

DIPLOMARBEIT

Approximative Ein-Ausgangslinearisierung von nichtlinearen dynamischen Systemen mittels neuronaler Netze und Regelung derselben

**ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Diplom-Ingenieurs unter der Leitung von**

o.Univ.Prof. Dipl.-Ing. Dr.techn. Alexander Weinmann

und

Dipl.-Ing. Dr.techn. Alexander Vieweider

als verantwortlich mitwirkende Universitätsassistenten

am Inst.376 Institut für Automatisierungs- und Regelungstechnik

eingereicht an der Technischen Universität Wien,

Fakultät für Elektrotechnik

von

Rouzbeh Karim

Matr.Nr 9427229

Währingerstrasse 202/4

1180 Wien

Wien

Datum

14.12.2004

Kurzfassung

Diese Diplomarbeit befasst sich mit der Problemstellung der Regelung nichtlinearer unbekannter Strecken.

Zur Lösung des Problems ist eine approximative Methode vorgestellt worden. Die approximative Methode basiert auf exakten Linearisierungsmethoden.

Nachdem eine nichtlineare Strecke identifiziert wird, können die Linearisierungen der Strecke eingesetzt werden.

Die Identifizierung der nicht linearen Strecke bedeutet hier die Erfassung der Zustandsraumdarstellung der Strecke.

Anhand der Zustandsraumdarstellung kann dementsprechend eine Feedbacklinearisierung durchgeführt werden.

Ein Linearisierungsverfahren kann entweder exakt realisiert werden, oder es kann auch approximiert werden.

Als Approximierungseinheiten können verschiedene Elemente eingesetzt werden.

Als Linearisierungsbausteine fungieren verschiedene Elemente die eine Rolle in der Linearisierungsstruktur spielen. Interpolationseinheiten, neuronale Netze sind zwei der bekanntesten Elemente, die im Zusammenhang mit Linearisierung benutzt werden können.

Im Rahmen dieser Diplomarbeit ist die approximative Linearisierung getestet worden.

Als Linearisierungseinheiten werden zwei verschiedene Bausteine verwendet. Einer ist das neuronale Netz und der andere ist die Interpolationseinheit.

Nach der Linearisierung der Strecke kann die linearisierte Strecke wie ein lineares System behandelt werden. Deswegen ist zum Stabilisieren der Strecke das Design eines Reglers notwendig. Aufgrund der Notwendigkeit der Stabilität sind alle getesteten Strecken sowohl linearisiert als auch geregelt worden. Eine andere Tendenz neben der Stabilisierung ist das Tracking. Beim Tracking wird versucht Führungsgröße (Sollwert) zu erreichen.

Eine Linearisierung inklusive des Trackings ist ebenfalls getestet worden.

Inhaltsverzeichnis

Einleitung	5
1. Exakte Zustandslinearisierung.....	7
1.1. Linearisierung.....	8
1.2. Koordinatentransformation.....	9
1.2.1. Nulldynamik und Unbeobachtbarkeit.....	9
1.2.2. Starker relativer Grad	10
1.2.3. Koordinatentransformation.....	11
1.3. Reglerdesign	15
1.4. Tracking.....	17
2. Adaptive exakte Linearisierung.....	19
2.1. Adaptionssysteme	19
2.2. Stabilität.....	23
3. Neuronale Netze zur Identifizierung beziehungsweise Linearisierung und Regelung der nichtlinearen dynamischen Systeme	24
3.1. Aufbau neuronaler Netze	24
3.1.1. Netze	24
3.1.2. Neuronenmodellierung	24
3.1.3. Knoten	24
3.1.4. Kanten.....	25
3.2. Lernalgorithmus	25
3.2.1. Backpropagation.....	26
3.2.2. Backpropagation mit Momentum Term.....	33
3.2.3. Gauss-Newton Methode	33
3.2.4. Levenberg Marquardt Algorithmus	36
4. Approximative Zustandslinearisierung mit neuronalen Netzen und Interpolationsmethoden.....	38
4.1. Identifizierung, Modellierung und Approximierung der unbekannten nichtlinearen Objekte.	38
4.1.1. Identifizierung und Modellierung der nichtlinearen Systemmatrix $f(x)$	39
4.1.2. Identifizierung und Modellierung der Zustandsgrößen $f(x)+g(x)$	41
4.1.3. Identifizierung und Modellierung der nichtlinearen Steuermatrix $g(x)$	42
4.1.4. Approximierung und Modellierung der Linearisierungseinheit	44
4.2. Architektur des gesamten Regelkreises	48
4.3. Analogie zwischen neuronalem Netz und Interpolationsmethode	50
4.4. Die Fehlerberechnung	51
5. Test und Simulation.....	54
5.1. Test der adaptiven Linearisierung	54
5.2. Test der approximativen Linearisierungsmethode mittels neuronaler Netze	58
5.2.1. Generalisierung.....	58
5.2.2. Cross Validation.....	59
5.2.3. Regulierung der frühzeitigen Einstellung des Trainings	60
5.2.4. Auswahl der neuronalen Netze	61
5.2.5. Test.....	62
5.2.5.1. Analytische Berechnung und Simulation.....	62
5.2.5.2. Allgemeiner Simulationsalgorithmus	64
5.2.5.3. Approximation mit Hilfe neuronaler Netze	67
5.2.5.4. Approximation mit Hilfe der Interpolation	70
5.2.5.5. Graphische Darstellung der Fehlerebenen.....	74
5.2.5.5.1. Neuronale Netze	74
5.2.5.5.2. Interpolation	76

6. Literaturverzeichnis	78
-------------------------------	----

Einleitung

Die linearen oder nichtlinearen Systeme werden meistens mit Hilfe einer Differentialgleichung präsentiert.

Eine lineare Differentialgleichung ist eine Differentialgleichung, die aus der Summe der linearen Terme besteht, denn sonst ist es keine lineare Differentialgleichung.

Das Konzept der Nichtlinearität basiert auf der Eigenschaft der Differentialgleichungsklasse. Für ein zeitunabhängiges System kann die Linearität mit Hilfe des Superpositionsgrundsatzes untersucht werden. Das bedeutet, ein System heißt nichtlinear, wenn der Superpositionsgrundsatz für das System nicht gilt. Ein lineares System ist ein System mit den folgenden Eigenschaften, die u.a. sind:

ein Input $x_1(t)$ erzeugt ein Output $y_1(t)$

ein Input $x_2(t)$ erzeugt ein Output $y_2(t)$

ein Input $c_1 x_1(t) + c_2 x_2(t)$ erzeugt ein Output $c_1 y_1(t) + c_2 y_2(t)$. Dies gilt für alle Paare des Inputs $x_1(t)$ und $x_2(t)$ und für alle Paare der Konstanten c_1 und c_2 .

Ein Regelsystem oder Regelkreis besteht prinzipiell aus zwei Komponenten, aus einer Strecke und einem Regler.

Die inneren und äußeren Verbindungen des Regelkreises werden mit Hilfe von Signalen wie der Regelgröße (als Istwert), Führungsgröße (als Sollwert), Regelabweichung, Stellgröße, und Störgröße bestimmt.

Die Regelkreisglieder können mit Differentialgleichungen beschrieben werden.

Das Verhalten eines Regelsystems kann auch anhand der Zustandsraumdarstellung beschrieben werden, wobei mit Hilfe der Zustandsraumdarstellung die Vektorschreibweise der Zustandsgrößen möglich sind.

Das gesamte Verhalten dieses Regelkreises wird von den verschiedenen Signalgrößen des Regelkreises, dem Regler und der Strecke beeinflusst, wodurch schließlich eine Wechselwirkung entsteht. Diese Einflüsse und Wechselwirkungen führen die Systemzustände zu instabilen-, stabilen- oder Grenzstabilenverhalten.

Die Zustände eines Systems besitzen stabile Eigenschaften, wenn für einen begrenzten Sollwert die Systemzustände eine begrenzte Größe zeigen.

Diese Diplomarbeit befasst sich hauptsächlich mit der Linearisierung von nichtlinearen Strecken mit Hilfe approximativer Methoden.

Außerdem wurden danach auch einige lineare Lösungsansätze, die durch das Vorhandensein der Linearisierung von nichtlinearen Strecken überhaupt erst möglich wurden, ebenfalls dargelegt.

Im ersten Kapitel wird die exakte Linearisierung erklärt. Linearisierung wird durch die Eliminierung der nichtlinearen Terme möglich. Nach der Linearisierung wird eine Zustandsregelung für das linearisierte System beschrieben. Beim Regeldesign ist eine geeignete Auswahl der Pollstellen unabdingbar. Diese Pollstellen gewährleisten eine Stabilität an den Ausgängen. Das System selbst muss aber auch über stabile innere Zustände verfügen, damit eine allgemeine Stabilität erreicht wird. In diesem Kapitel wird der Begriff „Tracking“ erklärt.

Im zweiten Kapitel werden die Zusammenhänge der adaptiven Zustandslinearisierung erläutert. Hier werden die Parameter der Strecke adaptiert und die entsprechenden Linearisierungen durchgeführt. Im dritten Kapitel werden die wichtigsten neuronalen Netze vorgestellt. Die neuronalen Netze erfordern ein eigenes Forschungskapitel.

Im vierten Kapitel wird die approximative Methode erklärt, hier wird speziell auf die Identifikation und die Linearisierung eingegangen. Die Approximierungseinheiten sind die neuronalen Netze und Interpolationseinheiten.

Kapitel fünf behandelt den praktischen Teil der Diplomarbeit. Die durchgeführten Tests und Simulationen werden beschrieben, Die Blockschaltbilder der Regelkreise und die Ergebnisse werden dargestellt. Test und Simulation sind unter Matlab und Simulink durchgeführt worden.

1.Exakte Zustandslinearisierung

In diesem Kapitel wird das Linearisierungsthema beschrieben. Die in der Diplomarbeit angewendete approximative Methode basiert auf exakter Linearisierung. Zur exakten Linearisierung wird die Lie-Ableitung verwendet, die bei der Vorgehensweise dieser Arbeit bedeutsam ist. Zustandslinearisierung ist ein Weg zur Umwandlung einer nichtlinearen Zustandsraumdarstellung in einer einfachen Form, wodurch nicht lineare Terme weggekürzt werden - der geschlossene Regelkreis - die Ein- und Ausgänge verhalten sich somit linear (siehe [Schwa 91], [SASTR 89], [He 98], [Marin 95], [Jacqu 91]). Hier wird zunächst die Zustandsraumdarstellung erklärt. Die Dynamik eines Systems kann durch die Zustandsraumdarstellung beschrieben werden. Die Zustandsraumdarstellung für ein lineares System wird allgemein mit Hilfe der Formel (1-1) und (1-2) beschrieben:

$$\dot{x} = Ax + Bu \quad (1-1)$$

$$y = Cx + Du \quad (1-2)$$

\dot{x} ist die Zustandsdifferentialgleichung, wobei A die Systemmatrix ist, B die Steuermatrix, C die Beobachtungsmatrix und D die Durchgangsmatrix. x ist der Zustandsvektor, y der Istwert oder Ausgangsvektor. Für ein nichtlineares System ohne Durchgangsfeld D lässt sich die Differentialgleichung in folgender funktionaler Form angeben:

$$\dot{x} = f(x) + g(x)u \quad (1-3)$$

$$y = h(x) \quad (1-4)$$

Im Fall einer Streckenänderung und bei funktionalen Unsicherheiten kann das nichtlineare System gemäß den Formeln (1-5) und (1-6) modelliert werden.

$$\dot{x} = f(x) + \Delta f(x) + (g(x) + \Delta g(x))u \quad (1-5)$$

$$y = h(x) \quad (1-6)$$

$\Delta f(x)$ wird als funktionale Unsicherheit der Systemmatrix bezeichnet, während $\Delta g(x)$ die funktionale Unsicherheit der Steuereinheit darstellt. Für ein System mit mehreren Ein- und Ausgängen wird die Zustandsraumdarstellung um die zusätzlichen Ein- und Ausgänge erweitert, wobei der Systeminput affine Eigenschaften besitzen müssen. Die Ermittlung einer formalen Lösung (Linearisierung mit Controller, eventuell inkludiertes Tracking) für ein solches nichtlineare MIMO ist weitaus komplizierter als etwa ein SISO System. Die Zustandsraumdarstellung wird allgemein mit Hilfe der Formeln (1-7) und (1-8) angegeben.

$$\dot{x}(t) = f(x, u, t) \quad (1-7)$$

$$y = h(x, u, t) \quad (1-8)$$

1.1. Linearisierung

Wenn nichtlineare Strecken ein autonomes Verhalten besitzen, und der dynamische Charakter zeitunabhängig ist, dann kann für den Istwert y unter Berücksichtigung der Zeit folgende Differentialgleichung angegeben werden.

$$\dot{y} = \frac{\partial h}{\partial x} \cdot \frac{\partial x}{\partial t} = \frac{\partial h}{\partial x} \cdot \dot{x} \quad (1-9)$$

Wie man sieht, setzt sich die Formel (1-9) aus zwei Anteilen zusammen. Der zeitunabhängige Term $\frac{\partial h}{\partial x}$ könnte als physikalischer Anteil (zum Beispiel raumabhängiger Anteil) interpretiert werden. Zusammen mit dem zeitabhängigen Term \dot{x} gibt diese Gleichung die Zustandsänderung im Zeitbereich an. Setzt man (1-3) in (1-9) ein, so ergibt sich für ein SISO System folgende Differentialgleichung erster Ordnung:

$$\dot{y} = \frac{\partial h}{\partial x} \dot{x} = \frac{\partial h}{\partial x} f(x) + \frac{\partial h}{\partial x} g(x)u \quad (1-10)$$

Für die Erfüllbarkeit der Gleichung (1-10) wird für $h(x)$ unendliche Differenzierbarkeit vorausgesetzt. Wenn man eine entsprechende gleichbedeutende Definition der Lie-Ableitung in (1-10) einsetzt, führt dies zu folgendem Ergebnis:

$$\dot{y} = L_f h(x) + L_g h(x)u \quad (1-11)$$

Um den nicht linearen Anteil der Strecke zu eliminieren, definiert man eine Stellgröße mittels folgendem Regelgesetz:

$$u = \frac{1}{L_g h(x)} (-L_f h(x) + v) \quad (1-12)$$

In (1-12) ist u die Stellgröße der nicht linearen Strecke, v ist der neue Eingang oder die Sollgröße für das linearisierte System. Die Multiplikation zwischen $\frac{\partial h(x)}{\partial x}$ und $f(x)$ oder $\frac{\partial h(x)}{\partial x}$ und $g(x)$ kann als inneres Produkt gesehen werden, somit sind die Terme $L_f h(x)$ und $L_g h(x)$ Skalare. $L_f h(x)$ und $L_g h(x)$ können als Abbildung (oder Mapping) der Richtungsableitung der Funktion $h(x)$ entlang der Vektoren $f(x)$ oder $g(x)$ betrachtet werden.

Die Abbildungen der Ableitungen $h(x)$ nach x bei den beiden Vektorfunktionen $f(x)$ und $g(x)$ sind konkret in den Formeln (1-13) und (1-14) angegeben.

$$L_f h(x) = \frac{\partial h(x)}{\partial x} \cdot f(x) \quad (1-13)$$

$$L_g h(x) = \frac{\partial h(x)}{\partial x} \cdot g(x) \quad (1-14)$$

Das $L_g h(x)$ muss ungleich Null sein, damit u in der Gleichung (1-12) keinen unbestimmten Wert annimmt. Wenn man (1-12) in (1-11) einsetzt, erhält man:

$$\dot{y} = L_f h(x) + (L_g h(x)) \left(\frac{1}{L_g h(x)} (-L_f h(x) + v) \right) \quad (1-15)$$

Nach vereinfachenden Umformungen wird \dot{y} zu:

$$\dot{y} = v \quad (1-16)$$

Wie die Gleichung (1-16) zeigt, entspricht die Istwertänderung gleich dem Sollwert, der nichtlineare Term ist verschwunden. Mit Hilfe der Gleichungen (1-9) bis (1-16) werden die ursprünglichen zeitlichen und physikalischen Komponenten durch die Größe v ersetzt.

1.2. Koordinatentransformation

In diesem Kapitel wird die Koordinatentransformation für nichtlineare Systeme mit so genanntem „starkem relativem Grad“ (engl.: strong relative degree) beschrieben.

1.2.1. Nulldynamik und Unbeobachtbarkeit

In diesem Abschnitt wird der Begriff der Nulldynamik erklärt. Die Nulldynamik ist als die innere Dynamik eines Systems definiert, wenn der Systemoutput mit Hilfe des Input gleich Null ist. Nicht lineare Systeme, deren Nulldynamik asymptotisch stabil sind, bezeichnet man als Systeme mit asymptotischer Minimumphase. Bei der Nulldynamik geht man davon aus, dass der Istwert mit Hilfe des Sollwertes zu Null wird. Wird v gleich Null gesetzt, wird die Gleichung (1-12) zu (1-17):

$$u = \frac{1}{L_g h(x)} (-L_f h(x)) \quad (1-17)$$

Setzt man (1-17) in (1-11) ein, so wird die Ableitung des Ausgangs y nach einer gewissen Zeitspanne zu Null.

$$\dot{y} = L_f h(x) + (L_g h(x)) \left(\frac{1}{L_g h(x)} (-L_f h(x)) \right) \quad (1-18)$$

$$\dot{y} = 0 \quad (1-19)$$

Im Fall der Formel (1-19) ist die Dynamik des Systems unbeobachtbar geworden. Für Systeme höherer Ordnung gilt die Darstellung gemäß Gleichung (1-20):

$$h(x) = 0 \quad (1-20)$$

$$L_f h(x) = 0$$

$$L_f^2 h(x) = 0$$

.

.

.

$$L_f^{n-2} h(x) = 0$$

$$L_f^{n-1} h(x) = 0$$

Ein lineares System ist ein Minimum Phasen System, wenn keine Nullstellen und Pollstellen in der rechten s-Halbebene liegen. Für nichtlineare Systeme ist die Minimum Phase anders definiert. Die nichtlinearen Systeme können als Minimum Phase bezeichnet werden, wenn die Nulldynamik asymptotisch stabil ist. Damit ein nicht lineares System stabil ist, muss der innere Zustand stabil sein.

1.2.2. Starker relativer Grad

Für lineare Systeme wird der relative Grad rg meistens als Differenz zwischen dem Grad des Nennerpolynoms np und dem Grad des Zählerpolynoms zp einer Übertragungsfunktion verstanden. D.h., der relative Grad ist gleich: $rg = np - zp$. Der starke relative Grad wird für nichtlineare Systeme in den Gleichungen (1-3) und (1-4) so definiert, dass der Istwert y mittels der Lie-Ableitung n mal differenziert wird, bis die Stellgröße u oder zusammenhängende Gleichungen von u in der differenzierten Gleichung des Istwertes y erscheint. Wenn im Rahmen der Differentiation die Stellgröße u nicht erreicht werden kann, dann ist das System unkontrollierbar.

Für den relativen Grad n gilt:

$$L_g h(x) = 0 \quad (1-21)$$

$$L_g L_f h(x) = 0$$

$$L_g L_f^2 h(x) = 0$$

•

•

•

$$L_g L_f^{n-2} h(x) = 0$$

$$L_g L_f^{n-1} h(x) \neq 0$$

Sobald $L_g L_f^{n-1} h(x)$ ungleich Null ist, dann ist die Stellgröße u erreicht, und eine Linearisierung ist möglich.

1.2.3. Koordinatentransformation

Ziel von Koordinatentransformationen ist es, Stabilitätsprobleme nichtlinearer Systeme zu vereinfachen. Ganz allgemein versteht man unter einer Koordinatentransformation den Übergang von einem Koordinatensystem zu einem anderen Koordinatensystem unter bestimmten Regeln und Voraussetzungen. Eine Koordinatentransformation soll bestimmte Eigenschaften besitzen, es wird Differenzierbarkeit der Abbildungsfunktion sowie die Rücktransformierbarkeit gefordert. Koordinatentransformationen können im allgemeinen entweder für den Bereich eines Arbeitspunktes, oder für einen globalen Bereich definiert werden. Für nicht lineare Systeme ist es schwierig eine globale Koordinatentransformation durchzuführen. Mit der Brunovsky Normal Form und Anwendung der Lie-Ableitung kann eine Koordinatentransformation in einer neuen Zustandsraumdarstellung beschrieben werden. So wird die nichtlineare Differentialgleichung in einem verketteten linearen Integralsystem abgebildet. Nach Definition von [Schwa 91], können transformierte Zustände gefunden werden, wenn der relative Grad rg kleiner als die Anzahl der Zustände az ist. Es ist dann möglich weitere $az - rg$ transformierte Zustände zu finden. Die zugehörige Jakobianmatrix der neu gefundenen Zustände muss singulär sein. Im folgendem werden für einen starken relativen Grad die neuen Zustände mit z bezeichnet.:

Für den Istwert gilt:

$$z_1 = h(x) = y \quad (1-22)$$

Für weitere Zustände gelten die Formeln (1-23) bis (1-33):

$$z_2 = \dot{z}_1 = \frac{\partial z_1}{\partial t} = \frac{\partial z_1}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial h}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial h}{\partial x} \dot{x} = \frac{\partial h}{\partial x} (f(x) + g(x)u) \quad (1-23)$$

Wenn man annimmt, dass für die primäre Ableitungen von z_1 , u gleich Null ist, dann entfällt in Formel (1-23) der u Term.

$$z_2 = \dot{z}_1 = \frac{\partial h}{\partial x} (f(x)) = L_f^1 h(x) \quad (1-24)$$

Die Vorgehensweise für weitere Zustände ist gleich, wenn u wiederum gleich Null ist, für den nächsten Zustand ergibt sich,:

$$z_3 = \dot{z}_2 = \frac{\partial z_2}{\partial t} = \frac{\partial z_2}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial (L_f^1 h(x))}{\partial x} \dot{x} = \frac{\partial (L_f^1 h(x))}{\partial x} (f(x) + g(x)u) = \frac{\partial (L_f^1 h(x))}{\partial x} f(x) \quad (1-25)$$

$$z_3 = \dot{z}_2 = \frac{\partial (L_f^1 h(x))}{\partial x} f(x) = \frac{\partial \left(\frac{\partial h}{\partial x} (f(x)) \right)}{\partial x} f(x) = L_f (L_f^1 h(x)) = L_f^2 h(x) \quad (1-26)$$

Für z_4 wird:

$$z_4 = \dot{z}_3 = \frac{\partial z_3}{\partial t} = \frac{\partial z_3}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial (L_f^2 h(x))}{\partial x} \dot{x} = \frac{\partial (L_f^2 h(x))}{\partial x} (f(x) + g(x)u) = \frac{\partial (L_f^2 h(x))}{\partial x} f(x) \quad (1-27)$$

$$z_4 = \dot{z}_3 = \frac{\partial (L_f^2 h(x))}{\partial x} f(x) = L_f (L_f (L_f^1 h(x))) = L_f^3 h(x) \quad (1-28)$$

In gleicher Art und Weise wird z_n zu:

$$z_n = \dot{z}_{n-1} = \frac{\partial z_{n-1}}{\partial t} = \frac{\partial z_{n-1}}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial (L_f^{n-2} h(x))}{\partial x} \dot{x} = \frac{\partial (L_f^{n-2} h(x))}{\partial x} (f(x) + g(x)u) = \frac{\partial (L_f^{n-2} h(x))}{\partial x} f(x) \quad (1-29)$$

$$z_n = \dot{z}_{n-1} = \frac{\partial (L_f^{n-2} h(x))}{\partial x} f(x) = L_f (\dots (L_f h(x))) = L_f^{n-1} h(x) \quad (1-30)$$

Diese Vorgehensweise wird bei einem starken relativen Grad solange durchgeführt, bis die Stellgröße in der Gleichung erscheint. D.h., jetzt wird $u \neq 0$, und die entsprechende Gleichung wird zu:

$$z_{n+1} = \dot{z}_n = \frac{\partial z_n}{\partial t} = \frac{\partial z_n}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial(L_f^{n-1}h(x))}{\partial x} \dot{x} = \frac{\partial(L_f^{n-1}h(x))}{\partial x} (f(x) + g(x)u) \quad (1-31)$$

$$z_{n+1} = \dot{z}_n = \frac{\partial(L_f^{n-1}h(x))}{\partial x} f(x) + \frac{\partial(L_f^{n-1}h(x))}{\partial x} g(x)u = L_f(L_f \cdots (L_f h(x))) + L_g(L_f \cdots (L_f h(x)))u \quad (1-32)$$

In der Kurzform wird (1-32) zu (1-33):

$$z_{n+1} = \dot{z}_n = L_f^n h(x) + L_g L_f^{n-1} h(x)u \quad (1-33)$$

Werden die Formeln (1-22) bis (1-33) zusammengefasst, ergibt dies:

$$\begin{pmatrix} \dot{z}_1 \\ \vdots \\ \dot{z}_{n-1} \\ \dot{z}_n \end{pmatrix} = \begin{pmatrix} L_f^1 h(x) \\ \vdots \\ L_f^{n-1} h(x) \\ L_f^n h(x) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ L_g L_f^{n-1} h(x) \end{pmatrix} u = \begin{pmatrix} z_2 \\ \vdots \\ z_n \\ L_f^n h(x) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ L_g L_f^{n-1} h(x) \end{pmatrix} u \quad (1-34)$$

Nach zwei Änderungen können die Gleichungen (1-34) in der Brunovsky Normal Form angegeben werden. Zuerst werden die nichtlinearen Terme durch die geeignete Wahl von u eliminiert. (siehe Formel (1-35)). Dann kann ein neuer Eingang v definiert werden. Die Linearisierungsgleichung mit starkem relativen Grad sieht folgendermaßen aus:

$$u = \frac{1}{L_g L_f^{n-1} h(x)} (-L_f^n h(x) + v) \quad (1-35)$$

Setzt man (1-35) in (1-34) ein, dann lautet das Ergebnis:

$$\begin{pmatrix} \dot{z}_1 \\ \vdots \\ \dot{z}_{n-1} \\ \dot{z}_n \end{pmatrix} = \begin{pmatrix} z_2 \\ \vdots \\ z_n \\ L_f^n h(x) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ L_g L_f^{n-1} h(x) \end{pmatrix} \frac{1}{L_g L_f^{n-1} h(x)} (-L_f^n h(x) + v) \quad (1-36)$$

Nach Kürzungen ergibt sich eine lineare Zustandsraumdarstellung:

$$\begin{pmatrix} \dot{z}_1 \\ \vdots \\ \dot{z}_{n-1} \\ \dot{z}_n \end{pmatrix} = \begin{pmatrix} z_2 \\ \vdots \\ z_n \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} v \quad (1-37)$$

Die Gleichung (1-37) ist graphisch folgendermaßen dargestellt:

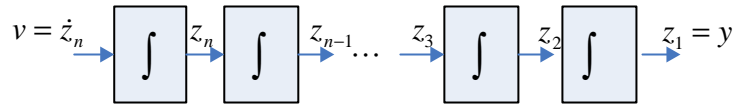


Abbildung 1:1: Koordinatentransformation.

