Direkte Approximation optimaler Wertefunktionen durch Perturbationsmethoden höherer Ordnung

Diplomarbeit

von

Christian Spieler

FAKULTÄT FÜR MATHEMATIK, PHYSIK UND INFORMATIK

MATHEMATISCHES INSTITUT

Datum: 29. Juli 2008

Aufgabenstellung / Betreuung:
Prof. Dr. L. Grüne

Danksagung

An dieser Stelle möchte ich mich bei Herrn Professor Dr. Grüne für die interessante Themenstellung und die sehr gute Betreuung im Laufe der Arbeit bedanken.
Durch den Besuch der von Herrn Grüne angebotenen Vorlesungen aus dem Bereich der dynamischen Optimierung und der mathematischen Kontrolltheorie im Hauptstudium wurde mein Interesse geweckt. In Form eines Seminarvortrags zur stochastischen, dynamischen Optimierung konnte ich mich erstmals in das Thema einarbeiten. Herr Grüne stand im Rahmen meiner Diplomarbeit jederzeit für Nachfragen und inhaltliche Diskussionen zur Verfügung und hatte stets ein offenes Ohr, falls Probleme auftraten.
Außerdem möchte ich mich bei all denen bedanken, die mich während der Entstehungszeit begleitet und unterstützt haben. Insbesondere danke ich meiner Familie und Freundin, die mir während meiner Studienzeit zur Seite standen.

Inhaltsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Hauptproblem in der Analyse zahlreicher ökonomischer Modelle ist die genaue Bestimmung einer unbekannten Funktion, die jedoch für die Aussagekraft des Modells von Bedeutung ist.
In makroökonomischen Wachstumsmodellen geht es beispielsweise um die Frage nach der Wohlfahrtsfunktion der Volkswirtschaft oder der optimalen Konsumfunktion der Haushalte. Diese Abbilder des komplexen Wirtschaftskreislaufs versuchen die Auswirkungen von verschiedensten, steuerbaren Faktoren auf gesamtwirtschaftliche Größen wie die des Bruttoinlandsprodukts widerzuspiegeln.
Sie lassen sich als zeitdiskretes, stochastisches, dynamisches Kontrollsystem formulieren und sind somit in eine ausgefeilte, mathematische Theorie eingebettet. Geschlossene, globale Lösungen für die gesuchten Funktionen des Systems sind analytisch oft schwer bzw. gar nicht zu berechnen. Daher verwendet man zur Annäherung der Lösungen sogenannte Approximationstechniken.
Nach [15] ist der gebräuchlichste Lösungsansatz zur Ermittlung der Funktion die Methode der Linearisierung um einen Gleichgewichtspunkt, den sogenannten „steady state“. Die Umgebung um dieses Equilibrium lässt sich als Störung interpretieren. Die Vorgehensweise wurde daher unter dem Namen Perturbationsmethode bekannt. Dank der Approximationen kann man analysieren, wie sich das System nahe des Gleichgewichts entwickelt und wie es auf äußere Einwirkungen reagiert. Betrachtete Einflussfaktoren auf die Volkswirtschaft können Naturkatastrophen, Rohstoffpreisveränderungen oder insbesondere der technologische Fortschritt sein. An diesem Punkt kommt zwangsläufig die Stochastik ins Spiel. Die Modellierung derartiger externer Inputparameter durch stochastische Prozesse führt das deterministische System in ein stochastisches über. Daher ist es naheliegend, dass eine Vielzahl von Wachstumsmodellen der Klasse der „Stochastic Growth Models“ angehören.
Die Approximationsgüte der Perturbationsmethoden richtet sich nach der gewählten Ordnung in der Taylorentwicklung. Lange Zeit beschränkte man sich auf lineare Approximationen mittels Taylorpolynomen vom Grad eins.
In der vorliegenden Arbeit werden zur Bestimmung der Approximationen, ebenso wie bei [19], höhere Ableitungen verwendet, welche für eine größere Genauigkeit der Lösung sorgen. Hierbei findet das von Schmitt-Grohé und Uribe entwickelte und in [24] geschilderte Verfahren der Perturbation Anwendung.

