (((v_3(9)·-3.5)) , ((u_3(30)·x_{t-4})))) , (u_3(31)))) , (Xcos (((u_3(32)·x_{t-12})) , (u_3(33)))))) , (u_3(34)))) , (Add ((Xcos (((u_3(35)·x_{t-12})) , (u_3(36)))) , (Xcos (((u_3(37)·x_{t-12})) , (u_3(38))))))))) , (u_3(39)))) , (u_3(40)))) , (Add ((Sub (((u_3(41)·x_{t-12})) , ((u_3(42)·x_{t-18})))) , (Xcos (((u_3(43)·x_{t-12})) , (u_3(44)))))))))))))) , (u_3(45)))) , (Add ((Sub (((u_3(46)·x_{t-12})) , ((u_3(47)·x_{t-18})))) , (Xcos (((u_3(48)·x_{t-12})) , (u_3(49)))))))) , (u_3(50)))) , (Xexp ((Add ((Xexp ((Sub ((Mul ((Xcos (((u_3(51)·x_{t-12})) , (u_3(52)))) , (Mul ((Xlog ((Mul ((Xlog ((Xexp ((Mul ((Xlog ((Xcos (((u_3(53)·x_{t-12})) , (u_3(54)))) , (u_3(55)))) , (Xcos ((Add (((v_3(10)·-3.5)) , ((u_3(56)·x_{t-4})))) , (u_3(57)))))) , (u_3(58)))) , (u_3(59)))) , (Xcos ((Add (((v_3(11)·-3.5)) , ((u_3(60)·x_{t-4})))) , (u_3(61)))))) , (u_3(62)))) , (Xcos ((Mul ((Xlog ((Xlog ((Xcos (((u_3(63)·x_{t-12})) , (u_3(64)))) , (u_3(65)))) , (u_3(66)))) , (Xcos (((u_3(67)·x_{t-12})) , (u_3(68)))))) , (u_3(69)))))))) , (Mul ((Add (((v_3(12)·-2.0)) , ((v_3(13)·10.0)))) , (Mul ((Xcos ((Add ((Xcos ((Xcos (((u_3(70)·x_{t-12})) , (u_3(71)))) , (u_3(72)))) , (Add ((Sub (((u_3(73)·x_{t-12})) , ((u_3(74)·x_{t-18})))) , (Xcos (((u_3(75)·x_{t-12})) , (u_3(76)))))))) , (u_3(77)))) , (Add ((Mul ((Xlog ((Xcos (((v_3(14)·8.0)) , (u_3(78)))) , (u_3(79)))) , (Xcos ((Add ((Xlog (((u_3(80)·x_{t-8})) , (u_3(81)))) , (Xexp (((v_3(15)·-9.0)) , (u_3(82)))))) , (u_3(83)))))) , (Mul ((Add ((Mul ((Xcos ((Add ((Sub (((u_3(84)·x_{t-12})) , ((u_3(85)·x_{t-18})))) , (Xlog (((v_3(16)·1.0)) , (u_3(86)))))) , (u_3(87)))) , (Xcos (((u_3(88)·x_{t-12})) , (u_3(89)))))) , (Add (((v_3(17)·-2.0)) , ((v_3(18)·10.0)))))) , (Sub (((u_3(90)·x_{t-12})) , ((u_3(91)·x_{t-18})))))))))))))))))) , (u_3(92)))) , (Add ((Xsin ((Div ((Xlog ((Xcos (((u_3(93)·x_{t-12})) , (u_3(94)))) , (u_3(95)))) , (Xexp ((Mul ((Xlog ((Xexp ((Mul ((Xlog ((Xcos (((v_3(19)·8.0)) , (u_3(96)))) , (u_3(97)))) , (Xcos ((Xlog ((Xcos (((u_3(98)·x_{t-12})) , (u_3(99)))) , (u_3(100)))) , (u_3(101)))))) , (u_3(102)))) , (u_3(103)))) , (Xcos ((Xexp ((Mul ((Xlog ((Xlog (((u_3(104)·x_{t-12})) , (u_3(105)))) , (u_3(106)))) , (Xcos ((Add (((v_3(20)·-3.5)) , ((u_3(107)·x_{t-4})))) , (u_3(108)))))) , (u_3(109)))) , (u_3(110)))))) , (u_3(111)))))) , (u_3(112)))) , (Add ((Xexp ((Sub ((Xsin ((Mul ((Mul ((Xexp ((Xexp ((Xlog ((Xcos (((u_3(113)·x_{t-12})) , (u_3(114)))) , (u_3(115)))) , (u_3(116)))) , (u_3(117)))) , (Xexp (((v_3(21)·-9.0)) , (u_3(118)))))) , (Mul ((Xlog ((Xcos (((u_3(119)·x_{t-12})) , (u_3(120)))) , (u_3(121)))) , (Xlog (((v_3(22)·1.0)) , (u_3(122)))))))) , (u_3(123)))) , (Add ((Xlog ((Xexp ((Xcos (((u_3(124)·x_{t-12})) , (u_3(125)))) , (u_3(126)))) , (u_3(127)))) , (Add ((Xcos ((Xcos (((u_3(128)·x_{t-12})) , (u_3(129)))) , (u_3(130)))) , (Xsin ((Mul ((Mul ((Xlog ((Xcos (((u_3(131)·x_{t-12})) , (u_3(132)))) , (u_3(133)))) , (Xexp (((v_3(23)·-9.0)) , (u_3(134)))))) , (Mul ((Xlog ((Xcos (((u_3(135)·x_{t-12})) , (u_3(136)))) , (u_3(137)))) , (Xlog (((v_3(24)·1.0)) , (u_3(138)))))))) , (u_3(139)))))))))) , (u_3(140)))) , (Add ((Xsin ((Div ((Xlog ((Xcos (((u_3(141)·x_{t-12})) , (u_3(142)))) , (u_3(143)))) , (Xexp ((Mul ((Xlog ((Xexp ((Mul ((Xlog ((Xcos (((u_3(144)·x_{t-12})) , (u_3(145)))) , (u_3(146)))) , (Xcos ((Add (((v_3(25)·-3.5)) , ((u_3(147)·x_{t-4})))) , (u_3(148)))))) , (u_3(149)))) , (u_3(150)))) , (Xcos ((Add (((v_3(26)·-3.5)) , ((u_3(151)·x_{t-4})))) , (u_3(152)))))) , (u_3(153)))))) , (u_3(154)))) , (Add (((v_3(27)·-2.0)) , ((v_3(28)·10.0)))))))))))))) , (u_3(155)))))) , (u_3(156)))