Man kann ein entsprechendes Zustandsmodell zur Beschreibung der Gleichung (1-37) finden. Um ein Modell in Brunovsky Normal zu erhalten, wird Systemmatrix A definiert. A dient als Verkettung der Integraleinheiten.

$$A = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 0 \end{pmatrix} \quad (1-38)$$

Matrix B wird zur Bestimmung des neuen Ausganges definiert.

$$B = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \quad (1-39)$$

Die gesamte lineare Zustandsraumdarstellung wird in der Formel (1-40) beschrieben:

$$\dot{z} = Az + Bv \quad (1-40)$$

Davon ist z der Zustandsvektor. Ausführlich lautet Gleichung (1-40) wie folgt:

$$\begin{pmatrix} \dot{z}_1 \\ \vdots \\ \dot{z}_{n-1} \\ \dot{z}_n \end{pmatrix} = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 0 \end{pmatrix} \begin{pmatrix} z_2 \\ \vdots \\ z_n \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} v \quad (1-41)$$

Man kann z_2 bis z_n entsprechend der Formel (1-23) bis (1-30) folgendermaßen darstellen:

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} h(x) \\ L_f^1 h(x) \\ L_f^2 h(x) \\ \vdots \\ L_f^{n-1} h(x) \end{pmatrix} \quad (1-42)$$

Wie (1-41) zeigt, kann der letzte Zustand \dot{z}_n gleich v gesetzt werden, die übrigen Zustandsvariablen können aus Formel (1-42) abgelesen werden. Die Zustände haben ihre eigene Dynamik und Eigenschaften, um Stabilität zu gewährleisten, muss man zusätzlich einen Regler entwickeln.

1.3. Reglerdesign

Man kann einen Zustandsregler auf Basis der linearen Modellierung in der Brunovsky Normal Form bilden. Daher muss man wieder einen neuen Term definieren, der den neuen Eingang v_{new} und einen proportionalen Regler enthält. Der alte Eingang wird durch diese beiden neuen Elemente ersetzt. In der Gleichung (1-43) sieht man den neuen Term inklusive mit neuem Eingang und Regler. k ist der Zustandsregler, z der Zustandsvektor, v ist die alte Stellgröße. Das Ziel ist hier die Zustandsrückführung von z mit Hilfe eines proportionalen Reglers zum Ziel der Regulierung des linearen Systems.

$$v = k^T z + v_{new} \quad (1-43)$$

Diese Art des Reglerdesigns wird ausführlich in weiterführender Literatur [Weinm 95] über lineare System beschrieben. Wenn man (1-43) in (1-40) einsetzt, erhält man folgende Formel:


$$\dot{z} = Az + Bv = Az + B(k^T z + v_{new}) = Az + Bk^T z + Bv_{new} \quad (1-44)$$

Für ein linearisiertes System darf man zur Lösung der Differentialgleichung die Laplace Transformation verwenden, d.h. (1-44) wird zu:

$$Sz = Az + Bk^T z + Bv_{new} \quad (1-45)$$

$$(SI - A - Bk^T)z = Bv_{new} \quad (1-46)$$

$$z = (SI - A - Bk^T)^{-1} Bv_{new} \quad (1-47)$$

Die Führungspollstellen werden gewählt, indem der entsprechende Zustandsregler kalkuliert wird. 

Die Determinante von $(SI - A - Bk^T)$ wird zur Kalkulation des Zustandsreglers ermittelt, wenn sie den Pollstellen gleich gesetzt wird, so wird der Zustandsregler \mathbf{K} berechnet. (1-43) in (1-35) eingesetzt, ergibt ein Regelgesetz mit Zustandsregler gemäß der Formel (1-48):

$$u = \frac{1}{L_g L_f^{n-1} h(x)} \left(-L_f^n h(x) + k^T z + v_{new} \right) \quad (1-48)$$

Die alte Stellgröße v wird ausführlich beschrieben durch:

$$v = k^T z + v_{new} = \begin{pmatrix} k_1 & k_2 & \dots & k_n \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} + v_{new} \quad (1-49)$$

Setzt man (1-42) in (1-49), dann erhält man:

$$v = \begin{pmatrix} k_1 & k_2 & k_3 & \dots & k_n \end{pmatrix} \begin{pmatrix} h(x) \\ L_f^1 h(x) \\ L_f^2 h(x) \\ \vdots \\ L_f^{n-1} h(x) \end{pmatrix} + v_{new} \quad (1-50)$$

(1-50) vereinfacht sich in (1-51) zu:

$$v = k_1 h(x) + k_2 L_f^1 h(x) + k_3 L_f^2 h(x) + \dots + k_n L_f^{n-1} h(x) + v_{new} \quad (1-51)$$

Jetzt setzt man (1-51) in (1-48) ein, und dadurch erhält man ein Regelgesetz inklusive Zustandsregler:

$$u = \frac{1}{L_g L_f^{n-1} h(x)} \left(-L_f^n h(x) + k_1 h(x) + k_2 L_f^1 h(x) + k_3 L_f^2 h(x) + \dots + k_n L_f^{n-1} h(x) + v_{new} \right) \quad (1-52)$$

Die Gleichung (1-52) in Summenform lautet:

$$u = \frac{1}{L_g L_f^{n-1} h(x)} \left(-L_f^n h(x) + \sum_{i=1}^{i=n} k_i L_f^{i-1} h(x) + v_{new} \right) \quad (1-53)$$

Formel (1-53) kann auch in Form von (1-54) auftreten. Mit der richtigen Auswahl von k und der passenden Polstelleneinstellung wird das Vorzeichen der Summe $\sum_{i=1}^{i=n} k_i L_f^{i-1} h(x)$ beeinflusst.

$$u = \frac{1}{L_g L_f^{n-1} h(x)} \left(-L_f^n h(x) - \sum_{i=1}^{i=n} k_i L_f^{i-1} h(x) + v_{new} \right) \quad (1-54)$$

1.4. Tracking

Das Ziel des Tracking ist es, einerseits eine Zustandsstabilisierung herzustellen, andererseits den Systemausgang in Richtung des gewünschten Ausgangs zu steuern. Ein Trackingmodell eignet sich für einen Regelkreis mit Rückführung. D.h., um die Regelgröße zu kontrollieren, benötigt man den Zugriff auf Zustandsgröße. Besonders wichtig wird Tracking im Zusammenhang mit der Nulldynamik und der Minimum Phase. Das Hauptziel des Tracking ist es zu erreichen, dass die Regelgröße dem Sollwert (oder Referenzsignal) entspricht. D.h., der Limes der Differenz zwischen den Ist-Zuständen und dem entsprechenden Referenzsignal gleichen Grades soll gegen Null konvergieren.

$$\lim_{t \rightarrow \infty} (v_{ref} - y) = \lim_{t \rightarrow \infty} (v_{ref} - h(x)) = 0 \quad (1-55)$$

Das Referenzsignal muss ableitbar sein, es muss bestimmte Eigenschaften (wie glatt und gebunden) besitzen. Die Differenz oder der Error für ein System mit hohem relativem Grad ist der Unterschied zwischen den rückführenden Größen und den vorhandenen Referenzsignalen (1-56). Formel (1-56) enthält folgende Größen, die Referenz und deren Ableitungen von Grad eins bis Grad $n-1$, die Zustandgrößen des transformierten Koordinatensystems verschiedener Istwerte mit unterschiedlichem Grad, von Null bis zu Grad $n-1$.

$$\begin{pmatrix} e \\ \dot{e} \\ \vdots \\ e^{n-1} \end{pmatrix} = \begin{pmatrix} v_{ref} - h(x) \\ \dot{v}_{ref} - L_f^1 h(x) \\ \vdots \\ v_{ref}^{n-1} - L_f^{n-1} h(x) \end{pmatrix} \quad (1-56)$$

Hier wird ein neuer Regelkontext nach (1-43) unter Beachtung des Tracking-Errors beschrieben:

$$v_{alt} = \begin{pmatrix} k_1 & k_2 & k_3 & \dots & k_n \end{pmatrix} \begin{pmatrix} e \\ \dot{e} \\ \vdots \\ e^{n-2} \\ e^{n-1} \end{pmatrix} + v_{ref}^n \quad (1-57)$$

Ausgehend von der Gleichung (1-57) ergibt sich:

$$v_{alt} = k_1 e + k_2 \dot{e} + k_3 \ddot{e} + \dots + k_n e^{n-1} + v_{ref}^n \quad (1-58)$$

Oder gleichwertig lautet der Ausdruck in (1-59):

$$v_{alt} = k_1 (v_{ref} - h(x)) + k_2 (\dot{v}_{ref} - L_f^1 h(x)) + k_3 (\ddot{v}_{ref} - L_f^2 h(x)) + \dots + k_n (v_{ref}^{n-1} - L_f^{n-1} h(x)) + v_{ref}^n \quad (1-59)$$

Schließlich kann das Regelgesetz mit Tracking für nichtlineare Systeme so aussehen:

$$u = \frac{1}{L_g L_f^{n-1} h(x)} (v_{ref}^n - L_f^n h(x) + k_1 (v_{ref} - h(x)) + k_2 (\dot{v}_{ref} - L_f^1 h(x)) + \dots + k_n (v_{ref}^{n-1} - L_f^{n-1} h(x))) \quad (1-60)$$

Bei der Wahl von k_1, k_2, \dots, k_n muss das Polynom $k_1 (v_{ref} - h(x)) + k_2 (\dot{v}_{ref} - L_f^1 h(x)) + \dots + k_n (v_{ref}^{n-1} - L_f^{n-1} h(x))$ ein Hurwitzpolynom sein.

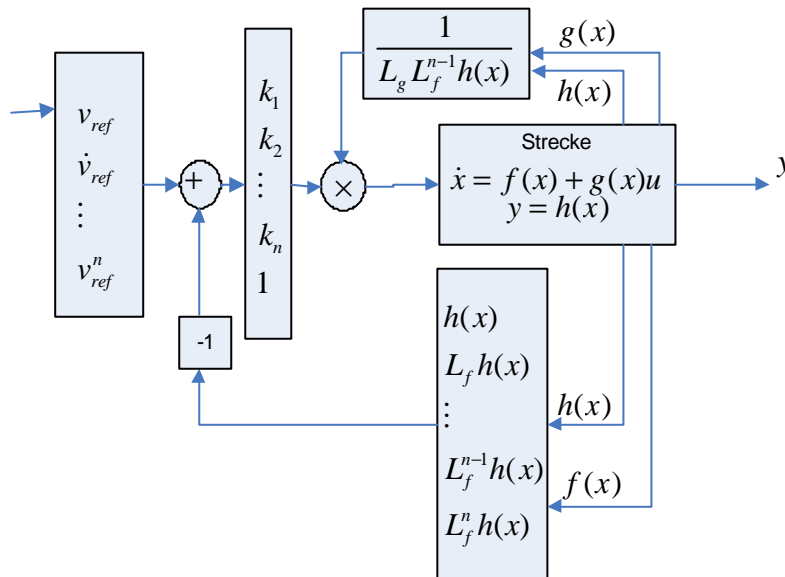


Abbildung 1:2: Exakte Linearisierung und Tracking.

2. Adaptive exakte Linearisierung

Die exakte Linearisierung eines SISO mit relativem Grad eins wird durch die Wegnahme der nichtlinearen Terme und mit Hilfe eines Regelgesetzes realisiert. Der Unterschied liegt in der Parameteradaption. Wenn man die dargestellten Systeme in (1-3) und (1-4) betrachtet, kann man davon ausgehen, dass $f(x)$ und $g(x)$ mit Unsicherheiten behaftet sind. Unter diesen Voraussetzungen kann die Linearisierung nur ungenau angesetzt werden. In diesem Abschnitt wird daher eine Parameteradaption für das linearisierte System durchgeführt. (siehe [SASTR 89])

2.1. Adaptionssysteme

Vorausgesetzt wird, dass der Ausdruck $L_g h(x)$ nach Gleichung (1-12) einen Wert ungleich Null annimmt.

Für $f(x)$ und $g(x)$ kann man zwei unterschiedliche Mengen von unbestimmten Parametern $P_i^{(1)*}$ und $P_j^{(2)*}$ annehmen. Parameter $P_i^{(1)*}$ mit Index $i = 1, 2, 3, \dots, n_1$ und Parameter $P_j^{(2)*}$ mit Index $j = 1, 2, 3, \dots, n_2$ werden definiert. Die zwei Mengen in einer Zustandsraumdarstellung lauten:

$$\dot{x} = P_1^{(1)*} f_1(x) + P_2^{(1)*} f_2(x) + \dots + P_{n_1}^{(1)*} f_{n_1}(x) + P_1^{(2)*} g_1(x)u_1 + P_2^{(2)*} g_2(x)u_2 + \dots + P_{n_2}^{(2)*} g_{n_2}(x)u_{n_2} \quad (2-1)$$

$$y = h(x) \quad (2-2)$$

D.h., $f(x)$ und $g(x)$ inklusive Parametern können in folgenden Formen dargestellt werden:

$$f(x) = \sum_{i=1}^{n_1} P_i^{(1)*} f_i(x) \quad (2-3)$$

$$g(x) = \sum_{j=1}^{n_2} P_j^{(2)*} g_j(x) \quad (2-4)$$

Wenn eine Abschätzung von $P_i^{(1)*}$ und $P_j^{(2)*}$ möglich ist, dann werden die abgeschätzten $f(x)$ und $g(x)$ mit dem Index e gekennzeichnet. Die beiden ähnlichen Gleichungen werden zu (2-3) und (2-4) lauten nach (2-5) und (2-6):

$$f_e(x) = \sum_{i=1}^{n_1} P_i^{(1)} f_i(x) \quad (2-5)$$

$$g_e(x) = \sum_{j=1}^{n_2} P_j^{(2)} g_j(x) \quad (2-6)$$

P_i und P_j sind die Abschätzungen von $P_i^{(1)*}$ und $P_j^{(2)*}$.

Das so entsprechend abgeschätzte Linearisierungsregelgesetz ist gemäß (1-12):

$$u = \frac{1}{(L_g h(x))_e} \left(- (L_f h(x))_e + v \right) \quad (2-7)$$

$(L_g h(x))_e$ und $(L_f h(x))_e$ sind die Abschätzungen von $(L_g h(x))$ und $(L_f h(x))$.

Die zwei Terme $(L_g h(x))_e$ und $(L_f h(x))_e$ lauten:

$$(L_f h)_e = \sum_{i=1}^{n_1} P_i^{(1)}(t) L_{f_i} h(x) \quad (2-8)$$

$$(L_g h)_e = \sum_{j=1}^{n_2} P_{(j)}^2(t) L_{g_j} h(x) \quad (2-9)$$

Die Aktualisierung des Parametergradientengesetzes, \hat{f} , wird nach Formel (2-10) berechnet. Dazu braucht die Berechnung des Fehlers e_0 .

$$\hat{f} = -e_0 w \quad (2-10)$$

w besteht aus zwei Anteilen, $w^{(1)}$ und $w^{(2)}$ (siehe [SASTR 89]). Diese zwei Anteile sind eigentlich zwei Gruppen von Lie-Ableitungen:

$$w = \begin{pmatrix} w^{(1)} \\ w^{(2)} \end{pmatrix} = \begin{pmatrix} - \begin{pmatrix} L_{f_1} h(x) \\ \vdots \\ L_{f_{n_1}} h(x) \end{pmatrix} \\ \begin{pmatrix} L_{g_1} h(x) \\ \vdots \\ L_{g_{n_2}} h(x) \end{pmatrix} \frac{((L_f h(x))_e - v)}{(L_g h(x))_e} \end{pmatrix} \quad (2-11)$$

Der Tracking Error e_0 ist:

$$e_0 = y - y_{ref} \quad (2-12)$$

Für Tracking Error kann man nach [SASTR 89] eine Differentialgleichung definieren. $y(t)$ soll gegen $y_{ref}(t)$ konvergieren, sodass e_0 gegen Null strebt. Der Koeffizient k beeinflusst das Konvergenzverhalten des Tracking Errors.

$$\dot{e}_0 + k e_0 = \mathbf{f}^T \mathbf{w} \quad (2-13)$$

Wenn (2-12) in (2-13) eingesetzt wird, dann erhält man folgende Trackinggleichung:

$$\dot{y} - \dot{y}_{ref} + k(y - y_{ref}) = \mathbf{f}^T \mathbf{w} \quad (2-14)$$

Unter der Annahme, dass die Fehlergleichung linear ist!, wird (2-13) Laplace transformiert, als Lösung erhält man:

$$S e_0 + k e_0 = \mathbf{f}^T \mathbf{w} \quad (2-15)$$

Weiteres folgt:

$$e_0(S + k) = \mathbf{f}^T \mathbf{w} \quad (2-16)$$

und:

$$e_0 = \frac{\mathbf{f}^T \mathbf{w}}{S + k} \quad (2-17)$$

Es ergibt sich für den Parameterfehler die Formel (2-18):

$$\mathbf{f} = \mathbf{P} - \mathbf{P}^* = \begin{pmatrix} P^{(1)} \\ P^{(2)} \end{pmatrix} - \begin{pmatrix} P^{(1)*} \\ P^{(2)*} \end{pmatrix} = \begin{pmatrix} P_1^{(1)} \\ P_2^{(1)} \\ \vdots \\ P_{n1}^{(1)} \\ P_1^{(2)} \\ P_2^{(2)} \\ \vdots \\ P_{n2}^{(2)} \end{pmatrix} - \begin{pmatrix} P_1^{(1)*} \\ P_2^{(1)*} \\ \vdots \\ P_{n1}^{(1)*} \\ P_1^{(2)*} \\ P_2^{(2)*} \\ \vdots \\ P_{n2}^{(2)*} \end{pmatrix} \quad (2-18)$$

Der Parameterfehler ist die Differenz zwischen abgeschätzten und realen Parametern. \mathbf{P}^* wird als konstant betrachtet, die Ableitung von (2-18) ergibt (2-19):

$$\dot{\mathbf{f}} = \dot{\mathbf{P}} = \begin{pmatrix} \dot{P}_1^{(1)} \\ \dot{P}_2^{(1)} \\ \vdots \\ \dot{P}_{n1}^{(1)} \\ \dot{P}_1^{(2)} \\ \dot{P}_2^{(2)} \\ \vdots \\ \dot{P}_{n2}^{(2)} \end{pmatrix} \quad (2-19)$$

Wenn man (2-19) in (2-10) einfügt, dann erhält man die Formel für die adaptierten Parametergradienten:

$$\begin{pmatrix} \dot{P}_1^{(1)} \\ \dot{P}_2^{(1)} \\ \vdots \\ \dot{P}_{n1}^{(1)} \\ \dot{P}_1^{(2)} \\ \dot{P}_2^{(2)} \\ \vdots \\ \dot{P}_{n2}^{(2)} \end{pmatrix} = -e_0 w \quad (2-20)$$

Die Linearisierung kann nach wie vor mit einem Trackingregelgesetz durchgeführt werden. $(L_g h(x))_e$ und $(L_f h(x))_e$ werden aus den abgeschätzten Parametern nach (2-20) abgeleitet, die Stellgröße u ergibt sich:

$$u = \frac{1}{(L_g h(x))_e} \left(- (L_f h(x))_e + \dot{y}_{ref} + k(y_{ref} - y) \right) \quad (2-21)$$

$$u = \frac{1}{\left(\sum_{j=1}^{n_1} P_{(j)}^2(t) L_{g_j} h(x) \right)} \left(- \left(\sum_{i=1}^{n_1} P_i^{(1)}(t) L_{f_i} h(x) \right) + \dot{y}_{ref} + k(y_{ref} - y) \right)$$

Die o.a. Problematik erinnert an die Berechnung der abgeschätzten Parameterfehler \mathbf{f} . Ein anderer wichtiger Punkt ist die Addition der abgeschätzten Parameteränderungen $\dot{\mathbf{f}} = \dot{\mathbf{q}}$ mit ursprünglichem Parameter \mathbf{P}^* . Das bedeutet, dass die Summe des Gradienten des Parameterupdates mit tatsächlichen Parametern \mathbf{P}^* zum Zwecke der Ermittlung der abgeschätzten Parametern, das heißt $\mathbf{P} = \dot{\mathbf{q}} + \mathbf{P}^*$, dient. Diese Darlegung ist auch positive getestet.

2.2. Stabilität

Parameter Updates könnten mit der Lypanov Funktion übereinstimmen. Eine wichtige Problemstellung ist, ob diese Parameter-Aktualisierung eine Senkung der Lypanov Funktion verursacht oder nicht. Die Lypanov Funktion wird in der Gleichung (2-22) angegeben.

$$v(e_0, \mathbf{f}) = \frac{1}{2} e_0^2 + \frac{1}{2} \mathbf{f} \mathbf{f}^T \quad (2-22)$$

Die Beurteilung der Stabilität erfordert die Berechnung der Ableitung der Lypanov Funktion v entlang der Kurve des Errorsystems. So kann man feststellen, ob die Fehlerkurve reduziert, ausgeweitet wird oder eine Instabilität verursacht.

Mit Hilfe der o.a. Berechnung, wird die Richtung der Fehlergröße ausgeforscht.

$$\dot{v}(e_0, \mathbf{f}) = \dot{e}_0 e_0 + \dot{\mathbf{f}} \mathbf{f}^T \quad (2-23)$$

Setzt man (2-13) beziehungsweise (2-10) in die Formel (2-23) ein, d.h.:

$$\dot{v}(e_0, \mathbf{f}) = e_0 (\mathbf{f}^T w - k e_0) + \mathbf{f}^T (-e_0 w) \quad (2-24)$$

so ergibt sich folgendes Ergebnis nach Vereinfachung der Formel:

$$\dot{v}(e_0, \mathbf{f}) = -k e_0^2 \quad (2-25)$$

wie man sieht, ist $-k e_0^2$ immer kleiner als Null:

$$-k e_0^2 \leq 0 \quad (2-26)$$

D.h., die Steigung der Lypanov Funktion ist negativ geworden. Das bedeutet die Abnahme der Lypanov Funktion oder der Stabilität.

$$\dot{v}(e_0, \mathbf{f}) \leq 0 \quad (2-27)$$

3. Neuronale Netze zur Identifizierung beziehungsweise Linearisierung und Regelung der nichtlinearen dynamischen Systeme

Neuronale Netze können Anwendungen im Gebiet der Regelungstechnik haben.

Eine wichtige Anwendung kann der Neuro-Kontroller sein, der als Linearisierungseinheit verwendet werden kann.

Das neuronale Netz wird kurz in diesem Kapitel erklärt. Die Linearisierung durch Neuronale Netze wird im nächsten Kapitel beschrieben.

3.1. Aufbau neuronaler Netze

Die neuronalen Netze können unterschiedlich gestaltet sein. Sie können zum Beispiel als einschichtiges Feedforward oder mehrschichtiges Feedforward symmetrisch oder unsymmetrisch aufgebaut sein. Sie können auch eine unterschiedliche Anzahl von Neuronen für jede Schicht besitzen.

3.1.1. Netze

Die neuronalen Netze bestehen aus Knoten und Kanten. Ein Netz darf verschiedene Schichten mit unterschiedlichen Topologien besitzen. Als Beispiel kann man sich eine Inputschicht, versteckte Schicht (oder hidden layer) und eine Outputschicht für ein klassisches Perceptron vorstellen, wobei die Elemente der versteckten Schicht mit Elementen oder Neuronen von der nächsten beziehungsweise der vorherigen Schicht verbunden sein können.

Neuronale Netze können individuell gestaltet sein. Die Knoten können sehr unterschiedlich sein. Die Auswahl des Netzes, der Lernalgorithmus, Knotenmodelle und die Trainingsmodelle hängen vom Anwendungsgebiet ab. Diese neuronalen Netze erfordern eine zusätzliche Erforschung.

3.1.2. Neuronenmodellierung

Ein den Neuronen verwandtes Modell enthält Eingänge, Gewichte, Aktivierungsfunktion und Ausgänge. In diesem Modell wurden einige synonyme Bedeutungen, die den biologischen Neuronen ähnlich sind, gefunden. Z.B. verknüpfen die Dendriten die Neuronen wie Eingänge zu anderen Neuronen. Axon und Synapsen hängen das Neuron an andere Neuronen, indem die Synapsen die Eingaben oder Aktivierungen gewichten. Diese werden nach einer Art Summierung der Eingaben und nachfolgenden Verarbeitung der Aktivierungen zu Axonen (oder Ausgang). Daraus folgt, dass verarbeitete Informationen an andere Neuronen weitergegeben werden. [Braus 95]

3.1.3. Knoten

Ein Knoten kann ein inneres Produkt (oder Skalarprodukt) zwischen den Eingängen und Gewichten enthalten. In dem Knoten ist ebenfalls eine zweite Einheit als

Aktivierungsfunktion enthalten (siehe Abbildung 4:1). Eine Aktivierungsfunktion ist nichts anderes als eine Transferfunktion. Die Aktivierungsfunktion transformiert die gewichtete Eingabesumme in gewünschte Ausgabewerte.

3.1.4. Kanten

Die Kanten sind einerseits als Datenflusskomponente zwischen unterschiedlichen Schichten graphisch symbolisiert worden, andererseits sieht man die Gewichtungen der geflossenen Informationen oder Aktivierungen zwischen den Knoten, die auf den Kanten liegen. Die Werte der Gewichte dürfen während des Lernens geändert werden, oder es dürfen entsprechende Lernmodelle adaptiert werden.

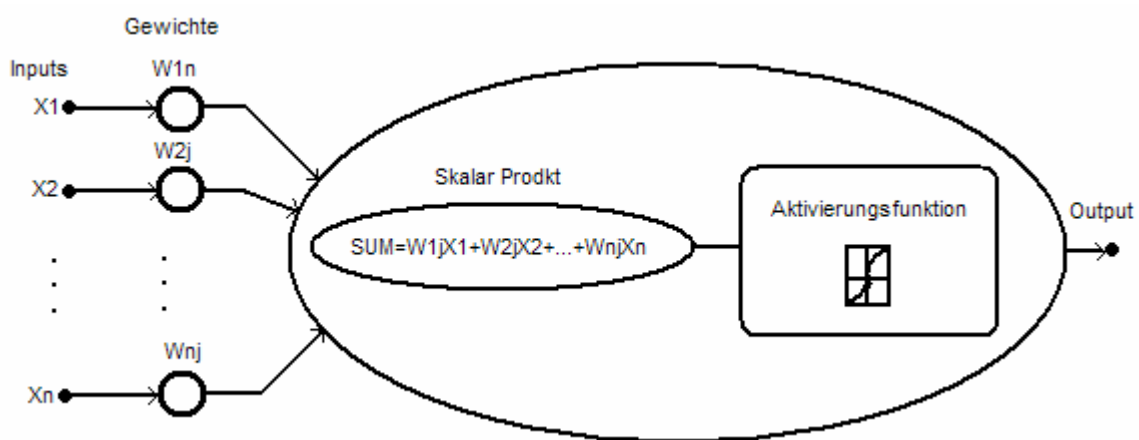


Abbildung 3:1: Neuron.