Da man an den optimalen Wertefunktionen der Modelle interessiert ist, werden alternativ moderne Algorithmen der dynamischen Programmierung, die auf [14] zurückgehen, zur numerischen Berechnung genutzt. Zusätzlich erfolgt ein Vergleich des Lösungsansatzes der dynamischen Programmierung mit dem Perturbationsverfahren. Bei der Berechnung der optimalen Wertefunktionen mittels Perturbationsmethoden wird zudem eine als „direkt“ zu bezeichnende Methode angewandt, die eine Approximation der Wertefunktion in Form einer quadratischen Taylorreihe liefert. Beide Lösungstechniken werden im Rahmen dieser Arbeit nach schrittweiser Erläuterung der theoretischen Grundlagen an zwei unterschiedlichen volkswirtschaftlichen Beispielen, die der Klasse neoklassischer Wachstumsmodelle zuzuordnen sind, getestet.

Die Arbeit ist folgendermaßen gegliedert:
Im zweiten Kapitel werden die makroökonomischen Grundlagen für die später als Anwendungsbeispiel dienenden Modelle gelegt. Es werden im ersten Abschnitt fundamentale Begriffe aus der Wachstumstheorie geklärt. Der zweite Abschnitt führt in das Konzept der neoklassischen Wachstumstheorie ein und erläutert ein repräsentatives Grundmodell. Dieses wurde vom britischen Wissenschaftler Frank Ramsey, der sowohl in der Mathematik als auch in den Wirtschaftswissenschaften zukunftsweisende Arbeiten verfasste, entwickelt.

Im dritten Kapitel werden die theoretischen Grundlagen für die zur Anwendung kommenden Verfahren gelegt. Zunächst erfolgt eine Einführung in die wesentlichen Begriffe der Wahrscheinlichkeitstheorie und in die stochastischen Prozesse. Damit lassen sich am Ende des ersten Abschnitts ein zeitdiskretes, stochastisches Kontrollsystem und ein optimales Kontrollproblem auf unendlichem Zeithorizont formulieren. Der zweite Abschnitt beschäftigt sich mit dem Lösungsansatz der dynamischen Programmierung. Ziel ist es, die notwendigen Definitionen für stochastische Kontrollsysteme und das Bellman’sche Optimalitätsprinzip darzulegen, um letztendlich über die Hamilton-Jacobi-Bellman Gleichung optimale Kontrollprobleme lösen zu können. Ein großer Teil widmet sich der Idee der adaptiven Gitter, durch deren Einsatz im Rahmen einer Werteiteration die optimale Wertefunktion des betrachteten dynamischen Optimierungsproblems numerisch effizient berechnet werden kann. Im dritten Abschnitt wird ein weiterer Lösungsansatz verfolgt und näher geschildert: Über das Konstrukt der Hamilton Funktion werden die Optimalitätsbedingungen des sogenannten stochastischen Maximumprinzips hergeleitet, mit Hilfe derer sich u.a. das ökonomische Gleichgewicht berechnen lässt.

In Kapitel vier wird das im Vordergrund stehende Lösungsverfahren der Perturbationsmethoden untersucht. Ausgehend von den über die Hamilton–Funktion aufgestellten Gleichgewichtsbedingungen wird das Perturbationsmodell nach Schmitt–Grohe und Uribé mit seinen Annahmen und Lösungswegen vorgestellt. Auf die Berechnung der Lösungsfunktionen des Modells mittels Taylor-Approximationen wird im zweiten Abschnitt eingegangen. Dabei wird beschrieben, wie die nötigen Koeffizienten der Dynamik, der Steuerung und auf zwei verschiedene Arten diejenigen der

optimalen Wertefunktion, an der wir im Besonderen interessiert sind, ermittelt werden. Abschließend folgt ein Thementeil über die Perturbationsroutinen, deren Implementierung mit der mathematischen Software MATLAB und MAPLE erläutert wird.

Kapitel fünf befasst sich mit Implementierungsfragen im Zusammenhang mit der dynamischen Programmierung. Zur erfolgreichen Umsetzung des Algorithmus zur Berechnung der optimalen Wertefunktion müssen einige Problemstellungen wie die Erzeugung und Handhabung normalverteilter Zufallszahlen oder die numerische Integration für die Berechnung des Erwartungswerts gelöst werden. Dies erfolgt in den thematisch gegliederten Abschnitten. So wird im dritten Abschnitt die Gitterstruktur GRIDGEN vorgestellt, die auf [12] basiert und den Einsatz von adaptiven Gittern im Rahmen der numerischen Berechnung ermöglicht.