Literaturverzeichnis

[Aho72] A. V. Aho und J. D. Ullman: *The Theory of Parsing, Translation and Compiling*, Prentice Hall Inc., New Jersey, 1972.

[Aho88] A. V. Aho, R. Sethi und J. D. Ullman: *Compilerbau*, Addison-Wesley Co., Bonn, 4. Auflage, 1988.

[Ang96] P. J. Angeline und K. E. Kinnear, Jr.: *Advances in Genetic Programming*, Vol. 2, MIT Press, Cambridge, MA, 1996.

[Ang97a] P. J. Angeline: *Comparing Subtree Crossover with Macro Mutation*, In: *The Sixth Conference on Evolutionary Programming*, P. J. Angeline, R. G. Reynolds, J. R. McDonnell and R. Eberhart (Ed.), Springer-Verlag, Berlin, S.101-111, 1997.

[Ang97b] P. J. Angeline: *Subtree Crossover: Building Block Engine or Macromutation?*, In: *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. Koza, K. Deb, M. Dorigo, D. Fogel, M. Garzon, H. Iba and R. Riolo (Ed.), Morgan Kaufmann, San Francisco, CA, S.9-17, 1997.

[Ang98] P. J. Angeline: *Evolving Predictors for Chaotic Time Series*, In: *Proceedings of SPIE (Vol. 3390): Application and Science of Computational Intelligence*, S. Rogers, D. Fogel, J. Bezdek und B. Bosacchi (Ed.), Bellingham, WA, S.170-180, 1998.