3.2. Lernalgorithmus

Der Lernalgorithmus des neuronalen Netzes lässt sich in drei verschiedene Kategorien einteilen. Die Kategorien sind Supervised Learning, Unsupervised Learning, und Reinforcement Learning.

Beim Supervised Learning oder Lernen mit Lehrer ist die Eingabe und Lernreaktion oder das Resultat (Output und Input) für das neuronale Netz bekannt. Hier wird die Differenz (oder der Error) zwischen den gewünschten Ausgaben (oder vorgelegten Lernausgaben) und aktuellen Ausgaben berechnet. Netzparameter werden durch eine Kombination zwischen beeinflusstem Trainingvektor und Errorsignal eingestellt. Diese Einstellung wird Schritt für Schritt in Richtung Optimum nachgebildet, damit ein Lehrer emuliert wird.

Unsupervised Learning hat keinen Lehrer. Beim Unsupervised Learning oder beim selbständigen Lernen ist die Ausgabe unbekannt, wobei kein Lehrer emuliert wird. Unsupervised Learning dient zum Zwecke der Klassifizierungen von Daten. D.h., die Merkmale der Eingaben werden verschlüsselt, und die neuen Klassen werden für die Eingabe automatisch erstellt.

Mit Hilfe des Reinforcement Learnings wird kein konkretes Ergebnis erörtert, sondern das Ergebnis wird allgemein betrachtet. Der Reinforcementlernalgorithmus gehört zur Klasse des Learnings ohne Lehrer. Das Lernen der Ein- und Ausgangsabbildungen wird durch

die kontinuierlichen Wechselwirkungen mit der Umgebung zum Zweck der Performanceminimierung durchgeführt. [Hayki 99]

3.2.1. Backpropagation

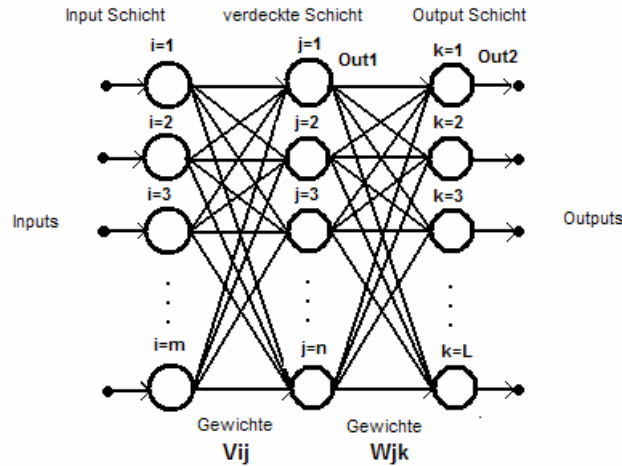


Abbildung 3:2: Neuronale Netze.

Lernalgorithmus der Backpropagation ist für eine Multilayer Architektur geeignet. [Patte 97] Backpropagationlernalgorithmus gehört zur Kategorie Lernen mit Lehrer. In dem Kapitel wird der Backpropagationlernalgorithmus für ein neuronales Netz mit drei Schichten beschrieben(siehe Abbildung 4:2).

(3-1)und (3-3) berechnen die inneren Produkte zwischen Gewichten und entsprechenden Eingängen an einer versteckten Schicht beziehungsweise der Ausgangsschicht. (3-2) und (3-4) berechnen die Istwerte oder die Ausgänge am Output von jeder Schicht durch die Aktivierungsfunktion.

$SP_j^{(1)}$ ist das Skalarprodukt zwischen den Eingängen (oder die Aktivierungen) x_i und den Gewichten v_{ji} . Die Gewichte v_{ji} liegen als Kanten zwischen der Eingangsschicht und der verdeckte Schicht (siehe Abbildung 4:2).

$$SP_j^{(1)} = \sum_{i=1}^m v_{ji} x_i \quad (3-1)$$

$Out_j^{(1)}$ ist der Ausgang der Aktivierungsfunktion am Ausgang der verdeckten Schicht. D.h.,

$SP_j^{(1)}$ wird zu einer Funktion wie z.B. einer Sigmoiden Funktion wie $f(x) = \frac{1}{1 + e^{-ax}}$

weitergegeben, das bedeutet es wird zu $Out_j^{(1)} = f(SP_j^{(1)}) = \frac{1}{1 + e^{-a(SP_j^{(1)})}}$.

$$Out_j^{(1)} = f(SP_j^{(1)}) \quad (3-2)$$

Die gleichen Prozesse wie in (3-1) und (3-2) gelten für die nächste Schicht und die nächste Gewichtsmenge zwischen der verdeckten Schicht und der Ausgangsschicht. Hier ist $SP_j^{(2)}$ ebenfalls das Skalarprodukt zwischen dem Output von der vorherigen Schicht (in diesem Fall der verdeckten Schicht) und Gewichten w_{kj} . Die Gewichte w_{kj} liegen zwischen verdeckter Schicht und Ausgangsschicht.

$$SP_k^{(2)} = \sum_{j=1}^n w_{kj} Out_j^{(1)} \quad (3-3)$$

Der zweite Schichtausgang wird mit $Out_k^{(2)}$ in (3-4) gezeigt.

$$Out_k^{(2)} = f(SP_k^{(2)}) \quad (3-4)$$

Die Gewichtsänderung wird durchgeführt, wenn ein zu großer Unterschied zwischen erwünschtem Ziel und existierender Ausgabe gibt.

In Falle einer zu großen Divergenz muss das neuronale Netz noch weiter trainiert werden, bis die geforderte Ähnlichkeit zwischen Soll- und Istwert mit bestimmter Genauigkeit erreicht wird. Die Gewichte werden somit solange geändert, bis diese gewisse Genauigkeit erfüllt wird. Die Gewichtsänderung wird nach (3-5) und (3-6) kalkuliert.

$$v_{ji}^{new} = v_{ji}^{alt} + \Delta v_{ji} \quad (3-5)$$

$$w_{kj}^{new} = w_{kj}^{alt} + \Delta w_{kj} \quad (3-6)$$

Es reicht also, wenn man Δv_{ij} und Δw_{jk} analytisch berechnet. Sie müssen solange rekursiv aktualisiert werden, bis das Ziel oder gewünschte Präzision erreicht ist. Die zwei Werte Δv_{ij} und Δw_{jk} sollen jeweils rekursiv mit altem Wert addiert werden. Die Größe der beiden Werte basieren auf der Errorberechnung am Ausgang. Das Ziel ist Ermittlung einer Relation zwischen Gewichtsänderung und Fehlergröße. Die Formel (3-7) wird als Gradient auf der Fehlerfläche interpretiert.

Bei (3-7) ist ∂E die Fehleränderung, und ∂w_{kj} ist die Gewichtsänderung der Gewichtsmenge zwischen der verdeckten Schicht und Ausgangsschicht. h ist die Lernrate und kann als konstante Zahl betrachtet werden. Δw_{kj} ist die Gewichtsänderung.

$$\Delta w_{kj} = -h \frac{\partial E}{\partial w_{kj}} \quad (3-7)$$

Ein Punkt ist als aktuelles Gewicht in der Abbildung 3:3 dargestellt. Hier kann man sehen, wie sehr sich der Error verändert, wenn das Gewicht geändert wird. Die Formel (3-7) ist die partielle Ableitung auf der Errorfläche unter Berücksichtigung der Gewichtsänderung. Wie in (3-7) dargestellt, wird die Gewichtsänderung negativ dargestellt. Das bedeutet eine

Gradientreduzierung des gesamten Fehlers oder Minimierung des Fehlers. Diese Gewichtsänderung ist auch als Deltaregel bekannt.

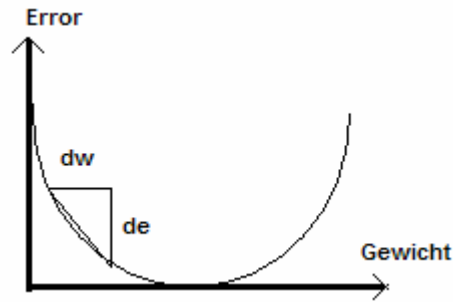


Abbildung 3.3: Gewichtsänderung und Fehlerminimierung.

Dies bedeutet, man versucht eine äquivalente Gleichung gleich wie Formel (3-7) zu finden, wobei diese Gleichung in der Praxis leicht mit vorhandenen Daten umgesetzt werden kann.

Man multipliziert einfach Nenner und Zähler von (3-7) in $\partial SP_k^{(2)}$:

$$\Delta w_{kj} = -h \frac{\partial E}{\partial SP_k^{(2)}} \frac{\partial SP_k^{(2)}}{\partial w_{kj}} \quad (3-8)$$

Der zweite Bruch von (3-8) wird mit einem gleichwertigen Ausdruck ersetzt, das heißt statt $SP_k^{(2)}$ liegt (3-3).

$$\frac{\partial SP_k^{(2)}}{\partial w_{kj}} = \frac{\partial \left(\sum_{i=1}^n w_{kj} Out_j^{(1)} \right)}{\partial w_{kj}} \quad (3-9)$$

nach Vereinfachung ergibt sich (3-10):

$$\frac{\partial SP_k^{(2)}}{\partial w_{kj}} = Out_j^{(1)} \quad (3-10)$$

Der erste Bruch von (3-8) wird auch verändert, d.h. seine Nenner und Zähler werden mit $\partial Out_j^{(2)}$ multipliziert:

$$\frac{\partial E}{\partial SP_k^{(2)}} = \frac{\partial E}{\partial Out_k^{(2)}} \frac{\partial Out_k^{(2)}}{\partial SP_k^{(2)}} \quad (3-11)$$

(3-11) kann wieder anders beschrieben werden. D.h., der erste und zweite Bruch von (3-11) wird mit dem gleichen Begriff umschrieben. Folglich setzt man

$\frac{\partial E}{\partial Out_k^{(2)}} = -(soll - ist)$, und $\frac{\partial Out_k^{(2)}}{\partial SP_k^{(2)}} = f' SP_k^{(2)}$ in (3-11), wobei *soll* gleich Target ist oder

das Ziel und *ist* die aktuelle Lage darstellt oder $Out_k^{(2)}$ am Ausgang der neuronalen Netze. D.h.:

$$\frac{\partial E}{\partial SP_k^{(2)}} = -(soll - ist) f'(SP_k^{(2)}) \quad (3-12)$$

Hier werden zwei Formeln kombiniert, nämlich (3-12) und (3-10) werden in (3-8) eingesetzt. Als Resultat ergibt sich in (3-13) :

$$\Delta w_{kj} = h(soll - ist) f'(SP_k^{(2)}) Out_j^{(1)} \quad (3-13)$$

Zur Vereinfachung wurde der Anteil $(soll - ist) f'(SP_k^{(2)})$ von (3-13) in d_k umbenannt. D.h.:

$$d_k = (soll - ist) f'(SP_k^{(2)}) \quad (3-14)$$

Somit werden die Gewichtsänderungen der Kanten zwischen der verdeckten Schicht und Ausgangsschicht ganz einfach mittels (3-15) berechnet.

$$\Delta w_{kj} = h d_k Out_j^{(1)} \quad (3-15)$$

Es ist möglich die Formel (3-15) in die Praxis umzusetzen.

Hier sieht man die Relation der Gewichtsänderung für die Gewichte zwischen Eingangsschicht und verdeckter Schicht. Die ursprüngliche Relation für die Gewichtsänderung ist ähnlich der gegebenen Relation von (3-7), hat aber unterschiedliche Nominierungen, d.h.:

$$\Delta v_{ji} = -h \frac{\partial E}{\partial v_{ji}} \quad (3-16)$$

Auf die gleiche Art und Weise wird der gleiche Anteil wie $\partial SP_j^{(1)}$ im Nenner und Zähler multipliziert, also:

$$\Delta v_{ji} = -h \frac{\partial E}{\partial SP_j^{(1)}} \frac{\partial SP_j^{(1)}}{\partial v_{ji}} \quad (3-17)$$

Der zweite Bruch von (3-17) wird mit der gleichen Beziehung umgetauscht, somit ist der gleiche Wert:

$$\frac{\partial SP_j^{(1)}}{\partial v_{ji}} = \frac{\partial \left(\sum_{i=1}^m v_{ji} x_i \right)}{\partial v_{ji}} \quad (3-18)$$

nach Vereinfachung bleibt:

$$\frac{\partial SP_j^{(1)}}{\partial v_{ji}} = x_i \quad (3-19)$$

Für den ersten Bruch von (3-17) hat man:

$$\frac{\partial E}{\partial SP_j^{(1)}} = \frac{\partial E}{\partial Out_j^{(1)}} \frac{\partial Out_j^{(1)}}{\partial SP_j^{(1)}} \quad (3-20)$$

Hierbei wurden wieder Nenner und Zähler in (3-20) mit $\partial Out_j^{(1)}$ multipliziert.
Für den ersten Bruch von (3-20) gilt (3-21):

$$\frac{\partial E}{\partial Out_j^{(1)}} = \frac{1}{2} \frac{\partial \sum_{k=1}^L (t_k - Out_k^{(2)})^2}{\partial Out_j^{(1)}} \quad (3-21)$$

Statt $Out_k^{(2)}$ setzt man einen gleichwertigen Term ein:

$$\frac{\partial E}{\partial Out_j^{(1)}} = \frac{1}{2} \frac{\partial \sum_{k=1}^L (t_k - f(SP_k^{(2)}))^2}{\partial Out_j^{(1)}} = \frac{1}{2} \frac{\partial \sum_{k=1}^L \left(t_k - f \left(\sum_{i=1}^n w_{kj} Out_j^{(1)} \right) \right)^2}{\partial Out_j^{(1)}} \quad (3-22)$$

Durch Ableitung der (3-22) ergibt sich folgendes Resultat:

$$\frac{\partial E}{\partial Out_j^{(1)}} = 2 \frac{1}{2} \sum_{k=1}^L \left(t_k - f \left(\sum_{i=1}^n w_{kj} Out_j^{(1)} \right) \right) \left(0 - w_{kj} f' \left(\sum_{i=1}^n w_{kj} Out_j^{(1)} \right) \right) \quad (3-23)$$

(3-23) wird nach Vereinfachung zu:

$$\frac{\partial E}{\partial Out_j^{(1)}} = \sum_{k=1}^L (t_k - f(SP_k^{(2)})) (-w_{kj} f'(SP_k^{(2)})) \quad (3-24)$$

Aber für den zweiten Term von (3-20) gilt,

$$\frac{\partial Out_j^{(1)}}{\partial SP_j^{(1)}} = f'(SP_j^{(1)}) \quad (3-25)$$

Jetzt werden beide Formeln, (3-25) und (3-24) in (3-20) eingesetzt.
Das Ergebnis ist:

$$\frac{\partial E}{\partial SP_j^{(1)}} = \sum_{k=1}^L (t_k - f(SP_k^{(2)})) (-w_{kj} f'(SP_k^{(2)})) f'(SP_j^{(1)}) \quad (3-26)$$

oder

$$\frac{\partial E}{\partial SP_j^{(1)}} = - \sum_{k=1}^L (t_k - f(SP_k^{(2)})) f'(SP_k^{(2)}) w_{kj} f'(SP_j^{(1)}) \quad (3-27)$$

D.h.:

$$\frac{\partial E}{\partial SP_j^{(1)}} = - \sum_{k=1}^L d_k(w_{kj}) f'(SP_j^{(1)}) \quad (3-28)$$

Man ersetzt (3-28) und (3-19) in (3-17), d.h. die Gewichtsänderung zwischen Eingangsschicht und verdeckter Schicht wird im Anwendungsfall nach (3-29) berechnet:

$$\Delta v_{ji} = -h \left(- \sum_{k=1}^L d_k(w_{kj}) f'(SP_j^{(1)}) \right) x_i \quad (3-29)$$

(3-30) ergibt sich nach einer Vereinfachung:

$$\Delta v_{ji} = h x_i f'(SP_j^{(1)}) \sum_{k=1}^L d_k(w_{kj}) \quad (3-30)$$

Die Ergebnisse und notwendigen Formel zum praktischen Modellieren sind in Abbildung 4:4 dargestellt:

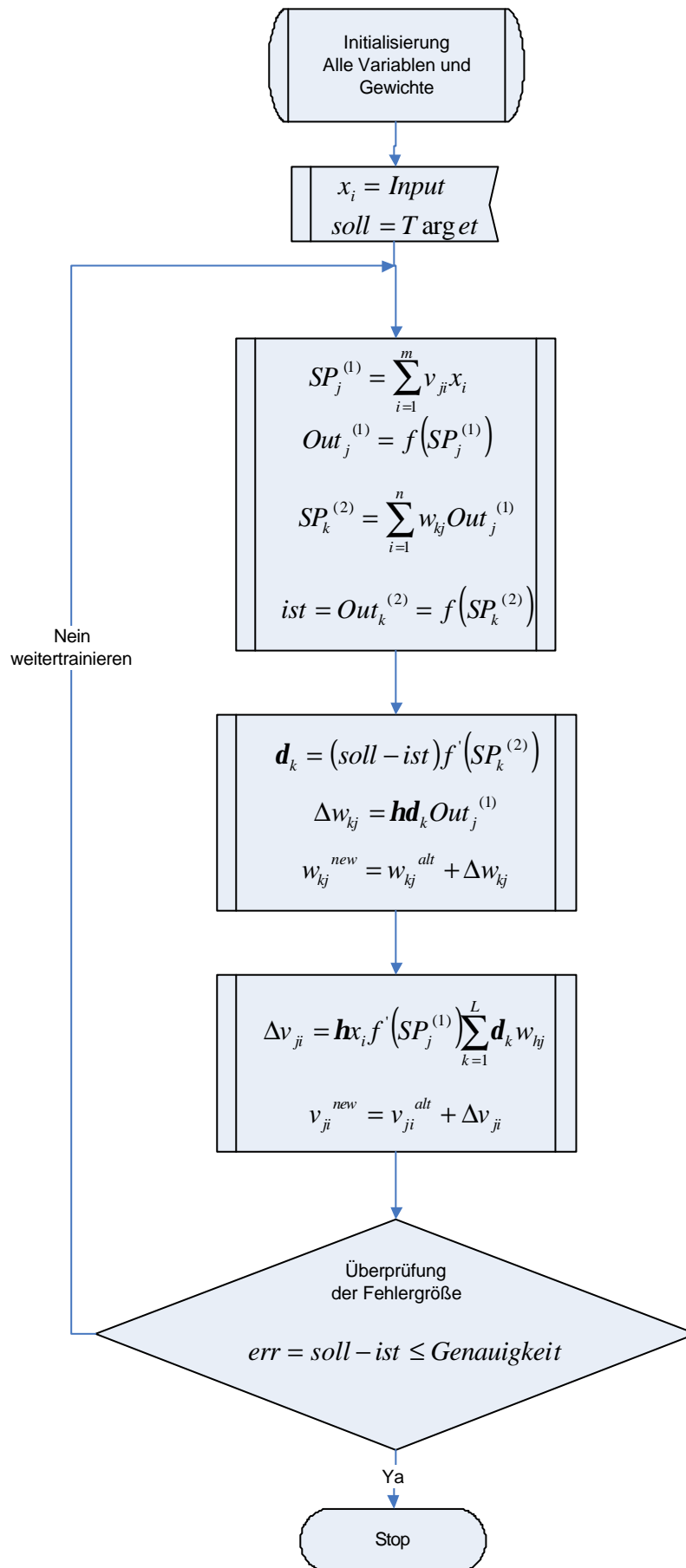


Abbildung 3:4: Backpropagation Algorithmus.

3.2.2. Backpropagation mit Momentum Term

Der Unterschied zwischen Backpropagation mit Momentum und normaler Backpropagation liegt in der Gewichtsänderung. Backpropagation Algorithmus mit Momentum Term wird in jedem Iterationsdurchlauf sowohl die aktuelle Gewichtsänderung von dem aktuellen Schritt als auch die schon berechnete Gewichtsänderung von dem vorherigen Schritt beeinflussen (siehe [Weing 01]).

D.h:

$$v_{ji}^{new} = v_{ji}^{alt} + \Delta v_{ji}(t) + \mathbf{m} \Delta v_{ji}(t-1) \quad (3-31)$$

$$w_{kj}^{new} = w_{kj}^{alt} + \Delta w_{kj}(t) + \mathbf{m} \Delta w_{kj}(t-1) \quad (3-32)$$

$\Delta v_{kj}(t-1)$ ist die Gewichtsänderung der vorherigen Stufe, und \mathbf{m} ist das Momentum Parameter und liegt zwischen $0 \leq \mathbf{m} \leq 1$.

So wird der Lernalgorithmus von lokalem Minimum nicht viel beeinflusst, und die Lerngeschwindigkeit erhöht sich.

3.2.3. Gauss-Newton Methode

Die Berechnung der Gewichtsänderung basiert auf der Gewichtsabschätzung von der Taylorreihe zweiter Ordnung. Für die Taylor Reihe zweiter Ordnung gilt folgendes [Cherk 98] [Nelle 01]:

$$f(x) \approx f(x_{AP}) + \nabla f(x_{AP})^T (x - x_{AP}) + (x - x_{AP})^T H_f(x_{AP})(x - x_{AP}) \quad (3-33)$$

$H_f(x_{AP})$ ist die Hesse-Matrix mit der zweiten partiellen Ableitung von Funktion f im Arbeitspunkt x_{AP} . Wenn $H_f(x_{AP})$ einen positiven Wert besitzt, dann ist das Minimum von der rechten Seite der Gleichung (3-33) das folgende:

$$\nabla f(x_{AP})^T (x - x_{AP}) + (x - x_{AP})^T H_f(x_{AP})(x - x_{AP}) = 0 \quad (3-34)$$

(3-34) zeigt die Iterationsformel der zweiten Ordnungsmethode (oder Newton Methode) auf:

$$x - x_{AP} = -(H_f(x_{AP}))^{-1} \nabla f(x_{AP})^T \quad (3-35)$$

Unter Berücksichtigung der Zeitsequenz oder Schrittfolge k wird folgende Formel hergeleitet:

$$x(k+1) - x(k) = -(H(x(k)))^{-1} \nabla f(x(k))^T \quad (3-36)$$

$\Delta x(k)$ wird anstelle der Differenz zwischen altem und aktuellem Wert von x eingesetzt.
D.h.:

$$\Delta x(k) = -(H(x(k)))^{-1} \nabla f(x(k))^T \quad (3-37)$$

oder

$$H(x(k)) \Delta x(k) = -\nabla f(x(k))^T \quad (3-38)$$

In dem o.a. System in (3-38) oder (3-37) Δx ist die Gewichtsänderung im Schritt k beinhaltet.

Δx wird bei der Iteration zum alten Gewicht dazu gegeben.
D.h.;

$$x^{new} = x^{alt} + \Delta x \quad (3-39)$$

Das Ziel des Newton Algorithmus ist die Optimierung oder Minimierung des empirischen Risikos. Das Risiko ist definiert als:

$$Err(w) = \sum_{i=1}^n \|y_i - f(x_i, w)\|^2 \quad (3-40)$$

Wenn e als Residuum mit (3-41) definiert wird, dann bedeutet das:

$$e_i(w) = y_i - f(x_i, w) \quad (3-41)$$

Dann gilt natürlich auch:

$$Err(w) = \sum_{i=1}^n \|e_i(w)\|^2 \quad (3-42)$$

Zur Berechnung des Gradienten und der Hesse-Matrix braucht man die Jacobi-Matrix. Die Jacobi-Matrix von der Funktion $f(x_i, w_j)$ auf w ist gleich:

$$[J(w)]_{ij} = \frac{\partial f(x_i, w_j)}{\partial w_j} \quad (3-43)$$

Die Jacobi-Matrix muss für verschiedenes Pattern und verschiedene Knoten und Kanten eingerichtet werden. Die Einstellung und Integration von der Jacobi-Matrix kann für Multischichtnetzwerke kompliziert sein.

$$[J(w)]_{nm}^p = \begin{pmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \dots & \frac{\partial e_{1,1}}{\partial w_m} \\ \frac{\partial e_{2,1}}{\partial w_1} & \frac{\partial e_{2,1}}{\partial w_2} & \dots & \frac{\partial e_{2,1}}{\partial w_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{n,1}}{\partial w_1} & \frac{\partial e_{n,1}}{\partial w_2} & \dots & \frac{\partial e_{n,1}}{\partial w_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{1,p}}{\partial w_1} & \frac{\partial e_{1,p}}{\partial w_2} & \dots & \frac{\partial e_{1,p}}{\partial w_m} \\ \frac{\partial e_{2,p}}{\partial w_1} & \frac{\partial e_{2,p}}{\partial w_2} & \dots & \frac{\partial e_{2,p}}{\partial w_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{n,p}}{\partial w_1} & \frac{\partial e_{n,p}}{\partial w_2} & \dots & \frac{\partial e_{n,p}}{\partial w_m} \end{pmatrix} = \begin{pmatrix} \frac{\partial(soll - ist)_{1,1}}{\partial w_1} & \frac{\partial(soll - ist)_{1,1}}{\partial w_2} & \dots & \frac{\partial(soll - ist)_{1,1}}{\partial w_m} \\ \frac{\partial(soll - ist)_{2,1}}{\partial w_1} & \frac{\partial(soll - ist)_{2,1}}{\partial w_2} & \dots & \frac{\partial(soll - ist)_{2,1}}{\partial w_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial(soll - ist)_{n,1}}{\partial w_1} & \frac{\partial(soll - ist)_{n,1}}{\partial w_2} & \dots & \frac{\partial(soll - ist)_{n,1}}{\partial w_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial(soll - ist)_{1,p}}{\partial w_1} & \frac{\partial(soll - ist)_{1,p}}{\partial w_2} & \dots & \frac{\partial(soll - ist)_{1,p}}{\partial w_m} \\ \frac{\partial(soll - ist)_{2,p}}{\partial w_1} & \frac{\partial(soll - ist)_{2,p}}{\partial w_2} & \dots & \frac{\partial(soll - ist)_{2,p}}{\partial w_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial(soll - ist)_{n,p}}{\partial w_1} & \frac{\partial(soll - ist)_{n,p}}{\partial w_2} & \dots & \frac{\partial(soll - ist)_{n,p}}{\partial w_m} \end{pmatrix} \quad (3-44)$$

$e_{n,p}$ beschreibt den Errorausgang der Neuronennummer n und Patternnummer p . Für jeden Knoten ist die Gewichtsveränderung ∂w von Kante eins bis m in (3-44) dargestellt. Der Gradient des Risikos $Err(w)$ ist in (3-45) dargestellt:

$$\nabla Err(w) = J^T(w)e(w) \quad (3-45)$$

Die Hesse-Matrix des Risikos $Err(w)$ entspricht:

$$H_{Err}(w) = J^T(w)J(w) + \sum_{i=1}^n e_i(w)H_i(w) \quad (3-46)$$

Im Falle des Verzichts auf den zweiten Term (3-46), wird die Abschätzung von der Hesse-Matrix vereinfacht, d.h. für die Abschätzung von der Hesse-Matrix ergibt sich folgendes:

$$H_{Err}(w) = J^T(w)J(w) \quad (3-47)$$

Wenn man die beiden letzten Formeln, nämlich (3-47) und (3-45) in der gegebenen Gleichung (3-38) zueinander in Beziehung setzt, kommt man zu dem folgenden Ergebnis:

$$J^T(w(k))J(w(k))\Delta w(k) = -J^T(w(k))e(w(k)) \quad (3-48)$$

Und Δw wird:

$$\Delta w(k) = -\left(J^T(w(k))J(w(k))\right)^{-1} J^T(w(k))e(w(k)) \quad (3-49)$$

Wird (3-49) in (3-39) gesetzt, dann erhält man:

$$w(k+1) = w(k) - \left(J^T(w(k))J(w(k))\right)^{-1} J^T(w(k))e(w(k)) \quad (3-50)$$

Unter Berücksichtigung der Schrittgröße \mathbf{h} , wird (3-50):

$$w(k+1) = w(k) - \mathbf{h} \left(J^T(w(k))J(w(k))\right)^{-1} J^T(w(k))e(w(k)) \quad (3-51)$$

3.2.4. Levenberg Marquardt Algorithmus

Der Levenberg Algorithmus ist die alternative Lösung der Gauss-Newton Methode. Der Levenberg Algorithmus ist der Ridge Regression ähnlich. (siehe [Nelle 01] [Cherk 98]). Hier wird der Levenberg Algorithmus im Zusammenhang mit dem vorherigen Kapitel als Erweiterung des Gauss-Newton Algorithmus betrachtet.