Die Anwendungsbeispiele zum Testen der beiden Lösungsverfahren, der Perturbationsmethoden und des Algorithmus der dynamischen Programmierung, werden in Kapitel sechs vorgestellt. Zwei stochastische Wachstumsmodelle, die beide von mehreren Parametern wie beispielsweise der Abschreibungsrate abhängen und sich in der Gewichtung technologischer Schocks unterscheiden, werden definiert und eingangs nach ökonomischen Gesichtspunkten interpretiert. Im zweiten Abschnitt werden die numerischen Ergebnisse der Perturbationsmethoden, speziell die Anwendung der direkten und indirekten Approximation der optimalen Wertefunktion unter Zuhilfenahme der Perturbationsroutinen aufgezeigt. Im Abschnitt drei erfolgt eine Auswertung und Analyse der beiden Modelle hinsichtlich der benutzten Lösungstechniken. Hierbei werden die numerischen Fehler der jeweiligen Approximation verglichen. Im letzten Teilabschnitt wird der Aussagekraft beider Beispiele mithilfe numerischer Simulationen, die sich auf den im vorhergehenden Kapitel vorgestellten Zufallszahlengenerator von Marsaglia stützen, Nachdruck verliehen.

Im letzten Kapitel werden die Resultate nochmals aufgegriffen und die beiden angewandten Methoden der dynamischen Programmierung und der Perturbation vergleichend gegenübergestellt. Ergänzend erfolgt ein kurzer Ausblick auf alternative Lösungsansätze und deren Gemeinsamkeiten zu den im Rahmen der Arbeit behandelten Konzepte.

Formeln, auf die im weiteren Verlauf der Arbeit Bezug genommen werden, erhalten eine Formelnummer. Diese Formelnummer wird kapitelweise hochgezählt. Literaturverweise erscheinen in eckigen Klammern und mit der jeweiligen Nummer der Literaturquelle aus dem Anhang. Die Abkürzungen beschränken sich auf allgemein gültige Abkürzungen, wie z.B., d.h. oder bzw.
Aus diesem Grund wurde ein Abkürzungsverzeichnis als unnötig erachtet. Ebenso wurde auf ein Abbildungsverzeichnis verzichtet.

2 Makroökonomische Wachstumstheorie

Das folgende Kapitel ist als Exkurs in die Wirtschaftswissenschaften zu verstehen und gibt eine Einführung in die makroökonomische Wachstumstheorie. Dabei orientiert es sich an [17] und [4]. Die grundlegende Methodik dieser Teildisziplin der Wirtschaftswissenschaften besteht in der modelltheoretischen Analyse. Allgemeine Grundstrukturen ökonomischer Sachverhalte werden am besten durch eine abstrahierende Generalisierung aufgedeckt. Allein durch das Sammeln und die Beschreibung von empirischen Tatbeständen lassen sich Wirkungszusammenhänge auf gesamtwirtschaftlicher Ebene nicht erklären. Daher denken besonders Ökonomen in Modellen, da diese die Realität auf die wichtigsten, relevanten Strukturen reduzieren.
Die Wachstumsmodelle, welche in dieser Arbeit als Anwendungsobjekt für die in der Einführung erwähnten mathematischen Verfahren dienen, sind mit vielen Annahmen versehen. Sie zeichnen sich durch einen hohen Abstraktionsgrad aus. Ihr Erklärungswert liegt darin, die Ursachen von Wirtschaftswachstum zu erfassen.
Dem Ziel eines stetigen und angemessenen Wirtschaftswachstums hat sich die Bundesregierung Deutschlands im Rahmen ihrer Wirtschaftspolitik verpflichtet. Bereits im Jahre 1967 hat die Regierung dies im sogenannten, damals verabschiedeten Stabilitätsgesetz „StWG“[1] festgehalten. Anhand dieser tiefen Verflechtung der Wachstumstheorie im politischen Alltag ist andeutungsweise erkennbar, wie bedeutsam eine gute Modellbildung und eine darauf beruhende Modellanalyse ist.
Während der Hauptteil dieser Arbeit ausführlich auf die mit modernen mathematischen Verfahren bewerkstelligte Analyse der Wachstumsmodelle abzielt, werden nun zunächst grundlegende ökonomische Begriffe definiert.
Sie spielen bei der Auseinandersetzung mit makroökonomischen Fragestellungen eine wesentliche Rolle und sind für das Verständnis der Modelle unerlässlich.

[1] Kurzbezeichnung für das „Gesetz zur Förderung der Stabilität und des Wachstums der Wirtschaft“ (StWG)

2.1 Definition grundlegender Begriffe

Dieser Abschnitt führt in die Grundlagen der Wachstumstheorie ein. Als Literatur für diesen Abschnitt wurde [4] sowie [17] verwendet.