[Arr94] D. K. Arrowsmith und C.M. Place: *Dynamische Systeme: Mathematische Grundlagen · Übungen*, Spektrum Akademischer Verlag, Heidelberg, 1994.

[Bae97a] T. Bäck, U. Hammel, H. Schwefel: *Evolutionary Computation: Comments on the History and Current State*, In: *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, S.3-16, April, 1997.

[Bae97b] T. Bäck, D. Fogel, Z. Michalewicz: *Handbook of Evolutionary Computation*, Oxford University Press, Oxford 1997.

[Ban96] W. Banzhaf, F. D. Francone, P. Nordin: *The Effect of Extensive Use of the Mutation Operation on Generalization in Genetic Programming using Sparse Data Sets*, In: *4th Int. Conf. on Parallel Problem Solving from Nature (PPSN96)*, W. Ebeling, I. Rechenberg, H.P. Schwefel, H.M. Voigt(Ed.), Springer, Berlin, S.300-309, 1996.

[Ban98] W. Banzhaf, P. Nordin, R. E. Keller und F.D. Francone: *Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1998.

[Ban00] W. Banzhaf, J. R. Koza, C. Ryan, L. Spector, C. Jacob: *Genetic progamming*, In: *IEEE Intelligent Systems*, Vol. 15, No. 13, S.74-84, 2000.

[Boy91] C. B. Boyer: *A History of Mathematics*, J. Wiley & Sons, 1991.

[Bro93] I. N. Bronstein, K. A. Semendjajew, G. Musiol und H. Mühlig: *Taschenbuch der Mathematik*, Verlag Harri Deutsch, 1.Aufl. Thun und Frankfurt am Main, 1993.

[Cao00] H. Cao, L. Kang, T. Gao, Y. Chen und H. de Garis: *A Two-Level Hybrid Evolutionary Algorithm for Modeling One-Dimensional Dynamic Systems by Higher-Order ODE Models*, In: *IEEE Transactions on Systems, Man and Cybernetics*, Part B, Vol. 30, No. 2, S.351-357, 2000.

[Cas89] M. Casdagli: *Nonlinear Prediction of Chaotic Time Series*, In: *Physica D: Nonlinear Phenomena*, Vol. 35 (1989), North-Holland Physics Publishing, Amsterdam, S.335-356, 1989.

[Cas92] M. Casdagli: *Nonlinear Modeling of Chaotic Time Series: Theory and Applications*, In: *Applied Chaos*, J. H. Kim und J. Stringer (Ed.), John Wiley & Sons, Inc., New York, 1992.

[Cha99] L. Chan: *Weighted Least Square Ensemble Networks*, In: *International Joint Conference on Neural Networks*, Vol. 2, S.1393-1396, 1999.

[Che97] K. Chellapila: *Evolving Computer Programs Without Subtree Crossover*, In: *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 3, S.209-216, September 1997.

[Clo98] G. Cloarec und J. Ringwood: *Incorporation of Statistical Methods in Multistep Neural Network Prediction Models*, In: *IEEE International Joint Conference on Neural Networks*, Vol. 3, S.2513-2518, 1998.

[Dev01] J. Devaney, J. Hagedorn, O. Nicolas, G. Garg, A. Samson und M. Michel: *A Genetic Programming Ecosystem*, In: *Parallel and Distributed Processing Symposium, Proceedings 15th International*, San Francisco, CA, S.1323-1330, 2001.

[Eck81] J. P. Eckmann: *Roads to Turbulence in Dissipative Dynamical Systems*, In: *Review of Modern Physics*, 53, S.643-654, 1981.

[Farm82] J. D. Farmer: *Chaotic Attractors of an Infinite-Dimensional Dynamical System*, In: *Physica* 4D, S.366-393, 1982.