Der Levenberg Algorithmus wird angewendet, wenn Matrix $J^T J$ (3-49) ein Rangdefizit hat, d.h., die Determinante $\det(J^T J)$ ist gleich Null. Unter diesen Umständen sieht man, entsprechend der Formel (3-49) $\Delta w \rightarrow \infty$. Um dies zu vermeiden, benutzt man den Levenberg –Marquadt Algorithmus. Beim Levenberg Algorithmus wird zu Term $J^T J$ ein zusätzlicher Anteil $\mathbf{a}_k I$ bestimmt, wobei I die Einheitsmatrix und \mathbf{a}_k der Einflussfaktor sind.

Der Unterschied liegt in der Gewichtsaktualisierung von Formel (3-48). (3-48) ist in Form von (3-52) umgewandelt worden;

$$\left(J^T(w(k))J(w(k)) + \mathbf{a}_k I\right)\Delta w(k) = -J^T(w(k))e(w(k)) \quad (3-52)$$

Von (3-52) $\Delta w(k)$ wird extrahiert:

$$\Delta w(k) = -\left(J^T(w(k))J(w(k)) + \mathbf{a}_k I\right)^{-1} J^T(w(k))e(w(k)) \quad (3-53)$$

Die Gewichtsaktualisierung mit Einfluss der Schrittgröße beträgt:

$$w(k+1) = w(k) - \mathbf{h} \left(J^T(w(k))J(w(k)) + \mathbf{a}_k I\right)^{-1} J^T(w(k))e(w(k)) \quad (3-54)$$

Bei der Aktualisierung des Algorithmus wird \mathbf{a}_k geändert. Dies hängt vom Optimum ab. Wenn das Optimum weit vom aktuellen Punkt abweicht, muss \mathbf{a}_k größer werden. In der Nähe des Optimums muss \mathbf{a}_k verkleinert werden.

Bei dem kleinen \mathbf{a}_k verhält sich der Levenberg Algorithmus wie ein Gauß Newton Algorithmus. Dies ist in (3-54) und (3-51) veranschaulicht. Bei dem großen \mathbf{a}_k hat der Levenberg Algorithmus die Eigenschaften wie der Steepest Descent Algorithmus (das einfache Steepest Descent kann mit Backpropagation oder Backpropagation mit Momentum Term veranschaulicht werden).

4.Approximative Zustandslinearisierung mit neuronalen Netzen und Interpolationsmethoden

Um eine Zustandslinearisierung zu erreichen, ist die Aufnahme bestimmter Informationen von der Strecke erforderlich. Diese Informationsaufnahme ist unter dem Namen“ Identifikation oder Modellierung“ der Strecke bekannt.

Man nimmt an, dass ein Zugriff auf die Stellgröße beziehungsweise Zustandsgröße und den Ausgang existiert. Die mathematische oder funktionale Beschreibung des Verhalten der Strecke ist nicht bekannt.

Wie in Kapitel eins dargelegt ist, wird für den Beginn der Linearisierung die Zustandsraumdarstellung der Strecke benötigt .

Diese Daten könnten mit Hilfe eines Messsystems und mit bestimmtem Methode oder Algorithmen erhalten werden.

Die wichtigsten Vorkenntnisse über ein System oder eine Strecke ist eine Differentialgleichung n ter Ordnung und der relative Grad der Strecke.

Allerdings zur Linearisierung muss ebenfalls die unbekannte nichtlineare Systemmatrix $f(x)$, die nichtlineare Steuermatrix $g(x)$ und Beobachtungsmatrix $h(x)$ bestimmt werden.

In diesem Kapitel wird zunächst die Identifizierung der unbekannten Strecke erklärt, und dann werden die Lösungsansätze zur Linearisierung und Regulierung präsentiert. Zur Durchführung der Linearisierungsalgorithmen gibt es mehrere Möglichkeiten. Es werden zwei Methoden erwähnt. Eine Lösung ist die Anwendung der neuronalen Netze, die andere Lösung zieht die Interpolationsmethoden in Betracht. Beide Methoden sind auch getestet worden, und im nächsten Kapitel werden sie dargelegt. (siehe [He 98])

4.1. Identifizierung, Modellierung und Approximierung der unbekannten nichtlinearen Objekte.

Zur Identifizierung der nichtlinearen Strecke (oder Objekte) werden bestimmte ursprüngliche Daten benötigt. Diese Daten sind die Stellgröße und die Zustandsgrößen. Ziel ist es, die Strecke mit Daten einzuspeisen, die Beobachtung des Verhalten der Zustandgröße und der Strecke und der Ausganggröße.

Vorausgesetzt wird die Kenntnis über die relative Ordnung der Differentialgleichung der Strecke.

Wenn diese Zahl nicht bekannt ist, dann kann man trotzdem Linearisierungsverfahren für verschiedene Ordnungen durchführen, bis ein positives Ergebnis vorhanden ist.

Dieses Verfahren kann aufwendig sein.

Die Eingabe zur Identifizierung der Zustandsgröße muss nach [He 98] eine Zufallsgröße sein. Aber aufgrund einer vollständigen Zustandsmenge innerhalb eines bestimmten Feldes der Zustandgröße ist in dieser Arbeit die Eingabe als eine additive Kombination zwischen einem Raster und einer Zufallsvariablen gewählt worden.

Wichtig ist ebenfalls die Bestimmung des Gleichgewichtspunktes. Systemzustand und Stellgröße sollen zum Gleichgewichtspunkt verschoben werden.

Die Verschiebung erübrigt sich, wenn ein geeignetes Objekt mit Gleichgewichtspunkt gleich Null vorhanden ist.

Sonst wird die Eingabe(Raster oder Zufallvariable) als Input zur Stellgröße u verwendet.

Danach muss beobachtet werden bei welchem u Nullzustände auftreten. Diese Eingabe wird mit u_0 bezeichnet, und x_0 ist der entsprechende Zustand. Dann muss u nach $u - u_0$ und x zu $x - x_0$ verschoben werden. D.h.:

$$u = u - u_0 \quad (4-1)$$

$$x = x - x_0 \quad (4-2)$$

Für manche Systeme liegt der Ausgleichspunkt (oder equilibrium point) auf Null, das bedeutet $u_0 = 0$, und auch $x_0 = 0$, dann wird eine Verschiebung überflüssig. D.h. die Stellgröße bleibt erhalten, oder $u = u$ und $x = x$.

4.1.1. Identifizierung und Modellierung der nichtlinearen Systemmatrix $f(x)$

Die Zustandsraumdarstellung wird in folgender Form dargestellt:

$$\dot{x} = f(x) + g(x)u \quad (4-3)$$

$$y = h(x) \quad (4-4)$$

Zur Modellierung $f(x)$ wird zuerst die Stellgröße u gleich Null gesetzt:

$$u = 0 \quad (4-5)$$

Dann wird (4-3) gleich:

$$\dot{x} = f(x) + g(x)u = f(x) + 0 = f(x) \quad (4-6)$$

Wie man sieht, ist hier die Stellgröße bereits gleich Null, und die Zustandgröße wird zu $f(x)$:

$$\dot{x} = f(x) \quad (4-7)$$

Bei der Identifizierung des $\dot{x} = f(x)$, sollen die Zustandsänderung der aktuellen Zustandsraumdarstellung in einer bestimmten kleinen Zeitspanne geändert, gemessen und registriert werden.

Das bedeutet, als Eingabe werden die Zustandsvariablen mit Werten belegt (stochastisch oder Addition von stochastischen und Rasterwerten). D.h., die Zustandsvariable von $\dot{x} = f(x)$

ändert sich in Abhängigkeit von der Zeit und der stochastisch bekannten Position oder Größe. Der Endzustand soll mit einem Messsystem bestimmt und registriert werden.

$$\dot{x} = f(x_{rand}) \quad (4-8)$$

oder

$$\frac{\partial x}{\partial t} = f(x_{rand}) \quad (4-9)$$

In der praktischen Anwendung werden die Differenzwerte verwendet, (weil eine Ableitung in dem Fall nicht möglich ist):

$$\frac{x_2 - x_1}{t_2 - t_1} = f(x_{rand}) \quad (4-10)$$

Die Zeitspanne wird fix gewählt, das bedeutet $t_2 - t_1 = t$. Der Anfangszustand x_{rand} wird stochastisch angenommen. Die Messwerte stellen die Zustandsänderung oder x_2 dar.

Das heißt, man kann nur $\frac{x_{mess} - x_{rand}}{t}$ als $f(x)$ akzeptieren.

$$f(x) = \frac{x_2 - x_1}{t} \quad (4-11)$$

Die Zustandgrößen \dot{x} oder $\frac{x_2 - x_1}{t}$ sollen registriert werden. Diese Daten können als simulierte Datenquelle von $f(x)$ zum Trainieren der neuronalen Netze verwendet werden.

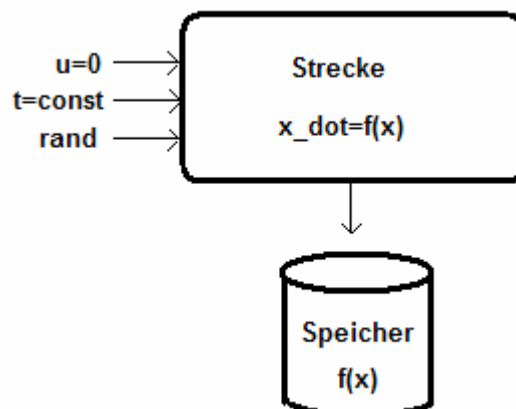


Abbildung 4.1: Identifikation von $f(x)$.

4.1.2. Identifizierung und Modellierung der Zustandsgrößen $f(x)+g(x)$

Die Modellierung von $f(x)+g(x)$ ist Voraussetzung zur Modellierung der nichtlinearen Steuermatrix $g(x)$. Dabei soll wie oben über die Einstellung der Stellgröße u die Zustandsgrößen $f(x)+g(x)$ extrahiert werden.

Man setzt u gleich:

$$u = 1 \quad (4-12)$$

Die Zustandgröße nimmt folgende Form an:

$$\dot{x} = f(x) + g(x)u = f(x) + g(x) \cdot 1 = f(x) + g(x) \quad (4-13)$$

So wie oben wird nun $f(x)+g(x)$ mit einer neuen Einstellung u berechnet. Die jetzige Zustandgröße ist:

$$\dot{x} = f(x) + g(x) \quad (4-14)$$

Oder

$$\frac{\partial x}{\partial t} = f(x) + g(x) \quad (4-15)$$

Bei der praktischen Umsetzung:

$$\frac{x_2 - x_1}{t_2 - t_1} = f(x) + g(x) \quad (4-16)$$

Für eine Messung mit einer festen Zeitspanne:

$$\frac{x_{mess} - x_{rand}}{t} = f(x) + g(x) \quad (4-17)$$

Das heißt $f(x)+g(x)$ ist gleich:

$$f(x) + g(x) = \frac{x_2 - x_1}{t} \quad (4-18)$$

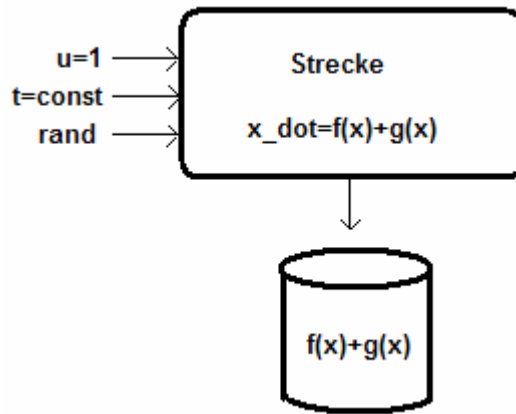


Abbildung 4:2: Identifikation von $f(x) + g(x)$.

4.1.3. Identifizierung und Modellierung der nichtlinearen Steuermatrix $g(x)$

Die Modellierung der nichtlinearen Steuermatrix $g(x)$ ist am einfachsten. Hier soll zunächst nach Einstellung von $u = 0$ die Zustandsgröße $f(x)$ ermittelt werden, dann wird $f(x)$ von der bereit ermittelten Zustandsgröße $f(x) + g(x)$ mit $u = 1$ subtrahiert. Ergebnis ist der Rest der Subtraktion, d.h. $g(x)$.

$$g(x) = (f(x) + g(x)) - f(x) = g(x) \quad (4-19)$$

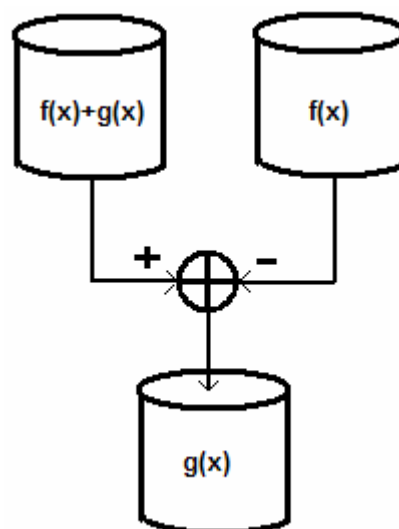


Abbildung 4:3: Ermittlung von $g(x)$.

Unten sieht man die Modellierungen von $g(x)$ und $f(x)$ in dem entsprechenden Algorithmus.

Wie hier dargestellt ist, ist nur $g(x)$ und $f(x)$ gespeichert worden. Eine Speicherung der Zustandsgröße $f(x) + g(x)$ ist hier nicht notwendig.

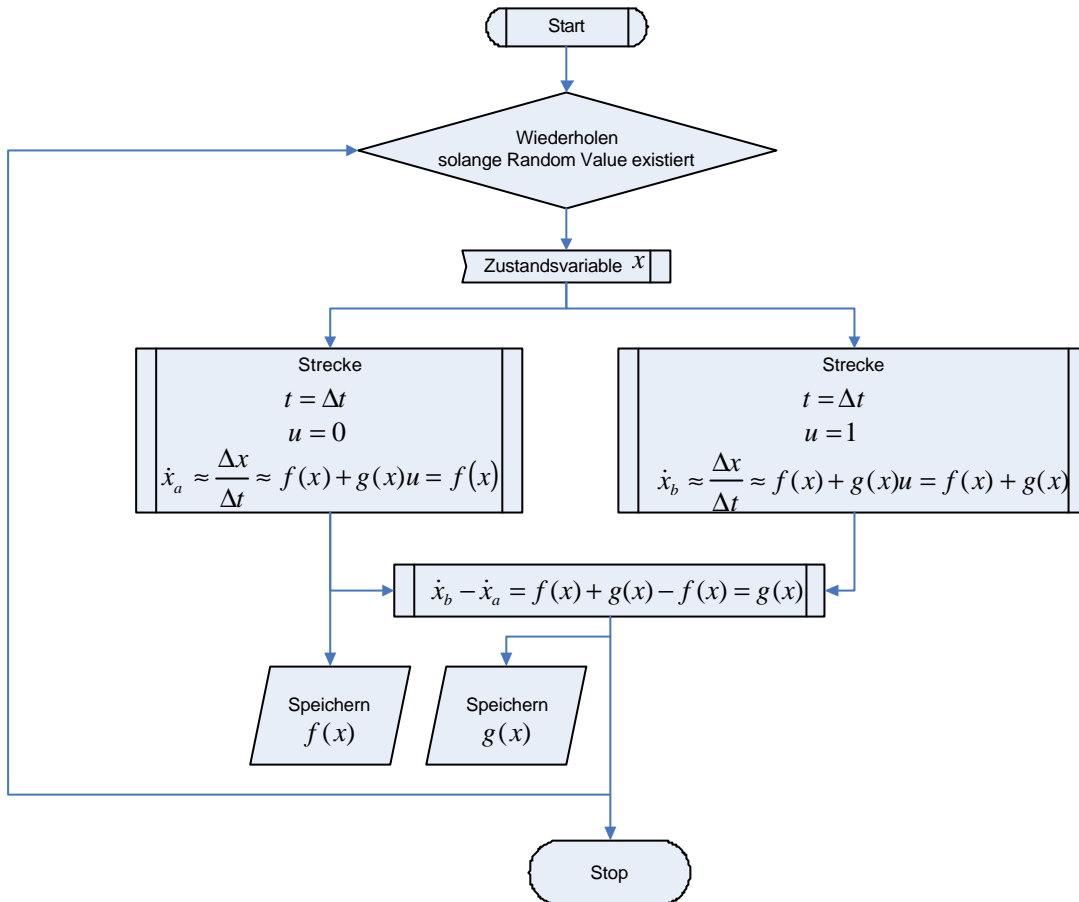


Abbildung 4:4: Identifikationsalgorithmus der Zustandsraumdarstellung.

Es sind zwei neuronale Netze zum Trainieren der Daten $f(x)$ und $g(x)$ erforderlich. Diese beiden Netze können im nächsten Schritt angewendet werden, um eine Linearisierung zu erreichen. Es ist auch aber möglich, dass die o.a. Daten in der Datenbank gespeichert werden, und schließlich zum Zwecke Simulation der Linearisierungseinheiten abgerufen werden. Der Grund hierfür ist, dass $f(x)$ und $g(x)$ im Linearisierungssystem nicht angewendet werden, sondern sie sind nur ein Mittel zur Bildung der Linearisierungseinheiten durch Lie-Ableitung.

Da die Daten von $f(x)$ und $g(x)$ immer aktualisiert werden müssen, bleibt die Adaption immer aktuell, und somit bleibt die entsprechende Linearisierung auch immer auf dem neuesten Stand. Solche Verfahren brauchen ein Echtzeitsystem. Die Bildung solcher schnellen Systeme mit neuronalen Netzen könnte schwierig sein, weil die neuronalen Netze eine gewisse Zeit zum Trainieren brauchen. Dieses Problem ist bei den Offline Systemen mit den neuronalen Netzen kein Thema mehr, weil die Anwendung der neuronale Netze anders definiert wird. Denn hier wird zuerst das neuronale Netz vollständig trainiert, und danach angewendet. Wenn man aber eine Einheit $f(x)$ mit neuronalem Netz simulieren will, braucht man zum Trainieren neuronaler Netze sowohl gemessene, registrierte Daten der Modelle, z.B. $f(x)$ Daten als auch die entsprechende, wahrscheinliche, registrierte Eingabe.

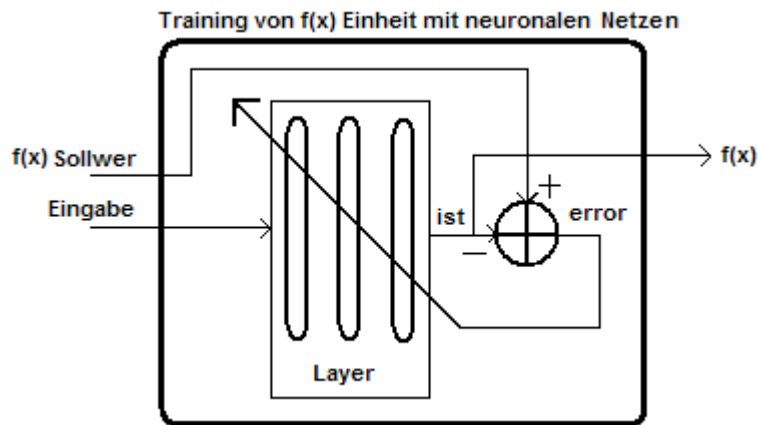


Abbildung 4:5: Training von neuronalen Netzen.

Im Gegensatz zur Trainingphase des neuronalen Netzes, bei der Simulation, wird nur eine Menge als Eingabe betrachtet. Entsprechend dieser Eingaben sollte das neuronale Netz die richtige Entscheidung am Ausgang treffen.

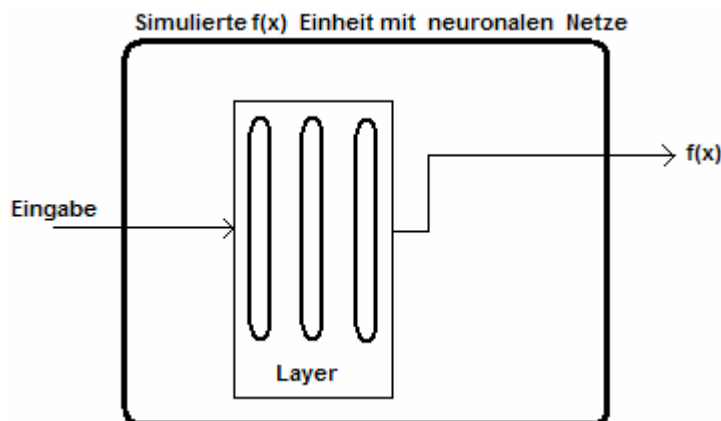


Abbildung 4:6: Anwendung von neuronalen Netzen.

Beim Offline System können die Daten von $f(x)$ und $g(x)$ auch ohne Einsatz von neuronalen Netzen angewendet werden. So werden sie lediglich in der Datenbank gespeichert und können abgerufen und zur Vorbereitung der Linearisierungseinheit mit neuronalen Netzen angewendet werden. Zum Trainieren der neuronalen Netze könnte dem System eine einheitliche Eingabe eingespeist werden.

4.1.4. Approximierung und Modellierung der Linearisierungseinheit

Zur Regelung einer nichtlinearen Strecke ist Linearisierung der nichtlinearen Strecke von großer Bedeutung. Der erste Schritt ist die Linearisierung. Die Bildung des Reglers ist der zweite wichtige Schritt, und am Ende kann ein Tracking durchgeführt werden.

Der wichtigste Baustein ist die Simulation der Lie-Ableitungseinheit. Die partielle Ableitung muss von verschiedenen veränderlichen Variablen ableitbar sein.

Entsprechend der Formel (1-34) und (1-36) wird für die Linearisierung eine Kette von kombinierten Operation der partiellen Ableitungen und Skalar-Produkten benötigt. Die Formel (1-36) kann in folgender Form dargestellt werden.

$$\begin{pmatrix} \dot{z}_1 \\ \vdots \\ \dot{z}_{n-1} \\ \dot{z}_n \end{pmatrix} = \begin{pmatrix} L_f^{-1} h(x) \\ \vdots \\ L_f^{n-1} h(x) \\ L_f^n h(x) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ L_g L_f^{n-1} h(x) \end{pmatrix} \frac{1}{L_g L_f^{n-1} h(x)} (-L_f^n h(x) + v) \quad (4-20)$$

Die einzelnen Einheiten von $L_f^{-1} h(x)$ bis $L_f^n h(x)$ beziehungsweise $L_g L_f^{n-1} h(x)$ können auch mit neuronalen Netzen simuliert werden.

Zum Simulieren von $L_f^{-1} h(x)$ muss $\frac{\partial h(x)}{\partial x} f(x)$ simuliert werden. D.h., einerseits muss die

Partiellableitung $\frac{\partial h(x)}{\partial x}$ simuliert werden, andererseits muss diese partielle Ableitung durch

ein Skalarprodukt mit $f(x)$ multipliziert werden. Bei $L_g L_f^{n-1} h(x)$ muss dies genauso durchgeführt werden, nur wird beim letzten Schritt statt $f(x)$ das $g(x)$ mit $\frac{\partial h(x)}{\partial x}$

multipliziert, d.h. $\frac{\partial h(x)}{\partial x} g(x)$.

Die Ableitung des letzten Schrittes hängt davon ab, in wie weit die Stellgröße im abgeleiteten Ausgang verändert wird.

Für den ersten Teil oder die partielle Ableitung wird die Ableitung nach Formel (4-21) simuliert:

$$h'(x) = \lim_{\Delta x \rightarrow 0} \frac{h(x + \Delta x) - h(x)}{\Delta x} \quad (4-21)$$

Die Ableitung einer Funktion für jede konkrete Variable muss durchgeführt werden, und somit werden alle Variablen in einer Matrix zu einem Skalarprodukt gestellt. D.h.:

$$\left(\lim_{\Delta x_1 \rightarrow 0} \frac{h(x_1 + \Delta x_1, x_2, \dots, x_n)}{\Delta x_1} \quad \lim_{\Delta x_2 \rightarrow 0} \frac{h(x_1, x_2 + \Delta x_2, \dots, x_n)}{\Delta x_2} \quad \dots \quad \lim_{\Delta x_n \rightarrow 0} \frac{h(x_1, x_2, \dots, x_n + \Delta x_n)}{\Delta x_n} \right) \quad (4-22)$$

Bei der Formel (4-22) ist die Existenz von mehreren Zuständen nicht ausgeschlossen, deshalb sind mehrere Ableitungen von diversen Variablen erforderlich.

Der nächste Schritt ist das Skalarprodukt zwischen dem abgeleiteten Term und $f(x)$. D.h.:

$$\left(\lim_{\Delta x_1 \rightarrow 0} \frac{h(x_1 + \Delta x_1, x_2, \dots, x_n)}{\Delta x_1} \quad \lim_{\Delta x_2 \rightarrow 0} \frac{h(x_1, x_2 + \Delta x_2, \dots, x_n)}{\Delta x_2} \quad \dots \quad \lim_{\Delta x_n \rightarrow 0} \frac{h(x_1, x_2, \dots, x_n + \Delta x_n)}{\Delta x_n} \right) \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix} \quad (4-23)$$

Solche Skalarprodukte sind meistens leicht zu simulieren, aufgrund ihrer einfachen mathematischen Form.