Definition 2.1 (**Ökonomische Agenten, Wirtschaftssubjekte**). *Als ökonomische Agenten bezeichnet man die Handlungs- bzw. Entscheidungsträger in Wachstumsmodellen . Dies sind nachfolgend zwei Arten von Wirtschaftsubjekten: Unternehmen und Haushalte. Jedem der beiden Sektoren werden ökonomische Grundaktivitäten zugeordnet.*

- *Die **Unternehmen** produzieren Güter und Dienstleistungen, fragen Arbeitskräfte nach und investieren.*
- *Die **Haushalte** konsumieren die erzeugten Güter, bieten ihre Arbeitskraft an und bilden mit ihrem Einkommen Ersparnisse.*

In Modellen werden die Wirtschaftssubjekte durch einen repräsentativen Agenten nachgebildet. Auf die Aufführung des Sektors Staat und des Auslands, die zusammen einer offenen Volkswirtschaft angehören, wird in dieser Arbeit verzichtet.

Definition 2.2 (**Steady State**). *Unter einem volkswirtschaftlichen Gleichgewicht oder „steady state“ versteht man eine Situation, in der kein Wirtschaftssubjekt Veranlassung hat, sein Verhalten zu ändern. Es wird oft als gesellschaftlich optimaler Zustand oder ausgewogener Wachstumspfad bezeichnet.*

Im Kontext der Wachstumsmodelle bedeutet dies, dass sich die verwendeten Schlüsselvariablen nicht mehr ändern.

Die nachfolgend aufgeführten Begriffe bilden die zentralen Größen der ökonomischen Modelle und werden formal durch Variablen dargestellt.

Definition 2.3 (**Kapitalstock**). *Der aggregierte Kapitalstock enthält das gesamte Sachvermögen einer Ökonomie, welches für produktive Zwecke genutzt wird. Er beinhaltet u.a. den Wert sämtlicher Maschinen und Bürogebäude und stellt den wesentlichen Input für die Produktion dar (siehe 2.5). Er kann bei der Betrachtung eines längeren Zeitraums über die Perioden hinweg abgeschrieben werden.*

Definition 2.4 (**Konsum**). *Unter Konsum versteht man den Kauf von Gütern und Dienstleistungen durch Konsumenten (Haushalte). Die Höhe des Konsums bestimmt den unmittelbaren Nutzen der Haushalte (siehe Definition 2.7). Konsumverzicht führt dagegen zu einem Zuwachs des Kapitalstocks (Kapitalakkumulation).*

Wesentliche ökonomische Funktionen

Ausgangspunkt jeder Wachstumstheorie ist nach [4] die aggregierte Produktionsfunktion. Sie spezifiziert die Beziehung zwischen der gesamten Produktion und den dabei verwendeten Inputs. Die neoklassische Theorie unterscheidet dabei die drei Produktionsfaktoren Arbeit, Boden und Kapital, wobei letzterer für uns der relevanteste ist.
Dies führt auf die folgende formale Definition.

Definition 2.5 (**Die aggregierte Produktionsfunktion**).
Die Produktionsfunktion $P(K, L)$ bildet die Inputfaktoren Kapital K und Arbeit L mengenmäßig auf den Output der Ökonomie, die Güterproduktion Y, ab. Es gilt:

$$Y = P(K, L) \tag{2.1}$$

Obige Funktion quantifiziert demnach bei gegebener Menge an Kapital und Arbeit(skräften) den Output. Die erzeugte Menge an Gütern und Dienstleistungen ergibt nach entsprechender Berechnung, auf die im Rahmen der Arbeit nicht eingegangen wird, das Bruttoinlandsprodukt[2] - die wohl bekannteste volkswirtschaftliche Größe.
Bisherige Annahmen alleine reichen jedoch nicht zur modellhaften Beschreibung des Produktionsprozesses aus. Es existiert ein weiterer, nicht unerheblicher Faktor, der Einfluss auf den volkswirtschaftlichen Output hat. Er wird als technischer Fortschritt bezeichnet. Beispielsweise kann eine Volkswirtschaft mit fortgeschrittener Technologie mit der gleichen Menge an Arbeit und Kapital wesentlich mehr produzieren als eine Ökonomie mit primitiver Technologie.
Im Zusammenhang mit den Modellen stößt man dabei auf den Begriff der technologischen Schocks.

Definition 2.6 (**Technologische Schocks**). *Unter technologischen Schocks versteht man Ereignisse in einem Makromodell, die die Produktionsfunktion ändern.*
Sie gehen als zusätzliche Variable T in die Produktionsfunktion $P(K, L, T)$ ein.
Durch technischen Fortschritt kann entweder eine gleiche Produktionsmenge (Output) mit einem geringeren Einsatz an Arbeit oder Kapital erstellt werden oder eine höhere Menge mit dem gleichen Einsatz an Inputs.