[Fog66] L. J. Fogel, A. J. Owens und M. J. Walsh: *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

[Fog00] D. B. Fogel: *What is evolutionary computation?*, In: *IEEE Spectrum*, Vol. 37, No. 2, S.26-32, February, 2000.

[For00] C. Forster: *Entwicklung eines Programmpakets zur Parellelisierung des Fitnesstests bei der Genetischen Programmierung auf verteilte Rechnersysteme*, Diplomarbeit, Institut für Nachrichtentechnik, Fakutät für Elektrotechnik, Universität der Bundeswehr München, 2000.

[Free98] J. J. Freeman: *A Linear Representation for GP using Context Free Grammars*, In: *Proceedings of the Third Annual Conference on Genetic Programming 1998*, Morgan Kaufmann, S.72-77, 1998.

[Ger93] N. A. Gershenfel und A. S. Weigend: *The Future of Time Series: Learning and Understanding*, In: *Time Series Prediction: Forecasting the Future and Understanding the Past*, Proceedings of the NATO Advanced Research Workshop on Comparative Time Series Analysis, Vol. XV, Addison-Wesley Publishing Company, Reading, Massachusetts, USA, S.1-70, 1994.

[Gey97] A. Geyer-Schulz: *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, In: *Studies in Fuzziness and Soft Computing*, Vol. 3, 2. Auflage, Physica-Verlag, 1997.

[Goh00] W. Y. Goh, C. P. Lim, K. K. Peh und K. Subari: *A Neural-Network-Based Intelligent System for Time-Series Prediction Problems in Product Development*, *TENCON 2000. Proceedings*, Vol. 1, Kuala Lumpur, Malaysia, S.151 - 155, 2000.

[Gold89] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA, 1989.

[Gray96] G. J. Gray, Y. Li, D. J. Murray-Smith und K. C. Sharman: *Structural system identification using genetic programming and a block diagram oriented simulation tool*, In: *Electronics Letters*, Vol. 32, No. 15, S.1422-1424, 18 Juli 1996.

[Gray97] G. J. Gray, T. Weinbrenner, D. J. Murray-Smith, Y. Li und K. C. Sharman: *Issues in Nonlinear Model Structure Identification Using Genetic Programming*, In: *GALESIA 97, Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, Conference Publication No. 446, S.308-313, 1997.

[Hao90] Hao Bai Lin: *Chaos II*, World Scientific, Singapur, 1990.

[Her99] H. Herold: *Linux-Unix Systemprogrammierung*, Addison Wesley Verlag, München, 2. Auflage, 1999.

[Hol62] J. H. Holland: *Outline for a logical theory of adaptive systems*, In: *Journal of the Association for Computing Machinery*, Vol. 3, S. 297-314, 1962.

[Hoer96] H. Hörner: *Ein Kern für genetisches Programmieren*, Diplomarbeit, Wirtschaftsuniversität Wien, 1996.

[Houck95] C. H. Houck, J. A. Joines und M. G. Kay: *A Genetic Algorithm for Function Optimization: A Matlab Implementation*, In: http://www.ie.ncsu.edu/mirage/GAToolBox/gaot/

[Iba94] H. Iba, T. Sato und H. de Garis: *System Identification Approach to Genetic Programming*, In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Vol. 1, S.401-406, 1994.

[Kab01] M. A. Kaboudan: *Compumetric Forecasting of Crude Oil Prices*, *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 1, Seoul, South Korea, S.283-287, 2001.

[Kan97] H. Kantz und T. Schreiber: *Nonlinear time series analysis*, Cambridge University Press, 1997.

[Kel96] R. E. Keller und W. Banzhaf: *Genetic Programming using Mutation, Reproduction and Genotype-Phenotype Mapping from linear binary Genomes into linear LALR(1) Phenotypes*, In: *Proceedings of Genetic Programming 1996*, MIT Press, Cambridge, MA, S.116-122, 1996.

[Kha98] A. A.M.Khalaf und K. Nakayama: *Times Series Prediction Using a Hybrid Model of Neural Network and FIR filter*, In: *IEEE International Joint Conference on Neural Networks*, Vol. 3, 1975-1980, 1998.