Wenn $h(x)$ und $f(x)$ und $g(x)$ als neuronales Netz simuliert werden, und wenn die simulierten Einheiten NQ_1 und N_f und N_g sind, dann gilt folgendes:

Für $h(x)$ gilt

$$NQ_1(x) = h(x) \quad (4-24)$$

Und für $f(x)$

$$N_f(x) = f(x) \quad (4-25)$$

Und $g(x)$ wird

$$N_g(x) = g(x) \quad (4-26)$$

Wenn die Formel (4-23) mit dem neuronalen Netzen in Verbindung gebracht wird, dann wird (4-23) gleich:

$$\left(\lim_{\Delta x_1 \rightarrow 0} \frac{NQ_1(x_1 + \Delta x_1, x_2, \dots, x_n)}{\Delta x_1} \quad \lim_{\Delta x_2 \rightarrow 0} \frac{NQ_1(x_1, x_2 + \Delta x_2, \dots, x_n)}{\Delta x_2} \quad \dots \quad \lim_{\Delta x_n \rightarrow 0} \frac{NQ_1(x_1, x_2, \dots, x_n + \Delta x_n)}{\Delta x_n} \right) \begin{pmatrix} N_{f_1}(x) \\ N_{f_2}(x) \\ \vdots \\ N_{f_n}(x) \end{pmatrix} \quad (4-27)$$

Entsprechend dem Skalarprodukt zwischen dem Ableitungsanteil und dem $f(x)$ Anteil, enthält die Formel (4-28) den folgenden skalaren Anteil.

$$L_{N_f}^{-1} NQ_1(x) = \lim_{\Delta x_1 \rightarrow 0} \frac{NQ_1(x_1 + \Delta x_1, x_2, \dots, x_n)}{\Delta x_1} N_{f_1}(x) + \lim_{\Delta x_2 \rightarrow 0} \frac{NQ_1(x_1, x_2 + \Delta x_2, \dots, x_n)}{\Delta x_2} N_{f_2}(x) \dots \\ + \lim_{\Delta x_n \rightarrow 0} \frac{NQ_1(x_1, x_2, \dots, x_n + \Delta x_n)}{\Delta x_n} N_{f_n}(x)$$

(4-28)

Die Formel (4-28) ist einerseits eine Beschreibung der Lie-Ableitung, andererseits ist sie eine Solldatenvorbereitung für den nächsten Schritt.

Das folgende Bild zeigt die graphische Darstellung der Formel (4-28). Wie man sieht, wird der Sollwert zum Trainieren an die nächste Einheit weitergegeben. Die neue Einheit muss mit der gleichen Methode für die nächste Stufe bearbeitet werden, bis die Trainingsolldaten der neuronalen Netze erzeugt werden.

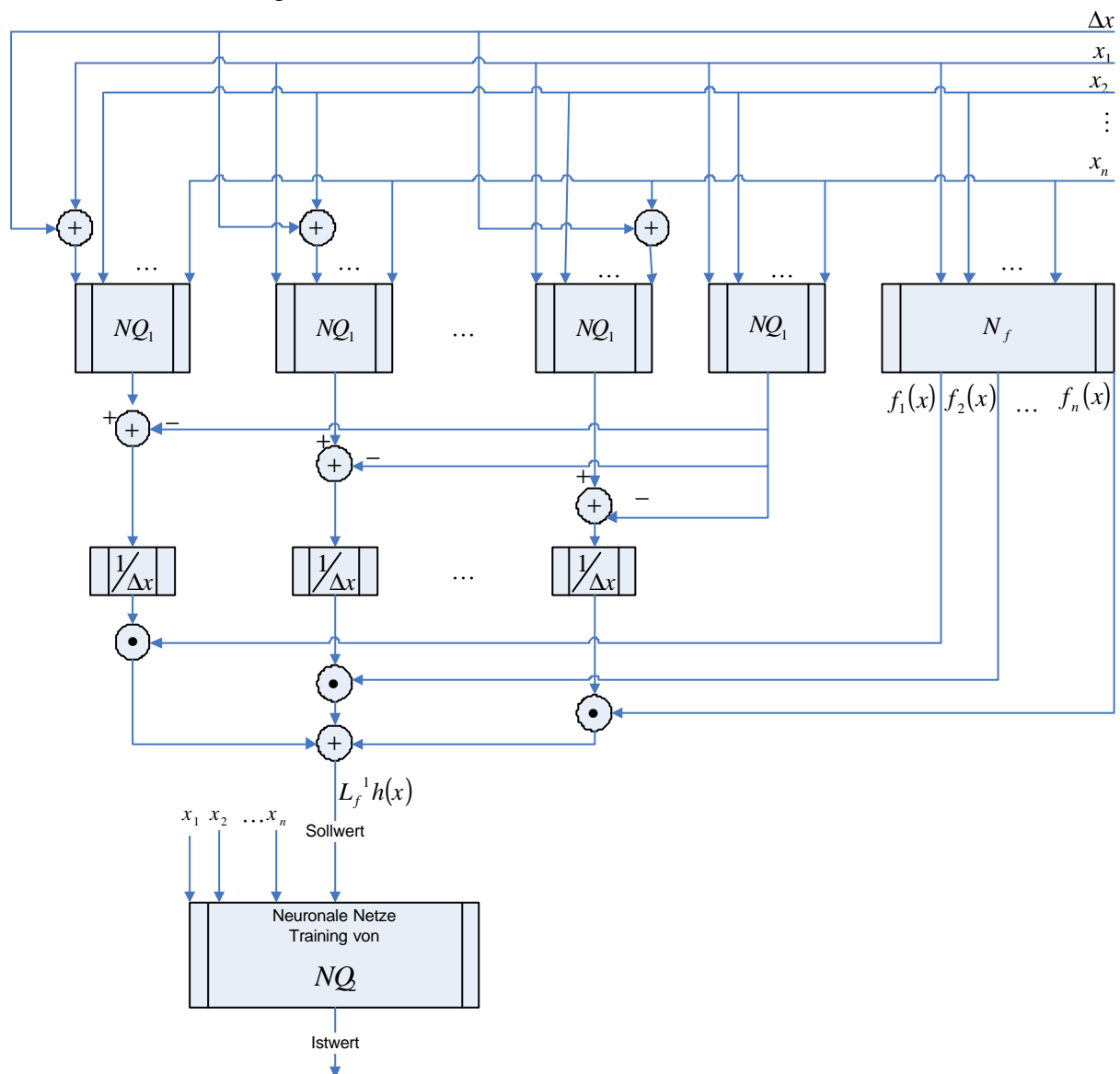


Abbildung 4:7: Solldatenvorbereitung zum Simulieren von der Lie-Ableitung mittels neuronaler Netze.

In der Abbildung 4:7 werden die Daten in Bezug auf $L_f^1 h(x)$ zum Trainieren der neuronalen Netze NQ_2 weitergeleitet.

Man kann $L_f^2 h(x)$ von $L_f^1 h(x)$ und $L_f^3 h(x)$ von $L_f^2 h(x)$ und so weiter extrahieren.

Das bedeutet:

$$L_f^{i+1}h(x) = \frac{\partial(L_f^i h(x))}{\partial x} \cdot f(x) \quad (4-29)$$

Diese Vorgehensweise ist auch für die neuronalen Netze gedacht, d.h.:

$$NQ_{i+1} = \frac{\partial(NQ_i(x))}{\partial x} \cdot N_f(x) \quad (4-30)$$

Entsprechend der Formel (4-30) werden die nächsten Einheiten aus den vorherigen Einheiten gebildet. Dies ist in folgendem Bild mit Hilfe der Differenzwerte dargestellt:

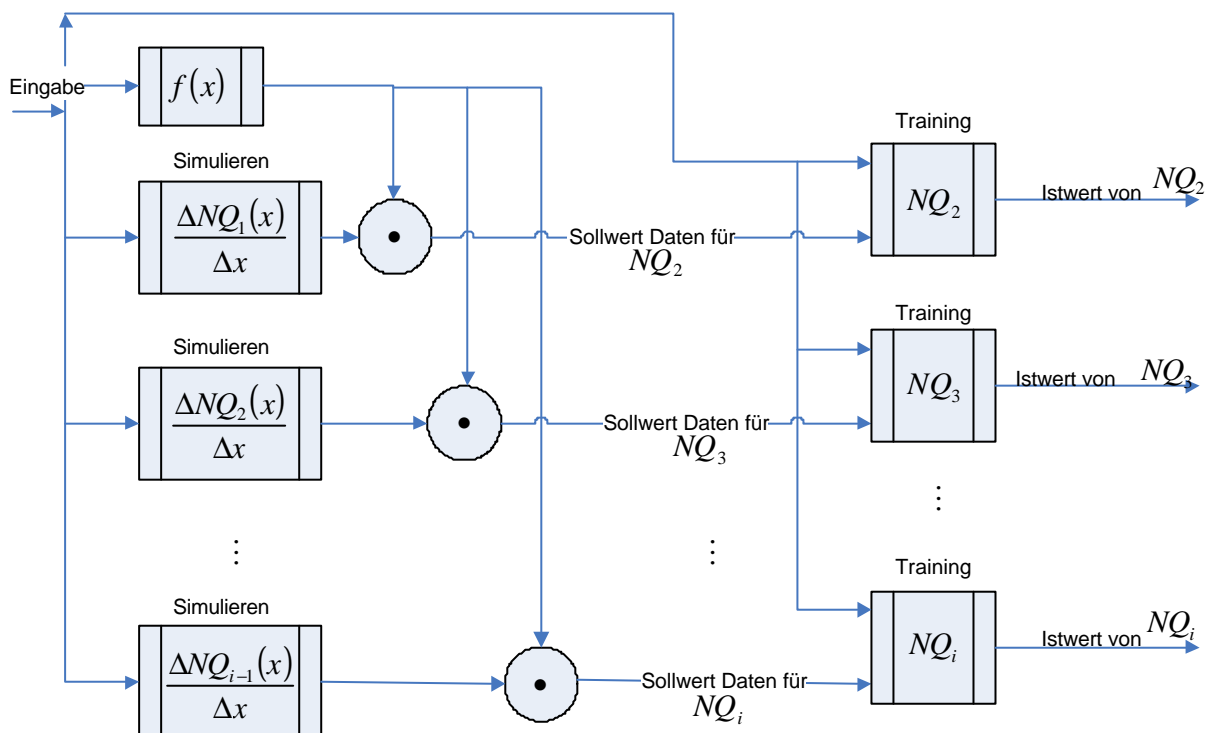


Abbildung 4:8: Ermittlung der Solldaten für die nächste Stufe.

4.2. Architektur des gesamten Regelkreises

Die Abbildung 4:9 ist ähnlich Abbildung 1:2. Nur an den Stellen der Linearisierungsfunktionen sind neuronale Netze gesetzt worden. Die Eingabe von neuronalen Netzen sind die Zustandsgrößen.

Die Erweiterung der Modellierung kann Echtzeitadaption oder Onlineadaption sein. D.h., die Gewichte der neuronalen Netze werden immer entsprechend der Streckeneigenschaften aktualisiert.

Eine andere Art der Modellierung kann das Offline-Training sein, d.h. es müssen zunächst die neuronalen Netze trainiert und dann in die Linearisierungseinheit eingesetzt werden. D.h., die Adaption wird nur am Anfang durchgeführt. Für die Strecken, die keine fixen Eigenschaften besitzen, ist das ein Nachteil. So können nämlich die Änderung der Eigenschaften von solchen Strecken nicht gefasst werden.

Eine Linearisierung durch neuronale Netze ohne Tracking ist mit Hilfe der Formel (1-54) möglich. Nur an der Stelle exakter Linearisierungseinheiten ist die abgeschätzte Einheit von neuronalen Netzen eingesetzt worden:

$$u = \frac{1}{L_{N_g} NQ_n(x)} \left(-L_{N_f} NQ_n(x) - k_1 NQ_1(x) - k_2 NQ_2(x) - k_3 NQ_3(x) - \dots - k_n NQ_{n-1}(x) + v_{new} \right) \quad (4-31)$$

Die Bildung eines Linearisierungssystems mit inkludiertem Tracking, kann nach Umsetzung der Formel (1-60) mit den neuronalen Einheiten durch die Formel (4-32) dargestellt werden

$$u = \frac{1}{L_{N_g} NQ_n(x)} \left(v_{ref}^n - L_{N_f} NQ_n(x) + k_1 (v_{ref} - NQ_1(x)) + k_2 (\dot{v}_{ref} - NQ_2(x)) + \dots + k_n (v_{ref}^{n-1} - NQ_{n-1}(x)) \right) \quad (4-32)$$

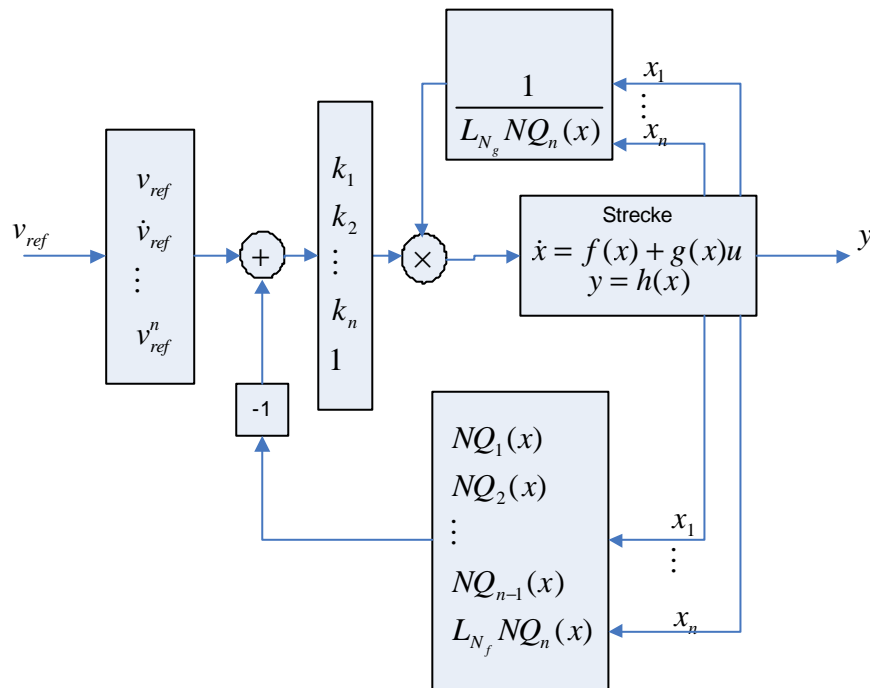


Abbildung 4:9: Bildung eines Regelkreis mit approximativen neuronalen Netzen.

In der Abbildung 4:9 ist sowohl die Linearisierung und Regelung als auch das Tracking dargestellt worden. Die Regelung, das Tracking und die Linearisierung wurde bereits im vorherigen Kapitel erörtert.

4.3. Analogie zwischen neuronalem Netz und Interpolationsmethode

Man kann neuronale Netze durch Interpolationsmodelle ersetzen. Das bedeutet, die schon dargelegten allgemeinen theoretischen Grundlagen der nichtlinearen Systeme können durch Interpolationsmodelle umgesetzt werden. Mit anderen Worten könnte man sagen, dass die Funktionen, die bis jetzt mit neuronalen Netzen ersetzt wurden, jetzt auch mit Interpolationseinheiten realisiert werden können.

Bei der Interpolationseinheit kann man mit Hilfe der bekannten Punkte von verschiedenen Koordinaten die unbekannten Zwischenpunkte finden. Dies ist z.B. bei der Newton Methode oder Lagrange Methode der Fall. Meistens wird bei dem Interpolationsverfahren eine mathematische Funktion mit Hilfe der bekannten Koordinatenpunkte bestimmt.

Diese neuen bekannten Zwischenpunkte werden der Funktion eingegeben. Die unbekannten Zwischenpunkte anderer Koordinaten können so berechnet und abgerufen werden.

Im Gegensatz zum neuronalen Netz braucht das Interpolationsmodell keinen Trainingsalgorithmus. Statt dessen werden die Entscheidungen mit Hilfe einer Funktion getroffen.

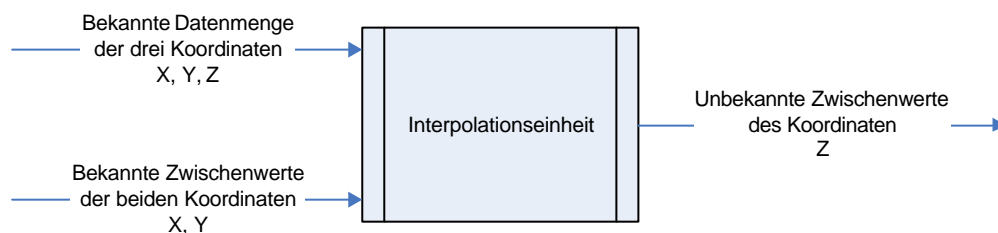


Abbildung 4:10: Interpolationsbeispiel für drei Koordinatensysteme.

Die Lie-Ableitung mit der Interpolationseinheit zu bestimmen, ist nicht schwer. Die Abbildung 5:11 zeigt die Simulation von $L_f^{-1}h(x)$ mit den Interpolationsmodellen.

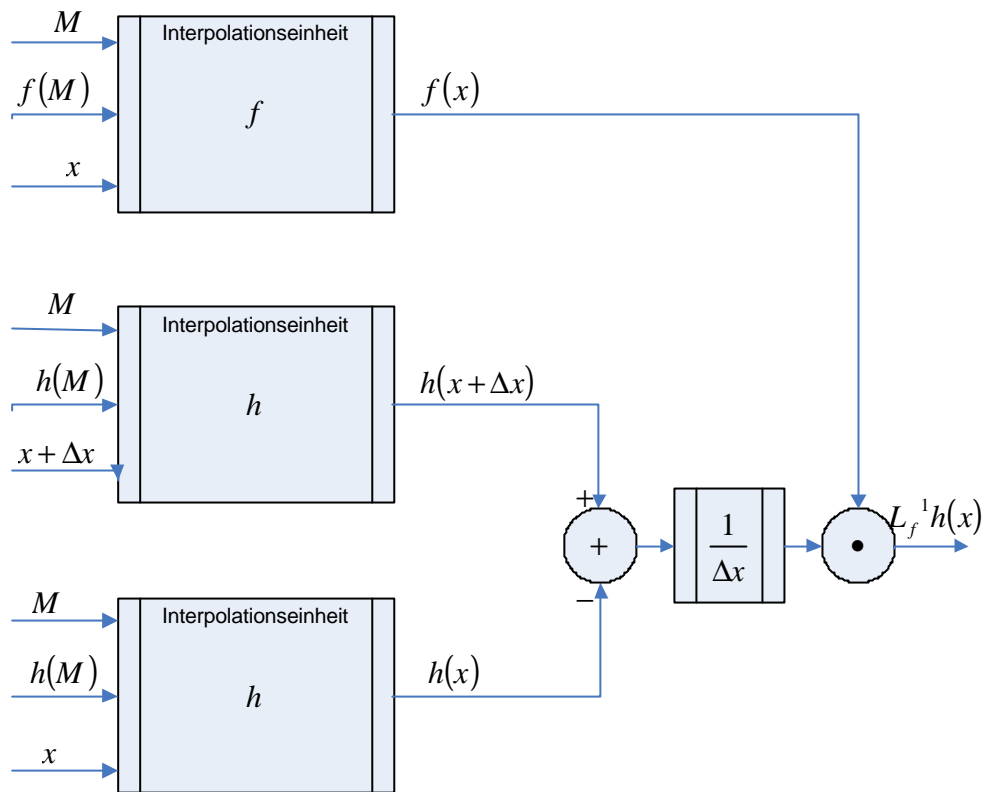


Abbildung 4:11: Lie Ableitung mit Interpolationsmethode.

Die Abbildung 4:11 zeigt eine Lie-Ableitung mit Hilfe der Interpolationseinheit. Hier ist M die bekannte Muttermenge, x ist die bekannte Untermenge von M , und $x + \Delta x$ sind die bekannten Zwischenwerte von x . $f(x)$ und $h(x)$ sind interpolierten Ausgänge der Untermenge x .

$h(x + \Delta x)$ ist der interpolierte Ausgang der Zwischenwerte von $x + \Delta x$ der Muttermenge M . Die beide anderen Eingänge sind $h(M)$ und $f(M)$. Diese beiden bestimmen die bekannten Eingänge der Muttermengen.

Die weiteren Linearisierungseinheiten können auch entsprechend der Formel (4-30) mit gleicher Methode berechnet werden.

4.4. Die Fehlerberechnung

Wie bereits o.a., kann man mit Hilfe der neuronalen Netzen oder Interpolationseinheiten Funktionen und Ableitungen allgemein in einer simulierten Form abschätzen.

So können die Fehler berechnet werden. Die Differenz zwischen exakter Lösung und abgeschätzter Lösung wird als Fehler bezeichnet. Somit gibt es bei der Simulation keine ganz exakte Linearisierung, sondern die Linearisierung basiert auf abgeschätzten Einheiten. Die Kalkulation ist für neuronale Netze gedacht, kann aber auch auf die Interpolationsmodelle angewandt werden.

Der Fehler zwischen $f(x)$ und $N_f(x)$ wird durch Δ_{ef} dargestellt:

$$\|f(x) - N_f(x)\| = \Delta_{ef} \quad (4-33)$$

Der Fehler zwischen $g(x)$ und $N_g(x)$ wird auch mit Δ_{eg} gezeigt.

$$\|g(x) - N_g(x)\| = \Delta_{eg} \quad (4-34)$$

Der Fehler zwischen $L_f^i h(x)$ und NQ_{i+1} ist Δ_{Ni}

$$\|L_f^i h(x) - NQ_{i+1}\| = \Delta_{Ni} \quad (4-35)$$

Die Differenz zwischen $\frac{\partial L_f^i h(x)}{\partial x}$ und $\frac{\partial NQ_{i+1}}{\partial x}$ wird durch Δ_{edi} dargestellt.

$$\left\| \frac{\partial L_f^i h(x)}{\partial x} - \frac{\partial NQ_{i+1}}{\partial x} \right\| = \Delta_{edi} \quad (4-36)$$

Somit wird das Maximum der abgeschätzten Fehler mit Δ_{em} bezeichnet.

$$\|L_f^0 h(x) - NQ_{0+1}\| \leq \Delta_{em} \quad (4-37)$$

oder

$$\|h(x) - NQ_1\| \leq \Delta_{em} \quad (4-38)$$

Das Ziel ist die Darstellung der abgeschätzten Fehlergröße der Lie-Ableitung $L_f^{i+1} h(x) = \frac{\partial h(x)}{\partial x} f(x)$, wenn diese Ableitung mit neuronalen Netzen simuliert wird.

Die simulierte Lie-Ableitung mit neuronalen Netzen wird mit Fehlern addiert. Somit kann durch die folgende Gleichung die Beziehung zwischen exakter und simulierter Lösung dargestellt werden:

$$L_f^{i+1} h(x) = \frac{\partial L_f^i h(x)}{\partial x} f(x) = \left(\frac{\partial NQ_i}{\partial x} + \Delta_{edi} \right) (N_f + \Delta_{ef}) \quad (4-39)$$

Die Formel ist gleich:

$$L_f^{i+1} h(x) = \frac{\partial NQ_i}{\partial x} N_f + \Delta_{edi} N_f + \frac{\partial NQ_i}{\partial x} \Delta_{ef} + \Delta_{edi} \Delta_{ef} \quad (4-40)$$

D.h. der beeinflusste Term des Fehlers ist:

$$\Delta_{eL} = \Delta_{edi} N_f + \frac{\partial NQ_i}{\partial x} \Delta_{ef} + \Delta_{edi} \Delta_{ef} \quad (4-41)$$

Die Lie-Ableitung wird folgendermaßen verändert:

$$L_f^{i+1} h(x) = \frac{\partial NQ_i}{\partial x} N_f + \Delta_{eL} \quad (4-42)$$

Man kann auch die Fehlerberechnung für die ganze Linearisierungseinheit erweitern. Solche Fehlerberechnungen sind komplizierter. Zunächst muss man die Differenz zwischen exakter Stellgröße und abgeschätzter Stellgröße durch neuronale Netze berechnen, und das Ergebnis muss gleich $\|u - u_N\|$ gesetzt werden.

$$\|u - u_N\| = \left\| \frac{-L_f^n h(x) - k_1 h(x) - \dots - k_n L_f^{n-1} h(x) + v}{L_g L_f^{n-1} h(x)} - \frac{-L_{N_f} NQ_n(x) - k_1 NQ_1(x) - \dots - k_n NQ_{n-1}(x) + v}{L_{N_g} NQ_n(x)} \right\| \quad (4-43)$$

5. Test und Simulation

Mehrere Systeme sind bei der Diplomarbeit getestet worden. Sie wurden aber nicht alle ausführlich dargestellt, da dies den Rahmen dieser Arbeit sprengen würde. Obwohl die Nebensysteme des Standard Algorithmus, des Backpropagation Algorithmus und der Backpropagation mit Momentum detailliert simuliert und getestet wurden, sind sie nicht ausführlich aufgeführt worden.

Hauptsächlich ist die adaptive exakte Linearisierung mit Hilfe von Simulink simuliert worden.

Im Zusammenhang mit der approximativen Linearisierung sind die Linearisierungseinheiten sowie die neuronalen Netze und die Interpolationseinheiten getestet worden. Sie sind teilweise mit der M-File und der übrige Teil ist mit Simulink getestet worden.

Die hypothetische Identifikation, die Lie-Ableitungen, und das Training der neuronalen Netze oder Interpolationen sind auf M-File programmiert worden.

Zur Simulation der Linearisierungsmethode wurden bereits trainierte neuronale Netze oder interpolierte Funktion auf der Simulinkenebene extrahiert, und sie sind auf den entsprechenden Strecken getestet worden. Ein erfolgreiches Testergebnis war nur unter bestimmten Voraussetzung möglich.