Erscheinungsformen technologischer Schocks sind beispielsweise die Erfindung des PCs sowie der Mikro-Chips in der Industrie und die damit verbundene Steigerung der Arbeitsproduktivität. Daraus leitet sich die Bedeutung dieses Faktors auf den wirtschaftlichen Output Y ab.

Nun führen wir den Begriff der Nutzenfunktion ein. Sie ist für den ökonomischen Agenten Haushalt von großer Bedeutung.

[2] Das BIP erfasst die gesamte Wertschöpfung aller Waren und Dienstleistungen für den Endverbrauch, die in einer Periode in einer Volkswirtschaft hergestellt wurden. (aus [4])

Definition 2.7 (**Nutzenfunktion, Grenznutzen**). *Die Nutzenfunktion N bildet die Konsummenge C auf den Nutzen ab, der sich für den handelnden Agenten aus der Entscheidung zu konsumieren ergibt.*
Sie besitzt die folgenden Eigenschaften:

$$N(C) \geq 0 \tag{2.2}$$
$$N'(C) > 0 \tag{2.3}$$
$$N''(C) < 0 \tag{2.4}$$

Unter dem Grenznutzen versteht man die erste Ableitung $N' = \frac{\partial N}{\partial C}$ der Nutzenfunktion nach der Konsummenge.

Umgangssprachlich beantwortet der Grenznutzen die Frage, wieviel zusätzlichen Nutzen eine weitere Einheit eines Gutes stiften würde. In der Nutzentheorie wird unterstellt, dass der Grenznutzen eines Gutes mit steigendem Konsum des Gutes abnimmt[3]. Die Nutzenfunktion wird konkav gewählt und erfüllt somit Eigenschaft 2.4.
Abbildung 2.1 zeigt den typischen Verlauf einer Nutzenfunktion.

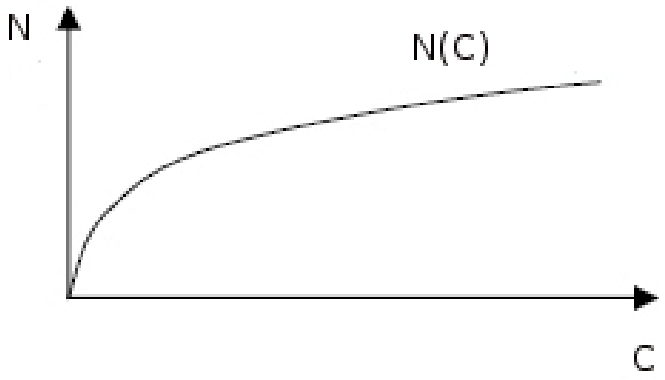

Abbildung 2.1: Typische Nutzenfunktion

2.2 Das neoklassische Wachstumsmodell nach Ramsey

Der zweite Abschnitt stellt nun konkret das neoklassische Wachstumsmodell nach Ramsey[4] vor, das als Referenzmodell dient. Die Arbeit [16] ist hierbei ebenso wie [27] und [20] als Literaturquelle zu nennen.

[3] In der Nutzentheorie wird dieser Grundsatz als erstes *Gossensches Gesetz* bezeichnet

[4] Frank Plumpton Ramsey; geb. 22. Februar 1903 in Cambridge, † 19. Januar 1930. Britischer Mathematiker & Logiker, zugleich Entwickler ökonomischer Wachstumsmodelle. Gilt als Urheber der mathematischen Analyse in der Wachstumstheorie.

2.2.1 Variablen des Modells

Die in Abschnitt (2.1) definierten Größen begegnen uns im Ramsey Modell. Da es sich um Stromgrößen handelt, d.h. um Größen, die pro Zeiteinheit definiert sind, erhalten die Variablen notationell einen Zeitindex t.