[Kim96] B. Kim und K. Park: *A Study on the Modeling of Nonlinear System Using Genetic Programming*, In: *18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Amsterdam, S.1668-1669, 1996.

[Kin94] K. E. Kinnear, Jr.: *Advances in Genetic Programming*, Cambridge, MA, MIT Press, 1994.

[Kinne94] W. Kinnebrock: *Optimierung mit genetischen und selektiven Algorithmen*, Oldenbourg Verlag, München, 1994.

[Koza92] J. R. Koza: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

[Koza94] J. R. Koza: *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA, 1994.

[Krai01] C. Kraikhaw: *Modellieren deterministische chaotische System*, Diplomarbeit, Institut für Nachrichtentechnik, Fakutät für Elektrotechnik, Universität der Bundeswehr München, 2000.

[Lan01] M. Landajo, M. J. Río und R. Pérez: *A Note on Smooth Approximation Capabilities of Fuzzy Systems*, In: *IEEE Transactions on Fuzzy Systems*, Vol. 9, No. 2, S.229-237, 2001.

[Lee99] G. Y. Lee: *Genetic Recursive Regression for Modeling and Forecasting Real-World Chaotic Time Series*, In: *Advances in Genetic Programming*, Vol. 3, L. Spector, W. B. Langdon, U. O'Reilly und P. J. Angeline (Ed.), MIT Press, Cambridge, S.401-423, 1999.

[Lee01] G. Y. Lee: *Time Series Perturbation by Genetic Programming*, In: *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 1, Seoul, South Korea, S.403-409, 2001.

[Lev89] R. W. Leven, B.-P. Koch und B. Pompe: *Chaos in dissipativen Systemen*, Vieweg, Braunschweig/Wiesbaden, 1989.

[Loi93] O. Loistl und I. Betz: *Chaostheorie: Zur Theorie nichtlinearer dynamischer Systeme*, R. Oldenbourg Verlag GmbH, München, 1993.

[Lor63] E. Lorenz: *Deterministic Nonperiodic Flow*, In: *Journal of the Atmospheric Sciences*, Vol. 20, S.130-141, 1963.

[Lor69] E. Lorenz: *Atmospheric predictability as revealed by naturally occurring analogies*, In: *J. Atmos. Sci.m* Vol. 26, No. 636, 1969.

[Luke97] S. Luke and L. Spector: *A Comparison of Crossover and Mutation in Genetic Programming*, In: *Genetic Programming 1997: Proc. Second Annu. Conference* San Francisco, CA: Morgan Kauffmann, S.240-248, 1997.

[Mac77] M. C. Mackey und L. Glass: *Oscillation and Chaos in Physiological Control Systems*, In: *Science*, Vol. 197, S.287-289, 1977.

[Man97] K. F. Man, K. S. Tang, S. Kwong und W. A. Halang: *Genetic Algorithms for Control and Signal Processing*, Springer-Verlag, London, 1997.

[Mat01] T. Matsumoto, Y. Nakajima, M. Saito, J. Sugi und H. Hamagishi: *Reconstructions and Predictions of Nonlinear Dynamical Systems: A Hierarchical Bayesian Approach*, In: *IEEE Transactions on Signal Processing*, Vol. 49, No. 9, S.2138-2155, 2001.

[Mau98] R. S. Maust und R. L. Klein: *Nonlinear MISO Modeling Using Genetic Programming*, In: *Symposium on System Theory*, Proceedings of the Thietieth Southeastern, S.42-46, 1998.

[May76] R. M. May: *Simple mathematical model with very complicated dynamics*, In: *Nature*, Vol. 261, S.459-467, 1976.

[Mey92] T. P. Meyer und N. H. Packard: *Local Forecasting of High-Dimensional Chaotic Dynamics*, In: *Nonlinear Modelling and Forecasting (Proceedings of The Workshop, Santa Fe, NM, Sept. 1990)*, M. Casdagli und S. G. Eubank (Ed.), Proceedings Vol. XII, Addison-Wesley Publishing Company, Redwood City, California, S.249-263, 1992.