5.1. Test der adaptiven Linearisierung

In diesem Test ist die Strecke mit folgender Differentialgleichung beschrieben.

$$\dot{x}_1 = p_1^{(1)} x_1^2 + p_2^{(1)} \sin x_1 + (p_1^{(2)} x_1 + p_2^{(2)} x_1^3) u \quad (5-1)$$

$$y = x_1 \quad (5-2)$$

Entsprechend der zwei Gleichungen (5-1) und (5-2) und der dargestellten Formeln (2-1) und (2-2) kann man einzelne Teile von den Gleichungen (5-1) und (5-2) in folgender Form darstellen:

$$f_1(x) = x_1^2 \quad (5-3)$$

$$f_2(x) = \sin(x) \quad (5-4)$$

$$g_1(x) = x_1 \quad (5-5)$$

$$g_2(x) = x_1^3 \quad (5-6)$$

$$h(x) = x_1 \quad (5-7)$$

Gemäß der Formel (2-11) ist

$w^{(1)}$ gleich:

$$w^{(1)} = - \begin{pmatrix} L_{f_1} h(x) \\ L_{f_2} h(x) \end{pmatrix} = - \begin{pmatrix} \frac{\partial h}{\partial x} f_1(x) \\ \frac{\partial h}{\partial x} f_2(x) \end{pmatrix} = - \begin{pmatrix} 1 \cdot x_1^2 \\ 1 \cdot \sin(x) \end{pmatrix} = - \begin{pmatrix} x_1^2 \\ \sin(x) \end{pmatrix} \quad (5-8)$$

Und $w^{(2)}$ ist:

$$w^{(2)} = \begin{pmatrix} L_{g_1} h(x) \\ L_{g_2} h(x) \end{pmatrix} \frac{\left(\sum_{i=1}^2 p_i^{(1)} L_{f_i} h(x) \right) - v}{\sum_{j=1}^2 p_j^{(2)} L_{g_j} h(x)} = \begin{pmatrix} \frac{\partial h(x)}{\partial x} g_1(x) \\ \frac{\partial h(x)}{\partial x} g_2(x) \end{pmatrix} \frac{p_1^{(1)} L_{f_1} h(x) + p_2^{(1)} L_{f_2} h(x) - v}{p_1^{(2)} L_{g_1} h(x) + p_2^{(2)} L_{g_2} h(x)} \quad (5-9)$$

$$w^{(2)} = \begin{pmatrix} 1 \cdot x_1 \\ 1 \cdot x_1^3 \end{pmatrix} \frac{p_1^{(1)} \cdot 1 \cdot x_1^2 + p_2^{(1)} \cdot 1 \cdot \sin(x_1) - v}{p_1^{(2)} \cdot 1 \cdot x_1 + p_2^{(2)} \cdot 1 \cdot x_1^3} = \begin{pmatrix} x_1 \\ x_1^3 \end{pmatrix} \frac{p_1^{(1)} \cdot x_1^2 + p_2^{(1)} \cdot \sin(x_1) - v}{p_1^{(2)} \cdot x_1 + p_2^{(2)} \cdot x_1^3} \quad (5-10)$$

oder

$$w^{(2)} = \begin{pmatrix} x_1 \frac{p_1^{(1)} \cdot x_1^2 + p_2^{(1)} \cdot \sin(x_1) - v}{p_1^{(2)} \cdot x_1 + p_2^{(2)} \cdot x_1^3} \\ x_1^3 \frac{p_1^{(1)} \cdot x_1^2 + p_2^{(1)} \cdot \sin(x_1) - v}{p_1^{(2)} \cdot x_1 + p_2^{(2)} \cdot x_1^3} \end{pmatrix} \quad (5-11)$$

Somit wird w zu:

$$w = \begin{pmatrix} w^{(1)} \\ w^{(2)} \end{pmatrix} = \begin{pmatrix} -x_1^2 \\ -\sin(x_1) \\ x_1 \frac{p_1^{(1)} \cdot x_1^2 + p_2^{(1)} \cdot \sin(x_1) - v}{p_1^{(2)} \cdot x_1 + p_2^{(2)} \cdot x_1^3} \\ x_1^3 \frac{p_1^{(1)} \cdot x_1^2 + p_2^{(1)} \cdot \sin(x_1) - v}{p_1^{(2)} \cdot x_1 + p_2^{(2)} \cdot x_1^3} \end{pmatrix} \quad (5-12)$$

Hier sieht man den abgeschätzten Parametergradient:

$$\dot{p} = -e_0 w = -(y - y_{ref}) w \quad (5-13)$$

Nach Umstellung mit den Variablen der Formel (5-13) wird der Parametergradient zu:

$$\begin{pmatrix} \dot{\hat{p}}_1 \\ \dot{\hat{p}}_2 \\ \dot{\hat{p}}_3 \\ \dot{\hat{p}}_4 \end{pmatrix} = -(y - y_{ref}) \begin{pmatrix} -x_1^2 \\ -\sin(x_1) \\ x_1 \frac{p_1^{(1)} \cdot x_1^2 + p_2^{(1)} \cdot \sin(x_1) - v}{p_1^{(2)} \cdot x_1 + p_2^{(2)} \cdot x_1^3} \\ x_1^3 \frac{p_1^{(1)} \cdot x_1^2 + p_2^{(1)} \cdot \sin(x_1) - v}{p_1^{(2)} \cdot x_1 + p_2^{(2)} \cdot x_1^3} \end{pmatrix} \quad (5-14)$$

Das Regelgesetz lautet:

$$u = \frac{1}{\sum_{j=1}^2 p_j^{(2)} L_{g_j} h(x)} \left(- \sum_{i=1}^2 p_i^{(1)} L_{f_i} h(x) + \dot{y}_{ref} + \mathbf{a}(y_{ref} - y) \right) \quad (5-15)$$

Nach Umsetzung gilt folgende Formel:

$$u = \frac{-p_1^{(1)} \cdot 1 \cdot x_1^2 - p_2^{(1)} \cdot 1 \cdot \sin(x_1) + \dot{y}_{ref} + \mathbf{a}(y_{ref} - y)}{p_1^{(2)} \cdot 1 \cdot x_1 + p_2^{(2)} \cdot 1 \cdot x_1^3} \quad (5-16)$$

oder

$$u = \frac{-p_1^{(1)} x_1^2 - p_2^{(1)} \sin(x_1) + \dot{y}_{ref} + \mathbf{a}(y_{ref} - y)}{p_1^{(2)} x_1 + p_2^{(2)} x_1^3} \quad (5-17)$$

Hier sieht man das Blockschaltbild des getesteten Systems beim Simulink.
Als Eingabe ist ein Sinussignal eingegeben worden.

5.2. Test der approximativen Linearisierungsmethode mittels neuronaler Netze

In diesem Kapitel werden zunächst Generalisierungsprobleme beschrieben, und Ferner wird ein nichtlineares System sowohl analytisch als auch mittels neuronaler Netze und Interpolationsmethoden getestet. Die Ergebnisse sind graphisch dargestellt, und die Unterschiede zwischen analytischem Ergebnis und dem Ergebnis der neuronalen Netze sind graphisch dargestellt worden.

5.2.1. Generalisierung

Nach der Trainingsphase, bei der praktischen Anwendung des neuronalen Netzes, ist es möglich, dass das neuronale Netz mit anderen Daten als den trainierten Daten eingespeist wird.

Das ist z.B. der Fall, wenn die Daten verrauscht sind, oder wenn ein nicht sauberes analoges Signal dem neuronalen Netz eingegeben wird, oder wenn bei dem neuronalen Netz nur die wichtigste Charakteristik vom Signal trainiert ist.

Die Wahrscheinlichkeit, dass nicht alle Punkte von solchen Signalen oder Kurven den neuronalen Netzen bekannt sind, ist groß. Solche Punkte können für das neuronale Netz verwirrend sein, wenn das neuronale Netz nur auf trainierte Daten spezialisiert ist.

Unter Umständen reagiert das neuronale Netz auf die eingegebenen Daten nicht richtig.

Um dieses Problem zu überwinden, braucht man ein neuronales Netz mit Generalisierungseigenschaften. Das neuronale Netz muss über eine Vorhersagefähigkeit verfügen, um so in der Lage zu sein das Problem richtig und robust zu behandeln.

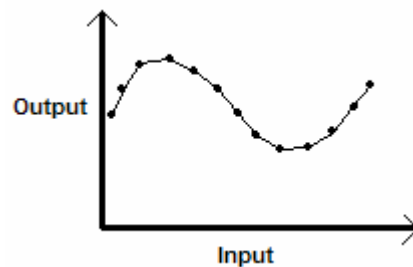


Abbildung 5.3: Gute Generalisierung.

Wenn ein Netz mit mehreren Mustern trainiert wird, dann speichert das Netz die trainierten Daten. Es wird vielleicht einige Merkmale von trainierten Daten lernen, aber nicht die Eigenschaften von der modellierten Funktion.

Das nennt man overfitting oder overtraining. Wenn das neuronale Netz übertrainiert wird, dann verliert es die Fähigkeit ähnliche Pattern zu generalisieren. Dieser unerwünschte Effekt kann auch auftauchen, wenn die neuronalen Netze mehr Neuronen als notwendig in der verdeckten Schicht haben. Dann kann der Rauschanteil des Inputspattern in den Gewichten der neuronalen Netze gespeichert werden.

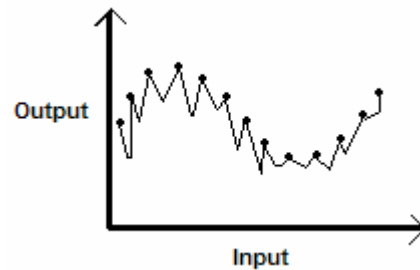


Abbildung 5:4: Schwache Generalisierung (overfitting oder overtraining).

Die Generalisierung wird von drei Faktoren beeinflusst. Ein Faktor ist die Größe des Training Sets, ein weiterer Faktor umfasst die Bildung des neuronalen Netzes, und ein anderes Element beinhaltet die physikalische Komplexität von vorhandenen Problemen.

5.2.2. Cross Validation

Bei der Realisierung der Simulation mittels neuronaler Netze kann das Problem der schwachen Generalisierung auftreten. Daher wird Cross Validation angewendet, um das Overfitting zu vermeiden. (siehe [Nelle 01], [Hayki 99])

Bei der Cross Validation können die neuronalen Netze besser trainiert werden. Das neuronale Netz lernt genug von der Vergangenheit, bis es die Zukunft besser generalisieren kann.

Um dieses Ziel zu erreichen, müssen die Daten in drei Teile aufgeteilt werden. Ein Teil ist das Training Set, ein weiterer Teil das Validation Set und der dritte Teil ist das Test Set.

Bei der Cross Validation ist die Datenmenge in n Anteile geteilt worden. Die $n - 1$ Anteile werden zufällig zum Training ausgewählt, der restliche Anteil wird zum Test (Validieren) verwendet. Dieser Ablauf wiederholt sich bei allen möglichen Kombinationen der n Anteile der Datenmenge.

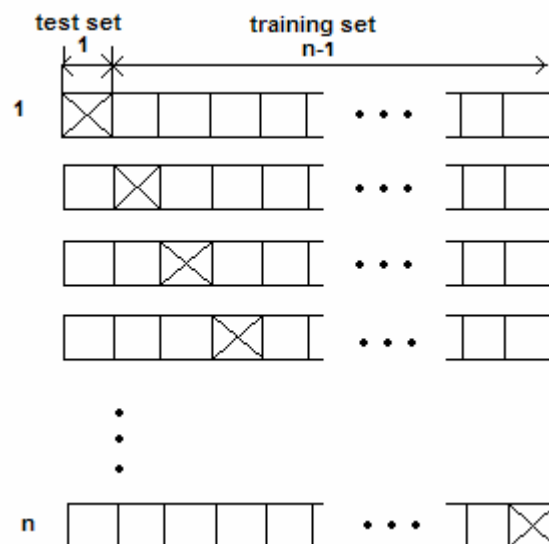


Abbildung 5:5: cross validation.

5.2.3. Regulierung der frühzeitigen Einstellung des Trainings

Es ist schwierig festzustellen, wann ein Training für eine gute Generalisierung eingestellt werden muss. Wenn das Training zum richtigen Zeitpunkt nicht gestoppt wird, kann es passieren, dass das neuronale Netz übertrainiert wird. Dies ist unabdingbar, wenn eine iterative Optimierung angewendet wird. Der Training Algorithmus checkt periodisch mittels Validation Set das neuronale Netz.

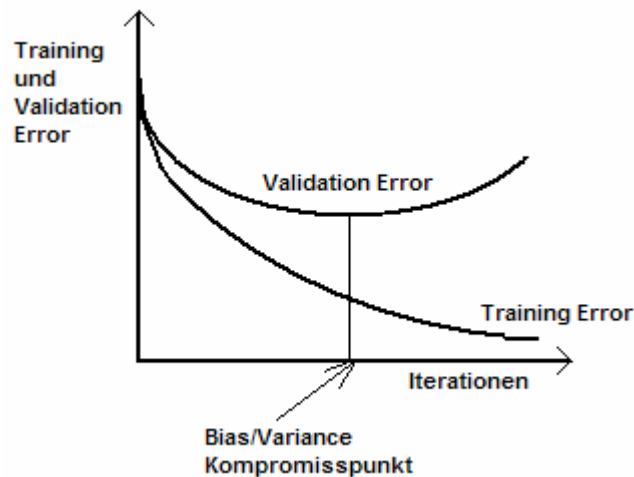


Abbildung 5:6: Bias Varianz Kompromisspunkt.

Hier werden die neuronalen Netze mit dem Training Set normal trainiert. Nach jedem Trainingsdurchlauf wird auch die Qualität des trainierten Netzes mit Validation Error gemessen und überprüft. Wenn die Validation Phase vollständig ist, geht das Training weiter.

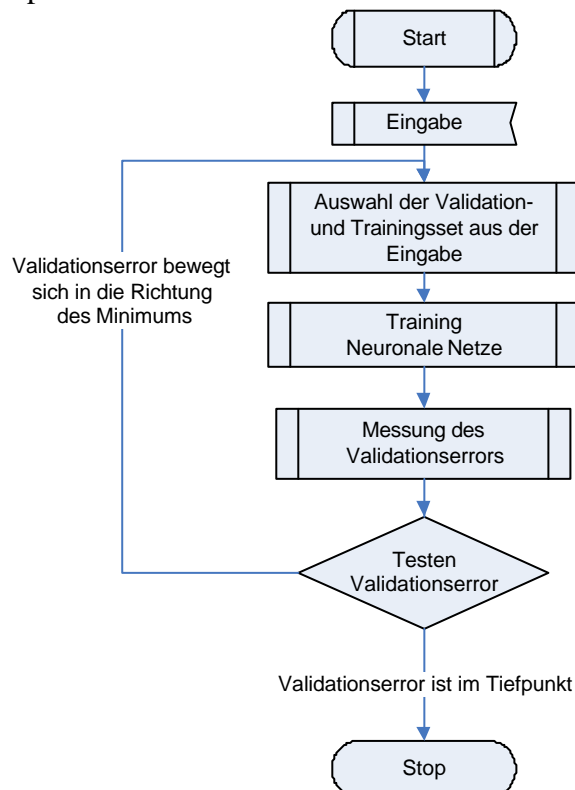


Abbildung 5:7: Frühzeitige Einstellung des Trainingsalgorithmus.

Wie in Abbildung 6:6 dargestellt ist, teilt der Bias Varianz Kompromisspunkt die waagerechte Achse in zwei Teile. Die rechte Seite zeigt ein Overfitting - die linke Seite ein Underfitting. Der optimale Punkt ist dort, wo das Minimum des Validationerrors ist. Somit, wird das Overfitting vermieden, wenn nach den Iterationen ein Kompromisspunkt eingestellt wird,

5.2.4. Auswahl der neuronalen Netze

Unter einer Netzauswahl versteht man das Trainieren mehrerer neuronaler Netze mit unterschiedlicher Topologie, aber gleichen Trainingsdaten. Jedes neuronale Netz wird mit Validationsdaten gecheckt, das beste Netz wird selektiert. Schließlich wird die Qualität der neuronalen Netze noch einmal mit Test Daten überprüft. (siehe [Nelle 01])

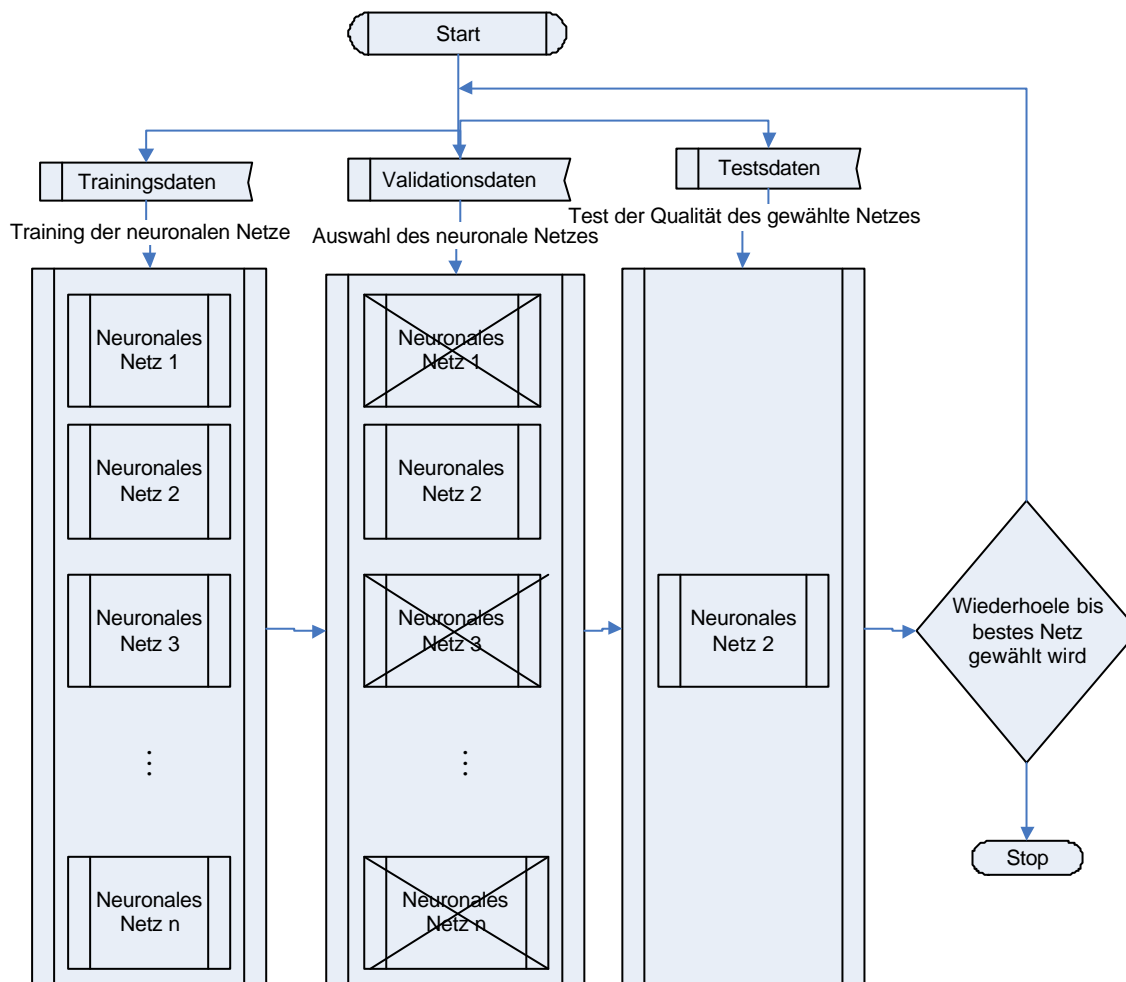


Abbildung 5:8: Auswahl des neuronalen Netzes.

Nach der Auswahl der besten Topologie, kann man auch die optimale Anzahl der Datensätze untersuchen. Dies ist notwendig, die Zeitspanne für den Ablauf des Algorithmus zu verringern. Die Anzahl der Datensätze muss solange geändert werden, bis die optimale Datenmenge gefunden wird.

5.2.5. Test

Zum Test der Linearisierungsmethode von den neuronalen Netzen wurde mit der folgendem Differentialgleichung der zweiten Ordnung gearbeitet.

$$\dot{x}_1 = x_1^2 + x_2 \quad (5-18)$$

$$\dot{x}_2 = u \quad (5-19)$$

$$y = x_1 \quad (5-20)$$

Die Linearisierung dieser Differentialgleichung wird zunächst analytisch beschrieben, dann wird dieselbe Linearisierung vom neuronalen Netz durchgeführt. Die analytische Beschreibung ist notwendig. Einerseits soll der Argumentwertbereich x und der Definitionsbereich der Eingabe ermittelt werden, andererseits sollen exakte Ergebnisse sichtbar sein, um einen Vergleich zwischen den Ergebnissen zu ermöglichen.

5.2.5.1. Analytische Berechnung und Simulation

Zunächst werden die verschiedenen Funktionen der Zustandsraumdarstellung von den Formeln (5-18), (5-19) und (5-20) in eine Matrizenform gebracht.

Eine Systemmatrix ist:

$$f(x) = \begin{pmatrix} x_1^2 + x_2 \\ 0 \end{pmatrix} \quad (5-21)$$

Eine Steuermatrix ist:

$$g(x) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (5-22)$$

Eine Ausgangsmatrix ist gleich:

$$h(x) = \begin{pmatrix} x_1 \\ 0 \end{pmatrix} \quad (5-23)$$

$$h(x) = x_1 \quad (5-24)$$

So werden die Linearisierungseinheiten mit Hilfe der Lie-Ableitung und mit Hilfe des Konzepts von Kapitel eins berechnet:

$$L_f h(x) = \begin{pmatrix} \frac{\partial h(x)}{\partial x_1} \\ \frac{\partial h(x)}{\partial x_2} \end{pmatrix}^T \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}^T \begin{pmatrix} x_1^2 + x_2 \\ 0 \end{pmatrix} = x_1^2 + x_2 \quad (5-25)$$

$$L_f^2 h(x) = L_f (L_f h(x)) = \begin{pmatrix} \frac{\partial (L_f h(x))}{\partial x_1} \\ \frac{\partial (L_f h(x))}{\partial x_2} \end{pmatrix}^T \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} 2x_1 \\ 1 \end{pmatrix}^T \begin{pmatrix} x_1^2 + x_2 \\ 0 \end{pmatrix} = 2x_1^3 + 2x_1 x_2 \quad (5-26)$$

$$L_g (L_f h(x)) = \begin{pmatrix} \frac{\partial (L_f h(x))}{\partial x_1} \\ \frac{\partial (L_f h(x))}{\partial x_2} \end{pmatrix}^T \begin{pmatrix} g_1(x) \\ g_2(x) \end{pmatrix} = \begin{pmatrix} 2x_1^2 \\ 1 \end{pmatrix}^T \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1 \quad (5-27)$$

Zur Bestimmung der Pollstellen werden Determinante $(SI - bk^T - A)$ berechnet und gleich Null gesetzt:

$$\Delta = \det(SI - bk^T - A) = \det \left(\begin{pmatrix} S & S \\ S & S \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} k_1 & k_2 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right) \quad (5-28)$$

d.h.

$$\Delta = \det \left(\begin{pmatrix} S & 0 \\ 0 & S \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ k_1 & k_2 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right) = \det \begin{pmatrix} S & -1 \\ -k_1 & S - k_2 \end{pmatrix} \quad (5-29)$$

Die Wurzel von Δ soll aus der linken Halbebene gewählt werden, zunächst wird mittels $\Delta = 0$ versucht eine Relation für die Wurzel zu finden, d.h.

$$\det \begin{pmatrix} S & -1 \\ -k_1 & S - k_2 \end{pmatrix} = 0 \quad (5-30)$$

$$S^2 - k_2 S - k_1 = 0 \quad (5-31)$$

Nun versucht man eine Relation zwischen Reglerkoeffizient und Pollstellen zu finden.

$$(S + a)(S + b) = S^2 + (a + b)S + a \cdot b \equiv S^2 - k_2 S - k_1 \quad (5-32)$$

Die Relation ist $a + b = -k_2$ und $a \cdot b = -k_1$

Danach werden die Pollen gewählt, d.h., es werden zwei Werte für a und b gewählt, somit wird $(S + a) = 0$ und $(S + b) = 0$ gleich Null.

$$(S + a) = 0, \quad S = -a = -\frac{1}{2} + j\frac{1}{2}, \quad a = \frac{1}{2} - j\frac{1}{2} \quad (5-33)$$

Für den zweiten Poll gilt:

$$(S + b) = 0, \quad S = -b = -\frac{1}{2} - j\frac{1}{2}, \quad b = \frac{1}{2} + j\frac{1}{2} \quad (5-34)$$

Wie man sieht, liegen beide Pollen auf der linken Halbebene.

Jetzt werden $a + b$ und $a \cdot b$ berechnet und die Reglerkoeffizienten ermittelt:

$$a + b = -k_2 = 1 \quad \text{also} \quad k_2 = -1 \quad (5-35)$$

$$a \cdot b = -k_1 = \frac{1}{2} \quad \text{also} \quad k_1 = -\frac{1}{2} \quad (5-36)$$

Das Regelgesetz ohne Tracking ist gleich:

$$u = \frac{1}{L_g L_f^{2-1} h(x)} \left(-L_f^2 h(x) + k_1 h(x) + k_2 L_f^1 h(x) + v_{new} \right) = \frac{1}{1} \left((2x_1^3 + 2x_1 x_2) - \frac{1}{2} x_1 - (x_1^2 + x_2) + v_{new} \right) \quad (5-37)$$

Das Regelgesetz mit Tracking ist gleich:

$$u = \frac{1}{L_g L_f^{2-1} h(x)} \left(\ddot{v}_{ref} - L_f^2 h(x) + k_1 (\dot{v}_{ref} - h(x)) + k_2 (\dot{v}_{ref} - L_f^1 h(x)) \right)$$

$$u = \frac{1}{1} \left(\ddot{v}_{ref} - (2x_1^3 + 2x_1 x_2) - \frac{1}{2} (\dot{v}_{ref} - x_1) - (\dot{v}_{ref} - (x_1^2 + x_2)) \right) \quad (5-38)$$

5.2.5.2. Allgemeiner Simulationsalgorithmus

Hier wird versucht, einen Algorithmus zur Anwendung der approximativen Methoden darzustellen, wobei der Algorithmus in einer allgemeinen Form sowohl für neuronale Netze als auch für Interpolationseinheiten gilt. Wie man sieht, werden die Streckendaten extrahiert,

und dementsprechende neuronale Netze trainiert. Die beiden Vorgänge werden unter M-File simuliert. Schließlich werden sie zur Linearisierung und Regelung an den Simulink weitergegeben. Auf der Simulinkebene wird ein Blockschaltbild abgebildet. Mit Hilfe dieses Blockschaltbildes wird die Linearisierungsregelung und das Tracking der nicht linearen Strecke getestet.