Variablen:

Y_t : Volkswirtschaftlicher Output zum Zeitpunkt t

K_t : Kapitalstock zum Zeitpunkt t

C_t : Konsumausgaben zum Zeitpunkt t

I_t : Investitionen zum Zeitpunkt t

S_t : Ersparnisse zum Zeitpunkt t

Modellparameter:

ρ : Diskontierungsfaktor

δ : Abschreibungsfaktor für Kapitalbestand

2.2.2 Das Ramsey Referenzmodell

Die Wachstumstheorie basiert nach den Worten von [16] auf der Pionierarbeit von [21], in der Ramsey sein neoklassisches Wachstumsmodell aufgestellt hat. Es handelt sich um ein Modell mit endogener Sparneigung, d.h. der Agent Haushalt bestimmt zu allen Zeitpunkten selbst über sein Spar- bzw. Konsumverhalten.
Als neoklassisch bezeichnet man „eng gefasst“ den Ansatz, dass die Sparentscheidung aufgrund der intertemporalen Maximierung des Konsumnutzens bestimmt wird. Das Mitte der 1950er Jahre weitaus bekannter gewordene und in mehreren Arbeiten von Makroökonomen erwähnte Modell von Robert Solow[5] ist ebenfalls ein neoklassisches, unterstellt jedoch im Unterschied eine exogene Sparquote. Wir wollen nun aber das Ramsey Modell mit den eben erwähnten Variablen und Parametern formulieren.

[5] Solows Artikel „A contribution to the theory of enonomic growth“ erschien 1956. Er erhielt 1987 den Nobelpreis für Wirtschaftswissenschaften.

Ramseys Modell

In mathematischer Schreibweise lautet das Modell, in welchem Ramsey (vgl. [21]) eine Antwort auf die folgende Frage

> **„How much of its income should a nation save?“**

sucht, folgendermaßen:

$$\max\ W := max \sum_{t=0}^{\infty} \left(\frac{1}{1+\rho}\right)^t \cdot N(C_t)$$

mit

$$Y_t = P(K_t) \tag{2.5}$$
$$Y_t = C_t + I_t \tag{2.6}$$
$$K_{t+1} = (1-\delta)K_t + I_t \tag{2.7}$$

Interpretation der Zielfunktion:
Es handelt sich bei obiger mathematischer Formulierung um ein Maximierungsproblem. Ramsey ging der Frage nach, wie der Agent Haushalt seinen Nutzen über mehrere Perioden anhand seines Konsum-/Sparverhaltens maximieren könne. Daher bezeichnet man W als Wohlfahrtsfunktion, N ist eine intertemporale Nutzenfunktion.
Es gilt der Grundsatz, dass der Nutzen eines Gutes in der Zukunft geringer zu bewerten ist als in der Gegenwart. Daher wird der zukünftige Nutzenzugang diskontiert [6], was durch den Vorfaktor $\left(\frac{1}{1+\rho}\right)^t$ im Zeitpunkt t erreicht wird.

Erläuterung der Modellgleichungen:

↪ Gleichung 2.5: **Beschreibung des Produktionsprozesses**
Zu jedem Zeitpunkt wird die produzierte Menge durch die Produktionsfunktion P bestimmt.

↪ Gleichung 2.6: **Verwendungsgleichung**
Der volkswirtschaftliche Output dient entweder dem Konsum oder wird investiert.
Dabei gilt in diesem Modell, wie auch in anderen geschlossenen Volkswirtschaften, die Gleichheit von Investition und Ersparnis:

$$\boxed{I_t = S_t \quad \forall t}$$

[6] Diese Annahme wurde erst später ins Modell mitaufgenommen. Ramsey lehnte diesen Ansatz ursprünglich ab.

↪ Gleichung 2.7: **Kapitalakkumulation**
Sie drückt aus, dass die zeitliche Veränderung des Kapitalstocks auf den getätigten Investitionen abzüglich der Abschreibungen beruht. Dies wird durch Umstellen der Gleichung verdeutlicht:

$$\Delta K := K_{t+1} - K_t = I_t - \delta \cdot K_t$$

Entsprechen sich beide Größen der rechten Seite, bleibt der Kapitalbestand unverändert und die Ökonomie befindet sich im steady state.

Vervollständigung des Modells:
Um das Modell zu komplettieren, werden eine Produktionsfunktion P sowie eine Nutzenfunktion N gewählt. Als konkave Nutzenfunktion gemäß Definition 2.7 eignet sich der Logarithmus:

$$N(C_t) := \log(C_t)$$

Als Produktionsfunktion P dient im Modell die sogenannte Cobb-Douglas Produktionsfunktion[7].

Definition 2.8 (Cobb-Douglas Produktionsfunktion). *Die Cobb-Douglas Produktionsfunktion ist von der Form* $Y = P(K, L) = A \cdot K^\alpha \cdot L^{1-\alpha}$.
Dabei gilt für die beiden Konstanten: $0 < \alpha < 1,\ A > 0$

Die Produktionsfunktion gehört nach [4] heutzutage zum Standardwerkzeug jedes Ökonomen und beschreibt sehr gut die Beziehung zwischen Produktion und den Inputfaktoren Kapital und Arbeit. Besonders im neoklassischen Konzept steht sie im Fokus des Modells.