[Moe92] D. Möller: *Modellbildung, Simulation und Identifikation dynamischer Systeme*, Springer-Verlag, Berlin, 1992.

[Mul96] B. S. Mulloy, R. L. Riolo and R. S. Savit: *Dynamics of Genetic Programming and Chaotic Time Series Prediction*, In: *Genetic Programming: Proceedings of the First Annual Conference*, J. Koza (Ed.), MIT Press, Cambridge, Massachusetts, S.166-174, 1996.

[Mun91] M. D. Mundt, W. B. Maguire II und R. P. Chase: *Chaos in the Sunspot Cycle: Analysis and Prediction*, In: *Journal of Geophysical Research*, Vol. 96, No. A2, S.1705-1716, 1991.

[Nag92] H. Nagashima und Y. Baba: *Introduction to Chaos: Physics and Mathematics of Chaotic Phenomena*, Institute of Physics Publishing Bristol and Philadelphia, 1999.

[Nak00] Y. Nakada und T. Matsumoto: *Bayesian MCMC Nonlinear Time Series Prediction: Predictive Mean and Error Bar*, In: *Proceedings of the 2000 IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing X*, Vol. 1, Sydney, NSW, Australia, S.155-164, 2000.

[Nak01] Y. Nakada und T. Matsumoto: *Bayesian MCMC Nonlinear Time Series Prediction*, In: *Proceedings of the 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 6, S.3509 - 3512, 2001.

[Neal96] R. M. Neal: *Bayesian Learning for Neural Networks*, Springer-Verlag, New York, 1996.

[Nik00] N. Nikolaev und H. Iba: *Inductive Genetic Programming of Polynomial Learning Networks*, In: *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, S.158-167, 2000.

[Nik01] N. Nikolaev und H. Iba: *Regularization Approach to Inductive Genetic Programming*, In: *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 4, S.359 - 375, 2001.

[Nis97] V. Nissen: *Einführung in Evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution*, Vieweg, Braunschweig/Wiesbaden 1997.

[Oak94] H. Oakley: *Two Scientific Applications of Genetic Programming: Stack Filters and Non-Linear Equation Fitting to Chaotic Data*, In: *Advances in Genetic Programming*, K. E. Kinnear (Ed.), MIT Press, Cambridge, Massachusetts, S.369-389, 1994.

[One99] M. O'neill und C. Ryan: *Evolving Multi-line Compilable C Programs*, In: *Proceedings of the Second European Workshop on Genetic Programming 1999*, 1999.

[One01] M. O'neill und C. Ryan: *Grammatical Evolution*, In: *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 4, S.349-358, 2001.

[Ous96] M. Oussaidène, B. Chopard, O. V. Pictet und M. Tomassini: *Parallel Genetic Programming: an application to Trading Models Evolution*, In: *Proceedings of the Genetic Programming 1996 Conference*, Stanford, MIT Press, S.357-362, 1996.

[Ous97] M. Oussaidène, B. Chopard, O. V. Pictet und M. Tomassini: *Parallel Genetic Programming and its application to trading models induction*, In: *Parallel Computing*, 23, S.1183-1198, 1997.

[Pat97] N. Paterson, M. Livesey: *Evolving caching algorithms in C by genetic programming*, In: *Proceedings of Genetic Programming 1997*, MIT Press, S.262-267, 1997.

[Pham99] D. T. Pham und X. Liu: *Neural Networks for Identification, Prediction and Control*, Springer-Verlag London, 4. Auflage, 1999.

[Pre98] L. Prechelt: *Automatic Early Stopping Using Cross Validation: Quantifying the Criteria*, In: *Neural Networks 11(4)*, S.761-767, 1998.

[Rec65] I. Rechenberg: *Cybernetic solution path of an experimental problem*, Royal Aircraft Establishment, Library translation No. 1122, Farnborough, Hants., U.K., August 1965.