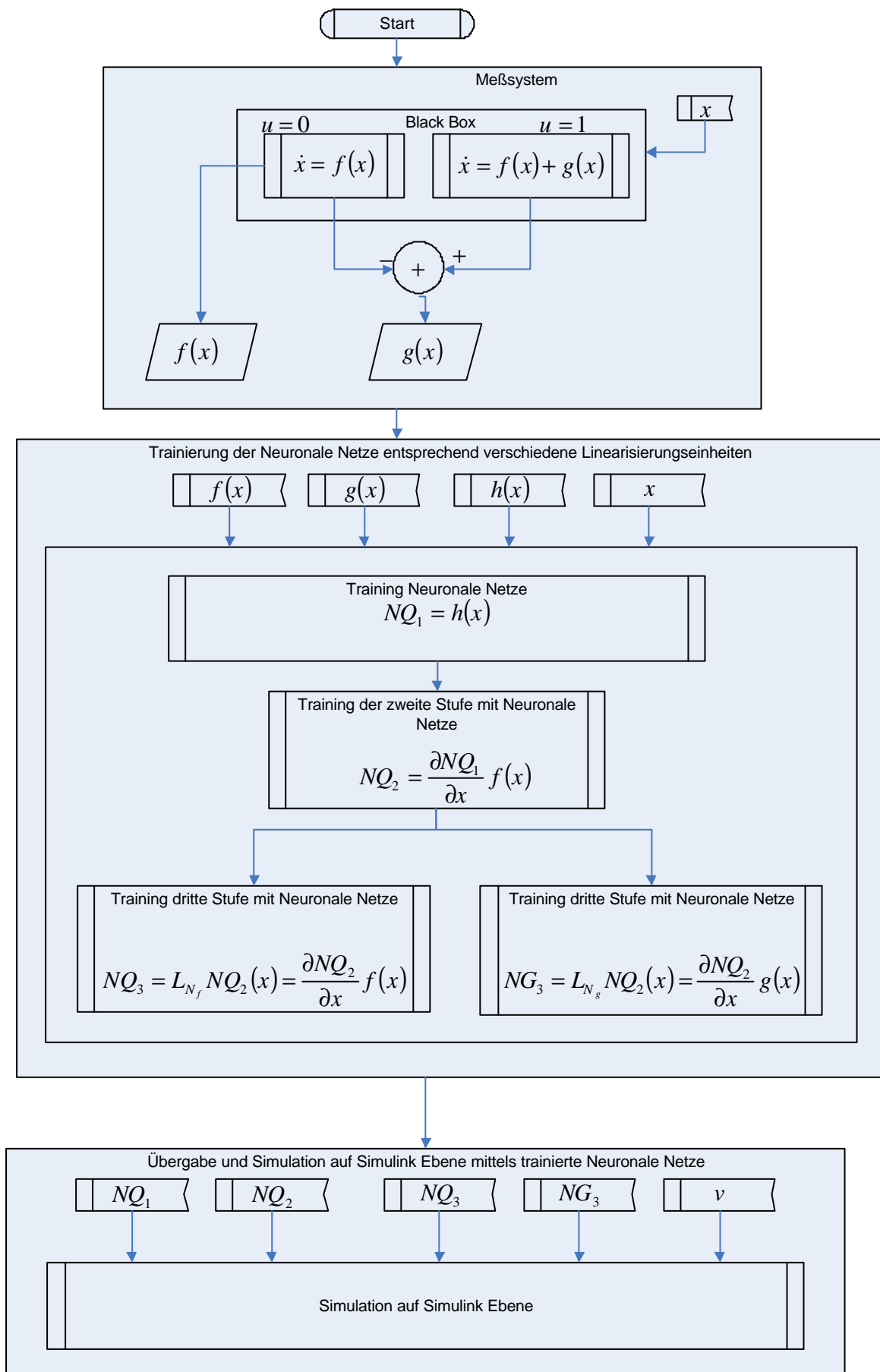


Abbildung 5:9: Allgemeiner Simulationsalgorithmus mittels neuronale Netze.

5.2.5.3. Approximation mit Hilfe neuronaler Netze

Abbildung 5:10 zeigt das Blockschaltbild mit dem vorgegebenen System der Formeln (5-18) (5-19) und (5-20) unter Simulink. Hier handelt sich um die Linearisierung und Regelung der nichtlinearen Systeme mit stabiler Nulldynamik. Das simulierte System besteht aus zwei Teilen. Der obere Teil ist der simulierte Teil, der untere Teil ist die exakte Lösung. Dies bedeutet, dass zwei Ergebnisse vorhanden sind, eines ist die exakte Lösung - das andere die approximative Lösung der neuronalen Netze. Die Differenzen der Anteile sind auch in den Abbildungen 5:11 und 6:12 als Error bezeichnet worden. Die Ergebnisse gelten für zwei verschiedene Eingaben – die Sprungantwort und das Sinussignal. Die neuronalen Netze sind schon unter M-File gemäß den theoretischen Grundlagen des Kapitels 4 programmiert worden. Hier sind sie wieder extrahiert und genutzt worden.

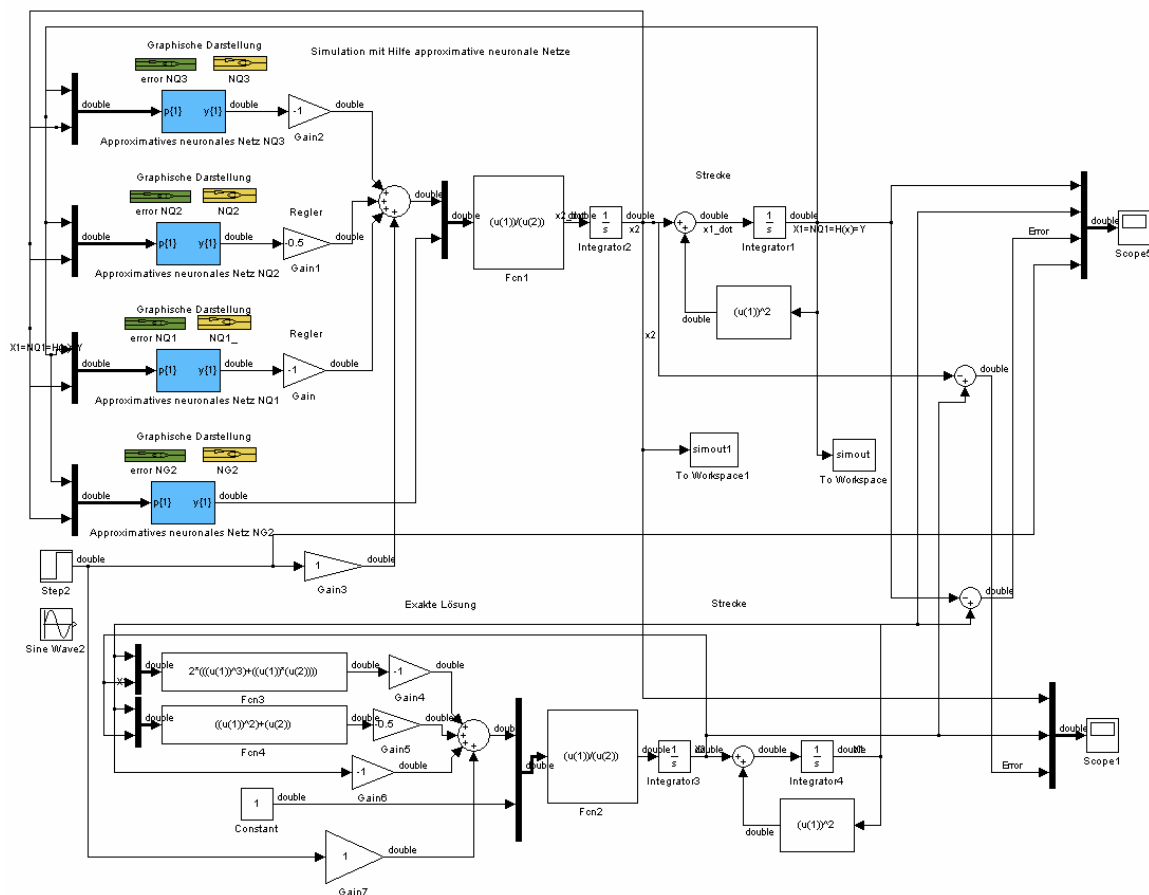


Abbildung 5:10: Approximative Linearisierung und Regelung mit neuronalen Netzen.

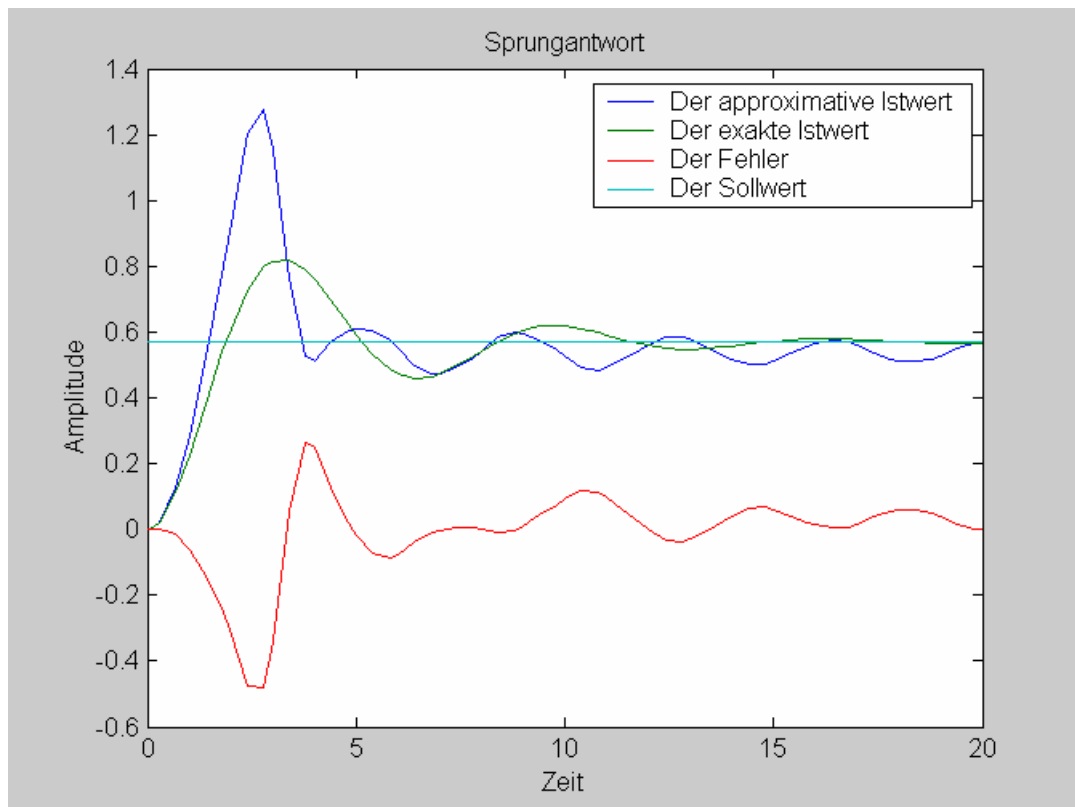


Abbildung 5:11: Sprungantwort des approximativen linearisierten Systems mit neuronalen Netzen und Regelung.

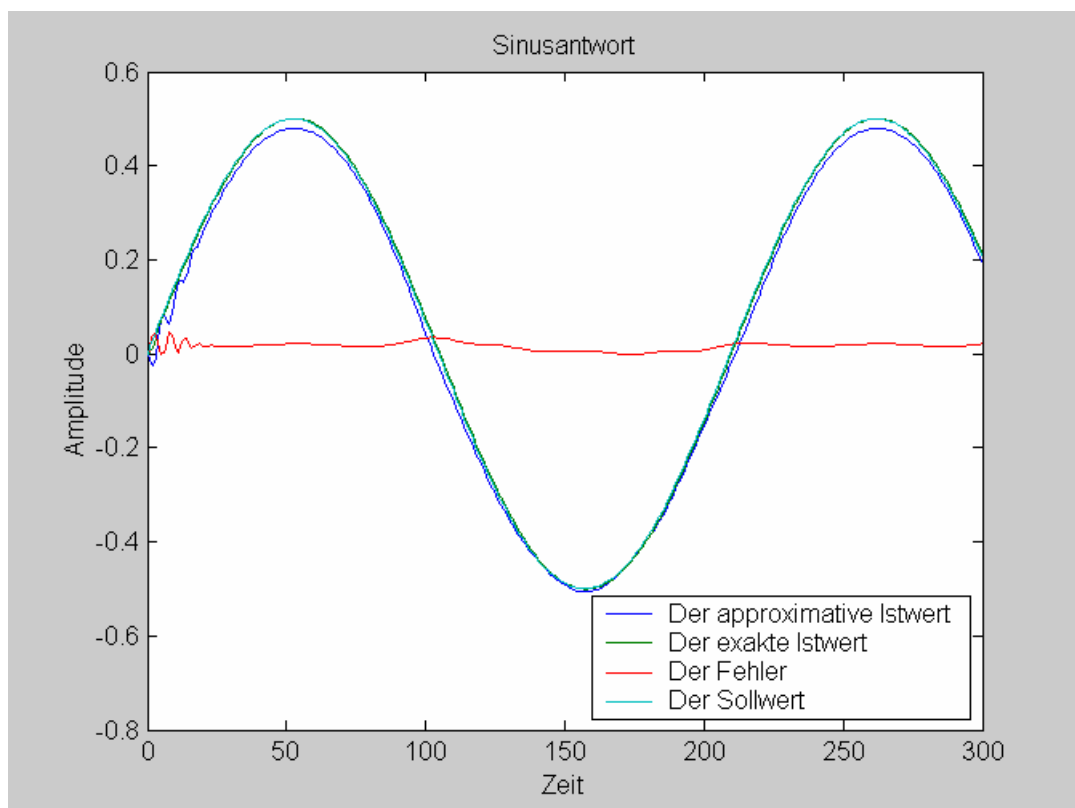


Abbildung 5:12: Sinusantwort des approximativen linearisierten Systems mit neuronalen Netzen und Regelung.

Jetzt wird weiter versucht, wieder dieselbe Strecke mit approximativer Linearisierung mit Tracking mittels neuronaler Netze zu testen. D.h., ein exaktes Ergebnis wird mit einem

approximativem Ergebnis verglichen, der Error oder die Differenz von den beiden wird in der Abbildung 6:14 dargestellt. Als Eingabe ist hier das Sinussignal gewählt worden.

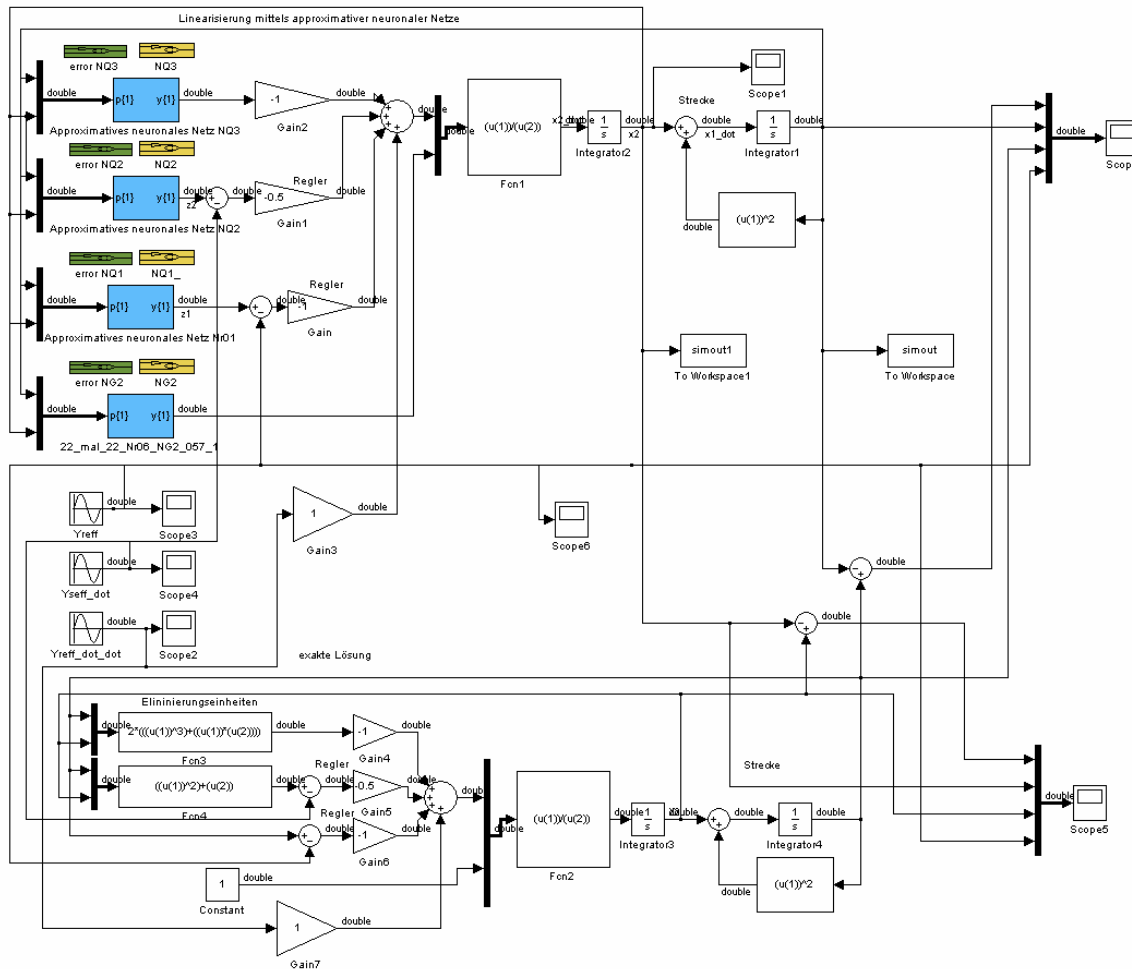


Abbildung 5:13: Approximative Linearisierung, Regelung und Tracking mittels neuronaler Netze.

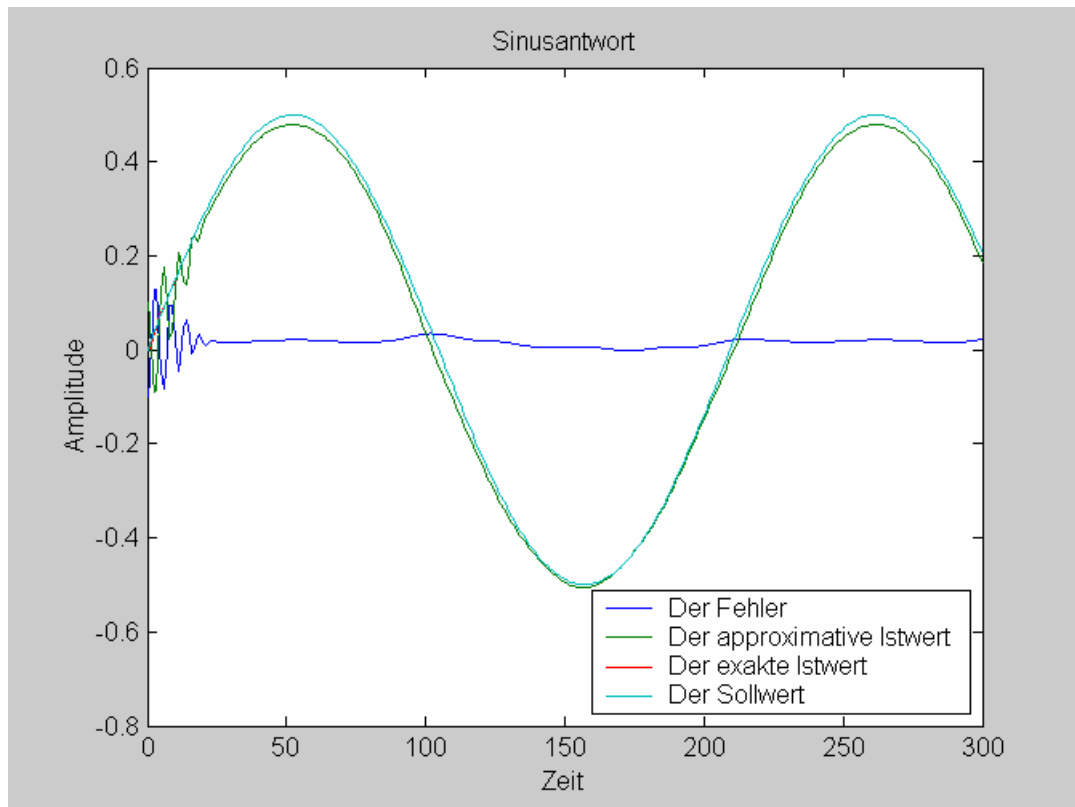


Abbildung 5:14: Sinusantwort der approximativen Linearisierung, Regelung und Tracking mittels neuronaler Netze.

5.2.5.4. Approximation mit Hilfe der Interpolation

In diesem Abschnitt wird versucht, die ganzen Vorgänge noch einmal mit Interpolationseinheiten mit Hilfe der gleichen Methoden wie in Kapitel 4 zu testen.

Zunächst wird die Linearisierung und Regulierung durchgeführt, und zwei verschiedene Eingänge - nämlich Sprungantwort und Sinusantwort - getestet.

Die Interpolationseinheiten sind mittels Matlab Funktion und M-File generiert worden. Sie werden dann wieder auf Simulinksebene abgerufen und angewendet.

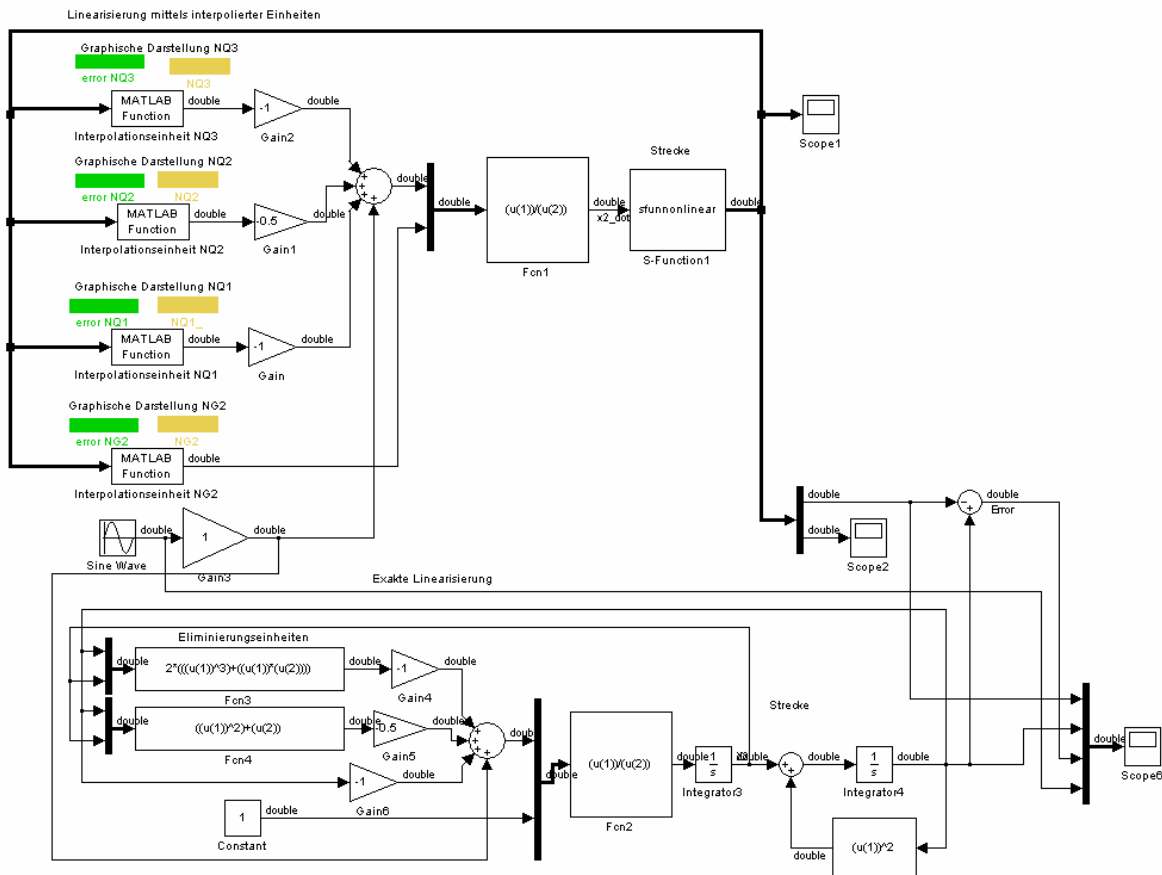


Abbildung 5:15: Approximative Linearisierung und Regelung mit Interpolation.

In den Abbildungen 5:16 und 5:17 ist die Sprungantwort und Sinusantwort für ein linearisiertes geregeltes System mittels approximierter Interpolationseinheit dargestellt worden. Der Fehler ist auch jeweils als Differenz zwischen exakter und approximierter Lösung dargestellt worden.

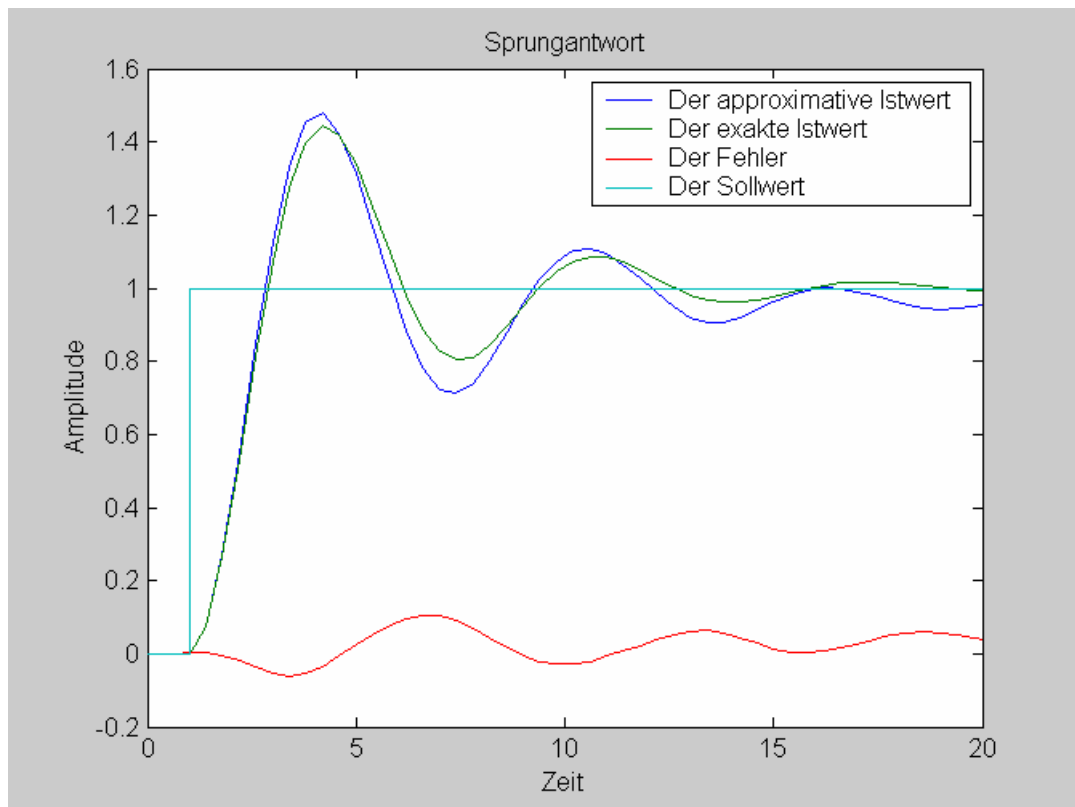


Abbildung 5:16: Sprungantwort der approximativen Linearisierung und Regelung mit Interpolation.