Definition 2.9 (Konstante Skalenerträge).

$$P(\lambda K, \lambda L) = \lambda P(K, L), \quad \lambda > 0, \lambda \in \mathbb{R}. \tag{2.8}$$

Diese Eigenschaft bedeutet, dass eine Vervielfachung der beiden Inputs Kapital und Arbeit zu einer entsprechenden Änderung der Outputmenge führt.
Für Cobb-Douglas Produktionsfunktionen trifft dies zu, wie leicht nachzuprüfen ist:

$$P(\lambda K, \lambda L) = A \cdot (\lambda K)^\alpha \cdot (\lambda L)^{1-\alpha} = A \cdot \lambda^{\alpha+1-\alpha} \cdot K^\alpha \cdot L^{1-\alpha} = \lambda \cdot A \cdot K^\alpha \cdot L^{1-\alpha}$$

Gleichbedeutend in mathematischer Sprache ist damit, dass die Funktion **linear homogen** ist.

Zum Abschluss dieses Kapitels wollen wir den dynamischen, volkswirtschaftlichen Prozess im Ramsey Modell mit nachfolgender Abbildung zusammenfassen.

[7] 1928 von den Amerikanern Charles Cobb (Mathematiker) und Paul Douglas (Ökonom) erfunden.

Zusammenfassung:

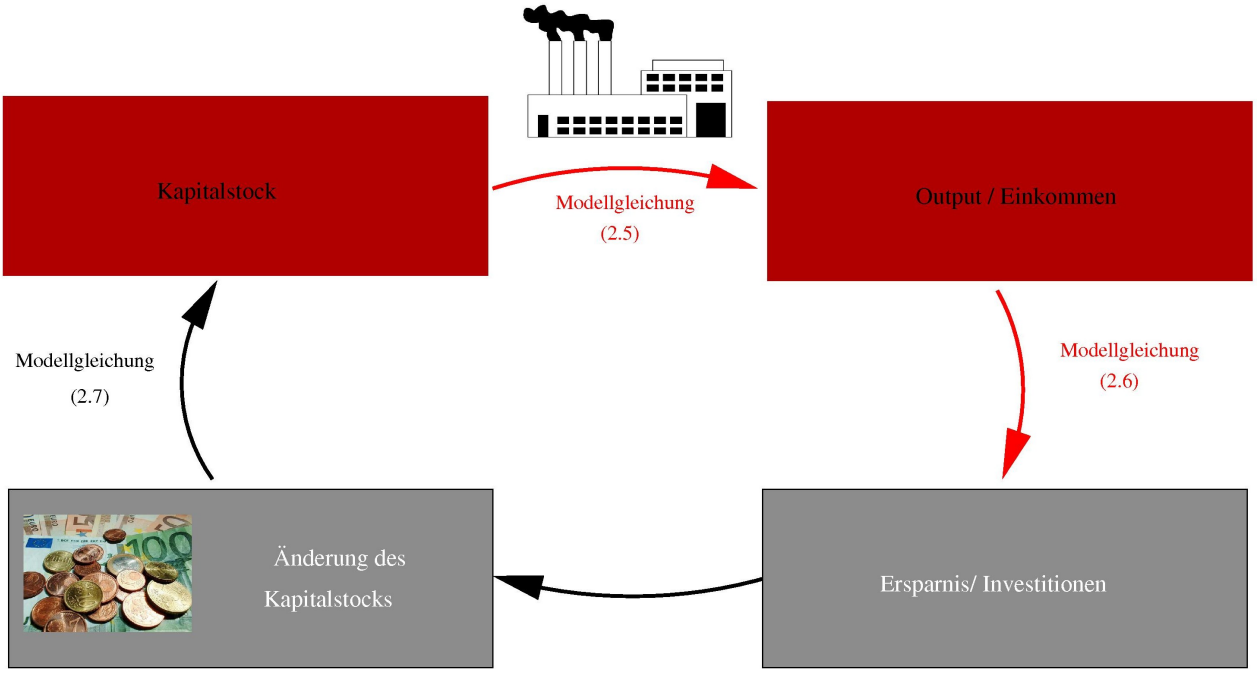

Abbildung 2.2: Zusammenspiel der Modellvariablen

In der Volkswirtschaft nach Ramsey stellt sich das Problem, das in einer Periode erwirtschaftete Einkommen (Output gemäß Modellgleichung (2.5)) entweder für den Konsum zu nutzen oder zu investieren (Verwendungsgleichung (2.6)). Während die getroffene Entscheidung zu konsumieren einen unmittelbaren Nutzen stiftet (Zielfunktion), führt die Investition andererseits zu einer Erhöhung des Kapitalstocks (Kapitalakkumulationsgleichung (2.7)). Dies bewirkt gleichzeitig einen höheren zukünftigen Output und damit einen höheren möglichen Konsum. Letztendlich ist der ökonomische Agent vor das Problem einer optimalen Ressourcenallokation gestellt.