[Rod68] I. Rodríguez-Iturbe und V. Yevjevich: *The Investigation of Relationship between Hydrologic Time Series and Sunspot Numbers*, Hydrology Papers, Colorado State University, Fort Collins, Colorado, 1968.

[Rod97] K. Rodríguez-Vázquez und P. J. Fleming: *A Genetic Programming/NARMAX Approach to Nonlinear System Identification*, In: *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA 97*, Conference Publication No. 446, S.409-414 , 1997.

[Rod99] K. Rodríguez-Vázquez und P. J. Fleming: *Genetic Programming for Dynamic Chaotic Systems Modelling*, In: *Proceeding of the 1999 Congress on Evolutionary Computation, CEC 99*, S.22-28 , 1999.

[Sar01] W. S. Sarle: *Neural Network Frequently Asked Questions*, In: ftp://ftp.sas.com/pub/neural/FAQ3.html, Letzte Änderung 21.05.2001.

[Sche97] A. Scherer: *Neuronale Netze: Grundlagen und Anwendungen*, Vieweg, Braunschweig/Wiesbaden, 1997.

[Schn96] B. Schneier: *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*, Addison-Wesley, Bonn, 1996.

[Scho94] E. Schöneburg, F. Heinzmann, S. Feddersen: *Genetische Algorithmen und Evolutionsstrategien: Eine Einführung in Theorie und Praxis der simulierten Evolution*, Addison-Wesley, Bonn, 1994.

[Schw68] H. -P. Schwefel: *Projekt MHD-Staustrahlrohr: Experimentelle Optimierung einer Zweiphasendüse. Teil I*, Technischer Bericht 11.034/68, 35, AEG Forschungsinstitut, Berlin, Germany, Oktober 1968.

[Shar95] K. C. Schrman, A. I. Esparcia Alcázar und Y. Li: *Evolving Signal Processing Algorithms by Genetic Programming*, In: *Genetic Algorithms in Engineering Systems: Innovations and Applications*, Conference Publication No. 414, S.473-480, 1995.

[She01] A. F. Sheta und A. Mahmoud: *Forecasting Using Genetic Programming*, In *Proceedings of the 33rd Southeastern Symposium on System Theory*, Athens, OH, USA, S.343 - 347, 2001.

[Smith94] L. A. Smith: *Does a Meeting in Santa Fe Imply Chaos?*, In: *Time Series Prediction: Forecasting the Future and Understanding the Past*, Proceedings of the NATO Advanced Research Workshop on Comparative Time Series Analysis, Vol. XV, Addison-Wesley Publishing Company, Reading, Massachusetts, S.323-343, 1994.

[South96] M. South, C. Bancroft, M. J. Willis und M. T. Tham: *System Identification Via Genetic Programming*, In: *International Conference on Control 96, UKACC*, Conference Publication No. 427, Vol. 2, S.912-917, 1996.

[Spec99] L. Spector, W. B. Langdon, U. O'Reilly und P. J. Angeline: *Advances in Genetic Programming*, Vol. 3, Cambridge, MA, MIT Press, 1999.

[Tan01] C. L. Tan, H. W. K. Chia: *Genetic Construction of Neural Logic Networks*, In: *Proceedings of the 2001 International Joint Conference on Neural Networks, IJCNN 01*, Vol. 1, Washington, DC, USA, S.732-737, 2001.

[Tane01] I. Tanev, T. Uozumi und K. Ono: *Parallel Genetic Programming: Component Object-based Distributed Collaborative Approach*, In: *Proceedings of the 15th International Conference on Information Networking*, Beppu City, Oita, Japan, S.129 -136, 2001.

[Tom99] M. Tomassini: *Parallel and Distributed Evolutionary Algorithms: A Review*, In: *Evolutionary Algorithm in Engineering and Computer Science*, K. Miettinen, M. Mäkelä, P. Neittaanmäki und J. Periaux (Ed.), J. Wiley and Sons, Chichester, S.113-133, 1999.