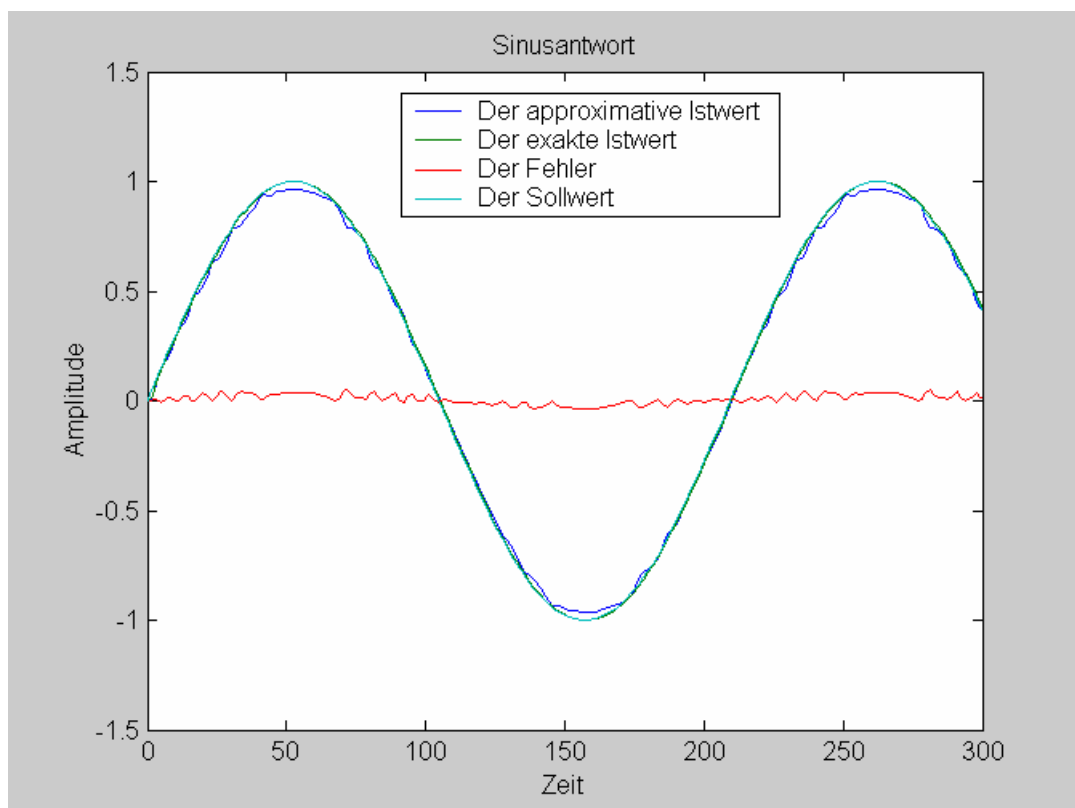


Abbildung 5:17: Sinusantwort der approximativen Linearisierung und Regelung mit Interpolation.

Die gleiche Methode wird beim Tracking angewendet. Als Eingabe ist das Sinussignal genutzt worden, weil es leicht ableitbar ist.

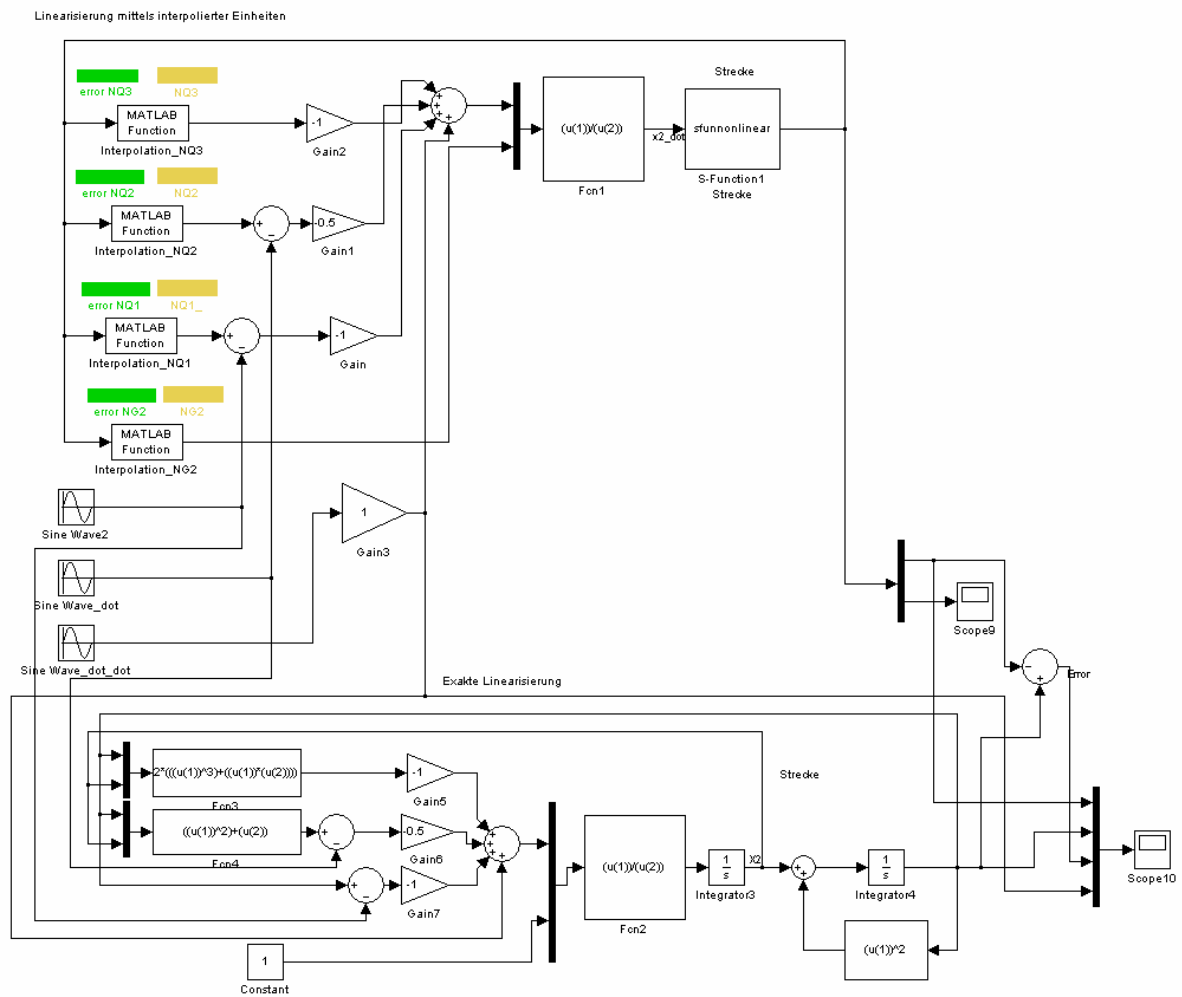


Abbildung 5:18: approximative Linearisierung, Regelung und Tracking mit Interpolation.

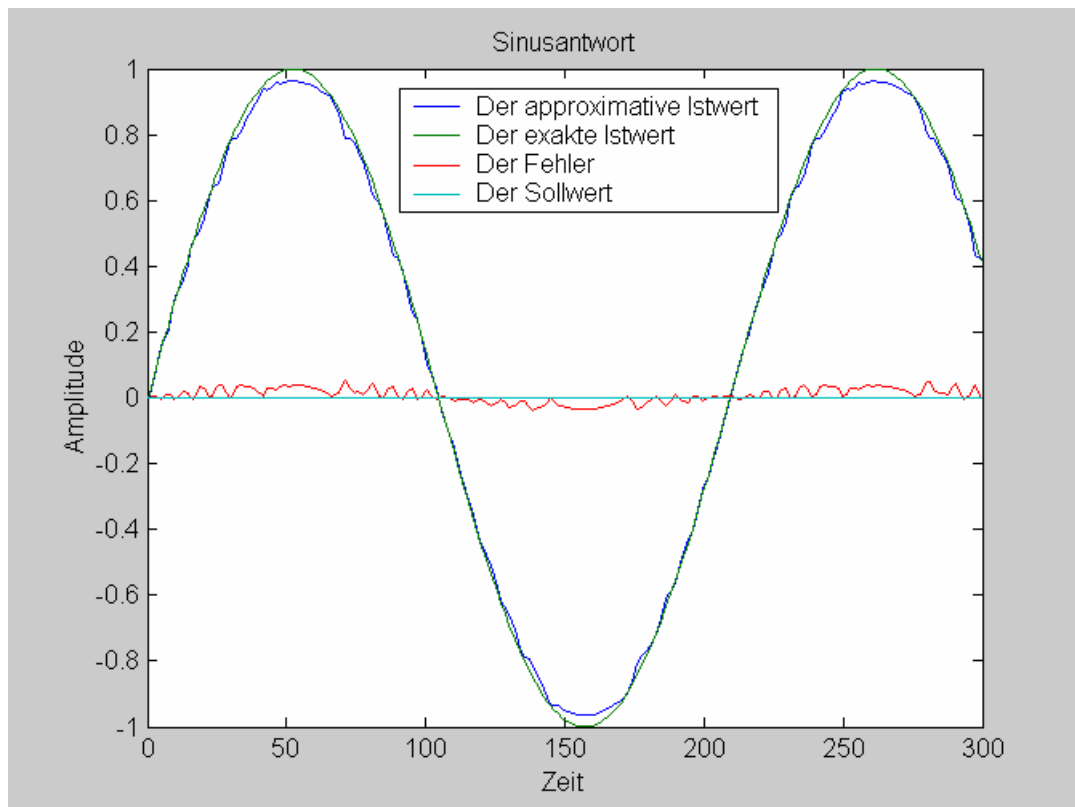


Abbildung 5:19: Sinusantwort von der geregelten approximativen Linearisierung mit Interpolation und Tracking.

5.2.5.5. Graphische Darstellung der Fehlerebenen

Zur Beurteilung und Beobachtung der Qualität der simulierten Einheiten, muss man die simulierten Einheiten und Differenzen zwischen der simulierten und echten Größe darstellen. Diese kann man optisch leicht hinsichtlich der Qualität ihrer approximierten Fläche überprüfen.

Man kann sehen, ob diese Fläche über eine ausreichende Glättung verfügt. Aber auch der Verlauf der Fehlergröße kann bestimmt werden.

Die Eingabe(Zustandsvariable) oder die Zustandsgrößen bilden eine zweidimensionale zugeordnete Koordinate, wobei die dritte senkrechte Dimension als Ausgang dient.

Das Testobjekt ist wie nach wie vor mit den Zuständen $\dot{x}_1 = x_1^2 + x_2$, $\dot{x}_2 = u$ und der Ausgangsgröße $y = x_1$ bekannt.

Der Test ist für zwei verschiedene Approximationstypen, nämlich die neuronalen Netze und die Interpolation durchgeführt worden. Für unser geschildertes Beispiel(Testobjekt) sind vier Bausteine für ein approximiertes Regelgesetz notwendig. Hier wird als Beispiel nur der letzte Baustein oder NQ_3 graphisch präsentiert, um die Fehlerdimension darzustellen.

5.2.5.5.1. Neuronale Netze

Hier sieht man den simulierten Ausgang von NQ_3 mit den neuronalen Netzen. Diese Simulation ist nur für einen bestimmten Bereich gültig. Eine unberechenbare Eingabe außerhalb des definierten Bereichs für neuronale Netze könnte chaotische Ergebnisse

verursachen. D.h., das approximative linearisierte System darf nur in trainiertem, definiertem Bereich in Betrieb gestellt werden.

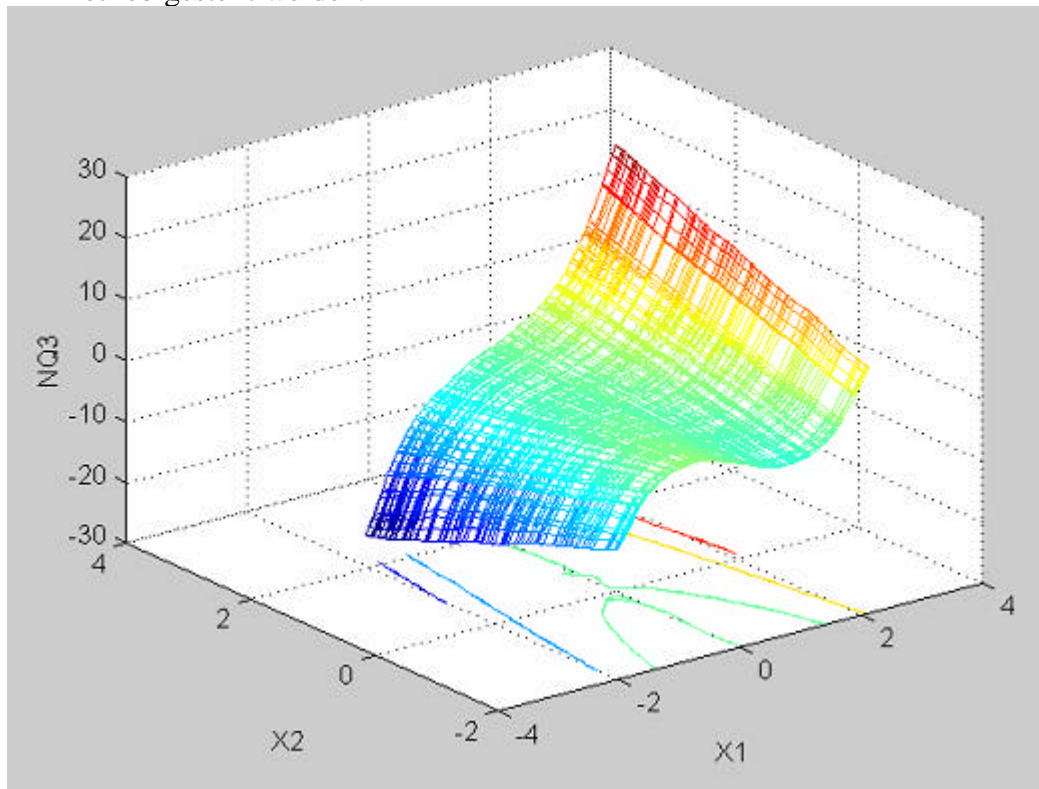


Abbildung 5:20: Approximative simulierte Einheit NQ3 mit neuronalen Netzen.

Hier ist lediglich die Fehlergröße dargestellt. Diese Größe ist die Subtraktion zwischen exakter und simulierter Lösung.

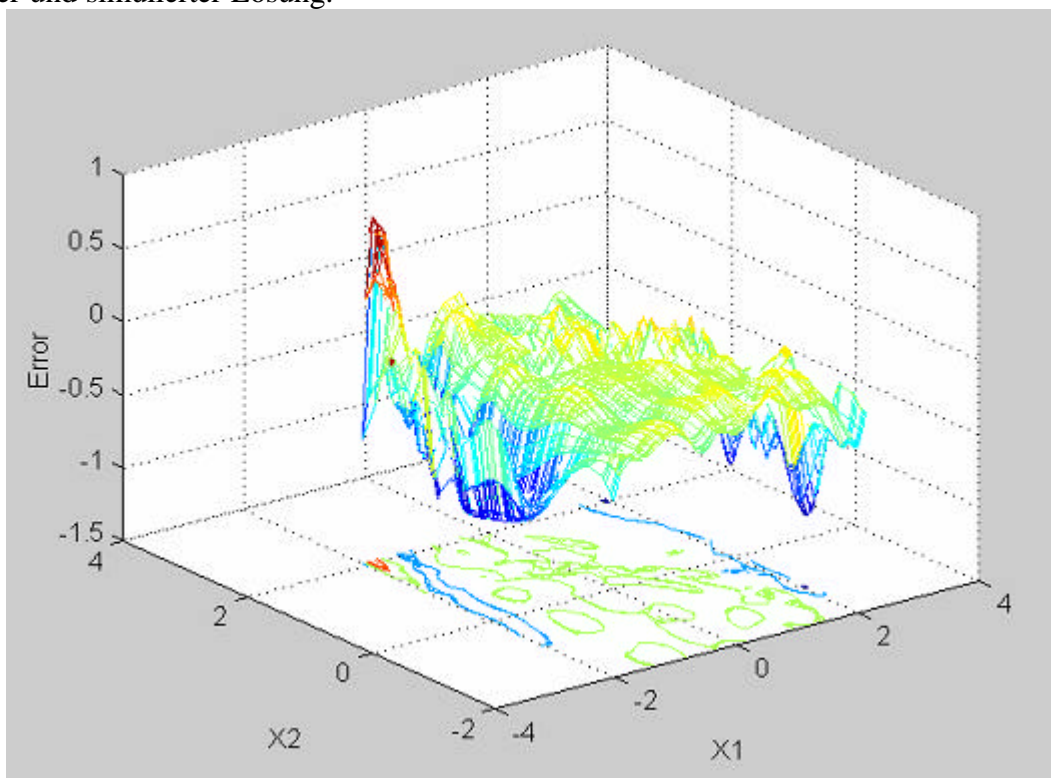


Abbildung 5:21: Darstellung der Fehler oder Differenz zwischen exakter Größe und approximativer simulierter Größe bei neuronalen Netzen.

5.2.5.5.2. Interpolation

Die gleiche Funktion, nämlich NQ_3 , ist diesmal mit Interpolationseinheit simuliert worden, und dadurch sind die gleichen Bilder mit unterschiedlichem Verlauf extrahiert worden. D.h., wie o.a. ist zunächst NQ_3 selbst dargestellt worden. Man sieht, dass die approximative simulierte Fläche scheinbar eine glatte Oberfläche besitzt.

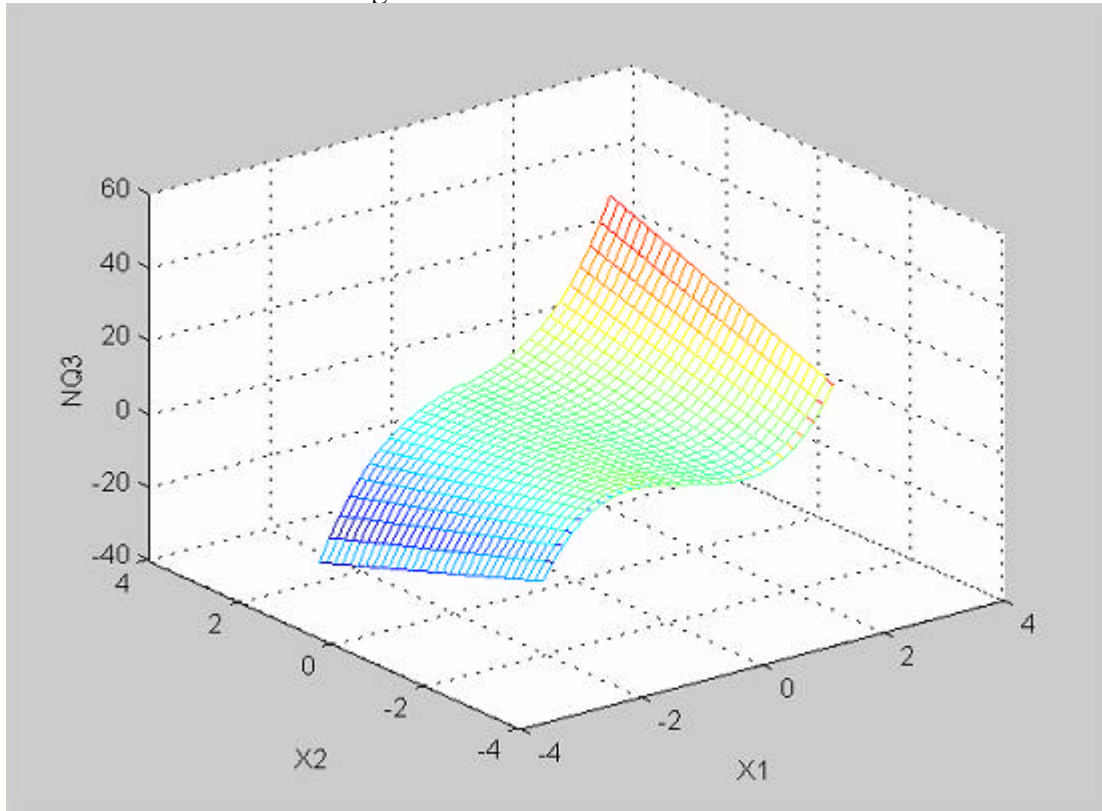


Abbildung 5:22: Darstellung der approximative simulierten Einheit NQ_3 mit Interpolationsmethode.

In der Abbildung 5:23 kann man die Fehlergröße sehen.

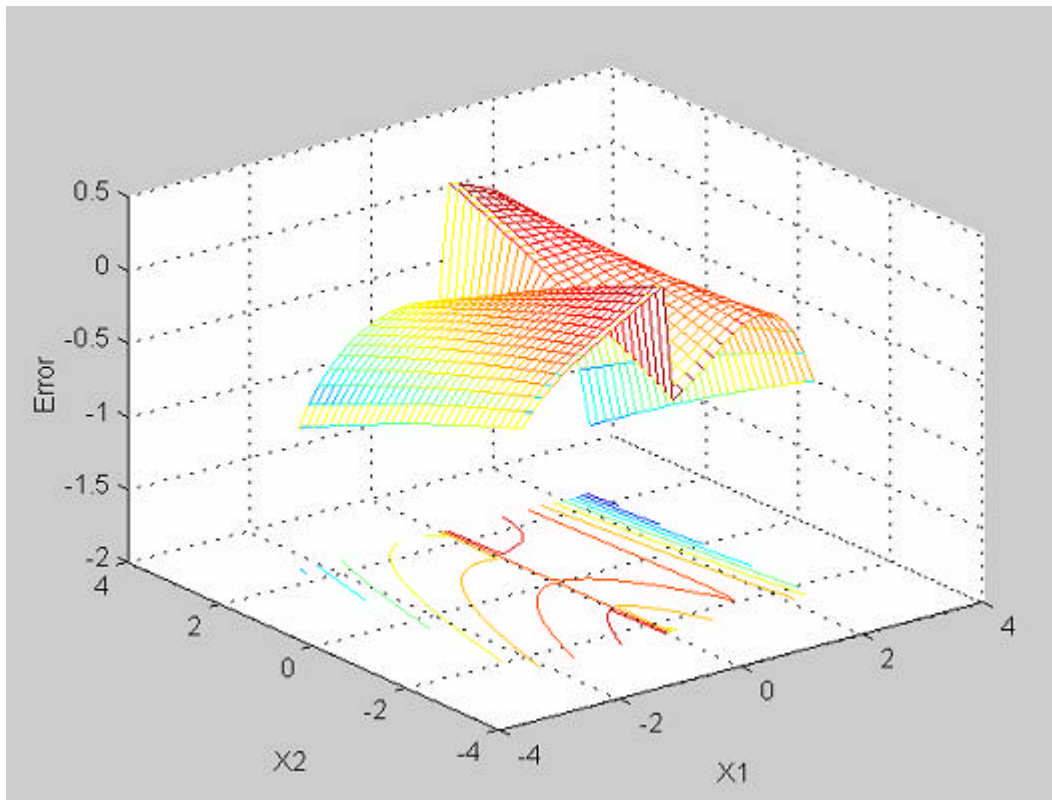


Abbildung 5:23: Darstellung der Fehler oder Differenz zwischen exakter Größe und approximativ simulierter Größe beim Interpolationsmethode.

6. Literaturverzeichnis

[He 98] A Neural Approach for Control of Nonlinear Systems with Feedback Linearization. Shouling He, Konrad Reif, Rolf Unbehauen, Life Floww. 1998 IEEE.

[SASTR 89] Adaptive Control, Stability, Convergence, and Robustness. Shanker Sastry, Marc Bodson. 1989 Prentice-Hall International Editions.

[Schwa 91] Nichtlineare Regelungssysteme, Systemtheoretische Grundlagen. Professor Dr.-Ing. Helmut Schwarz. Lehrstuhl für Meß-, Steuer- und Regelungstechnik der Universität-GH- Duisburg. 1991 R. Oldenbourg Verlag München Wien.

[Marin 95] Nonlinear Control Design, Geometrie, Adaptive, Robust. Riccardo Marino, Tomei Patrizio. 1995 Prentice Hall information and System sciences series

[Jacqu 91] Applied Nonlinear Control. Jean Jacques E. Slotine. Massachusetts Institute of Technology, Weiping Li Massachusetts Institute of Technology. 1991 Prentice-Hall, Inc.

[Weinm 95] Regelungen, Analyse und technischer Entwurf Band 2, Multivariable, digitale und nichtlineare Regelungen, optimale und robuste Systeme. o.Univ.Prof.Dipl.-Ing.Dr.techn. Alexander Weinmann. Lehrstuhl für Regelungstechnik und Automatisierung der technischen Universität Wien. 3., überarbeitete und erweiterte Auflage 1995 Springer-Verlag.

[Nelle 01] Nonlinear System Identification, for classical approaches to neural networks and fuzzy models. Dr. Oliver Nelles. 2001 Springer-Verlag.

[Braus 95] Neuronale Netze, eine Einführung in die Neuroinformatik. Dr. rer. nat. Rüdiger Brause Universität Frankfurt/Main. 2. überarbeitete und erweiterte Auflage. 1995 B.G. Teubner Stuttgart.

[Hayki 99] Neural Networks a Comprehensive Fundation. Simon Haykin, McMaster University, Hamilton, Ontario, Canada. Secound Edition. 1999 Prentice Hall.

[Cherk 98] Lerning From Data. Concepts, Theory, and Methods. Vladimir Cherkassky faculty of electrical and computer engineering at University of Minnesota. Filip Mulier PhD in electrical engineering from University of Minnesota. 1998 John Wiley & Sons, Inc.

[Patte 97] Künstliche neuronale Netze, das Lehrbuch. Dan W. Patterson. 1997 der deutschen Ausgabe by Prentice Hall Verlag GmbH.

[Nijme 95] Nonlinear Dynamical Control Systems. Henk Nijmeijer, Arjan van der Schaft. Department of Applied Mathematics, University of Twente. 1990 Springer-Verlag New York Inc.

[Weing 01] Neurale Netze. Dr.-Ing. Andreas Weingessel. Das Skriptum der Vorlesung Neuronale Netze. 2001 technischer Universität Wien.