[Tso92] A. A. Tsonis: *Chaos: From Theory to Applications*, Plenum Press, New York, 1992.

[Ulu98] Ö. Uluyol, M. Ragheb und S. R. Ray: *Local Output Gamma Feedback Neural Network*, In: *International Joint Conference on Neural Networks, IJCNN 00*, Vol. 1, S.337-342, 1998.

[Vaz01] K. R. Vázquez: *Genetic Programming in Time Series Modelling: An Application to Meteorological Data*, *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 1, Seoul, South Korea, S.261-266, 2001.

[Wak00] H. Wakuya und J. M. Zurada: *Time Series Prediction by a Neural Network Model Based on the Bi-Directional Computation Style*, In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 00*, Vol. 2, Como, Italy, S.225 -230, 2000.

[Wak01] H. Wakuya und K. Shida: *Bi-Directionalization of Neural Computing Architecture for Time Series Prediction. III: Application to Laser Intensity Time Record "Data Set A"*, In: *Proceedings of the 2001 International Joint Conference on Neural Networks, IJCNN 01*, Vol. 3, Washington, DC, USA, S.2098-2103, 2001.

[Wal99] P. J. Walsh: *A Parallel Evolutionary Algorithm for High-Level Program Induction*, In: *IEEE Proceedings of the 1999 Congress on Evolutionary Computation*, Vol. 2, Washington, DC, S.1027-1033 ,1999.

[Wan01] L. Wang, K. K. Teo und Z. Lin: *Predicting Time Series with Wavelet Packet Neural Networks*, In: *Proceedings of the 2001 International Joint Conference on Neural Networks*, Vol. 3, Washington, DC, S.1593-1597, 2001.

[Wein97] T. Weinbrenner: *GPC++ - Genetic Programming C++ Class Library*, Diplomarbeit, Institut für elektromechanische Konstruktionen, Technische Universität Darmstadt, 1997.

[Weig91] A. S. Weigend und D. E. Rumelhart: *The Effective Dimension of the Space of Hidden Units*, In: *1991 IEEE International Joint Conference on Neural Networks*, Vol. 3 of 3, S.2069-2074, 1991.

[Weig92] A. S. Weigend, B. A. Huberman und D. E. Rumelhart: *Predicting Sunspots and Exchange Rates with Connectionist Networks*, In: *Nonlinear Modelling and Forecasting (Proceedings of The Workshop, Santa Fe, NM, Sept. 1990)*, M. Casdagli und S. G. Eubank (Ed.), Proceedings Vol. XII, Addison-Wesley Publishing Company, Redwood City, California, S.395-432, 1992.

[Whig96] P. A. Whigham: *Grammatical Bias for Evolutionary Learning*, PhD. Thesis, University of New South Wales, Australian Defence Force Academy, 1996.

[Whig01] P. A. Whigham und J. Keukelaar: *Evolving Structure - Optimising Content*, In: *IEEE Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 2, Seoul, South Korea, S.1228-1235, 2001.

[Wong95] M. L. Wong und K. S. Leung: *Applying logic grammars to induce sub-functions in genetic programming*, In: *Proceedings of the 1995 IEEE conference on Evolutionary Computation*, USA, IEEE Press, S.737-740, 1995.

[Yos99] I. Yoshihara, M. Numata, K. Sugawara, S. Yamada und K. Abe: *Time Series Prediction Model Building with BP-like Parameter Optimization*, In: *Proceedings of the 1999 Congress on Evolutionary Computation*, S.295-301, 1999.

[Yos00] I. Yoshihara, T. Aoyama, M. Yasunaga: *GP-Based Modeling Method for Time Series Prediction with Parameter Optimization and Node Alternation*, In: *Proceedings of the 2000 Congress on Evolutionary Computation*, S.1475-1481, 2000.

[Zha99] B. Zhang und J. Joung: *Times Series Prediction Using Committee Machines of Evolutionary Neural Trees*, In: *Proceedings of the 1999 Congress on Evolutionary Computation*, Vol. 1, Washington, DC, USA, S.281-286, 1999.