3 Mathematische Theorie der Verfahren

In diesem Kapitel wird die mathematische Theorie der später angewandten Verfahren erläutert. Wie einleitend erwähnt, bedient man sich in der Analyse ökonomischer Probleme mathematischer Methoden, um durch Bestimmung der unbekannten Funktionen die Aussagekraft des Modells für die Realität herzuleiten.
Bei der Wahl des Lösungsansatzes der dynamischen Programmierung spielt die Hamilton-Jacobi-Bellman Gleichung und das Bellman'sche Optimalitätsprinzip die zentrale Rolle. Mit Einsatz von Gittern lässt sich ein numerischer Algorithmus zur Bestimmung der optimalen Wertefunktion entwickeln, der im zweiten Abschnitt beschrieben wird. Im Falle der Perturbationsmethoden, denen sich Schmitt-Grohe und Uribé in [24] widmen, beruht der Lösungsansatz auf zentralen Sätzen der Analysis und der Theorie der Hamilton-Funktion verknüpft mit dem stochastischen Maximumprinzip.

3.1 Stochastische Grundlagen und zeitdiskretes Kontrollsystem

Zunächst erfolgt eine kurze Darstellung zentraler stochastischer Grundlagen, welche in den daran anschließenden Ausführungen Verwendung finden. Ziel ist es, damit am Ende dieses Paragraphs ein zeitdiskretes, stochastisches Kontrollsystem zu definieren. Dieses bietet uns beispielweise die Möglichkeit, die im vorherigen Abschnitt erläuterten Gleichungen (2.5) – (2.7) in ein mathematisches Rahmenwerk einzubetten. Das resultierende optimale Kontrollproblem dient uns als Motivation für die beiden Lösungsverfahren, deren theoretischer Hintergrund in den nachfolgenden Abschnitten veranschaulicht wird. Als Literaturquellen für den ersten Abschnitt wurden insbesondere [8] und [13] herangezogen. In Fragen der Maß- und Integrationstheorie wird auf [1] verwiesen.

3.1.1 Grundbegriffe der Wahrscheinlichkeitstheorie

Bis wir an den Punkt gelangen, stochastische Parameter in die Modelle einzubauen, benötigen wir vorab einige zentrale Grundbegriffe der Wahrscheinlichkeitstheorie.

Definition 3.1 (**Wahrscheinlichkeitsraum, σ–Algebra**). *Ein Wahrscheinlichkeitsraum besteht aus einem Tripel (Ω, Σ, P) mit:*

- *Ω ist die Menge der Elementarereignisse $\omega \in \Omega$*
  (die ω lassen sich im ökonomischen Hintergrund beispielsweise als technologische Schocks (siehe (2.6)) auffassen)
- *$\Sigma \subseteq P(\Omega)$ ist eine σ-Algebra auf Ω, d.h. eine Teilmenge der Potenzmenge von Ω mit den folgenden Eigenschaften:*
  a) $\emptyset, \Omega \in \Sigma$
  b) $A \in \Sigma \Rightarrow A^C := \Omega \setminus A \in \Sigma$ *und*
  c) Seien $A_i \in \Sigma \; \forall i \Rightarrow \bigcup_{i=1}^{\infty} A_i \in \Sigma$ *und* $\bigcap_{i=1}^{\infty} A_i \in \Sigma$
- *P ist das Wahrscheinlichkeitsmaß auf Σ, d.h. $P : \Sigma \to [0,1]$ mit den Eigenschaften:*
  a) $P(\Omega) = 1$ *(Ω ist das sichere Ereignis)*
  b) $P(A^C) = 1 - P(A) \quad \forall A \in \Sigma$
  c) $P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$ *für jede Folge paarweise disjunkter Mengen aus Σ*
  (d.h. $A_i \cap A_j = \emptyset$ für beliebige $i, j : i \neq j$)

Definition 3.2 (**Borel-σ-Algebra**). *Die Menge $\mathbb{B}$ bezeichnet die Borel'sche-σ–Algebra. Sie ist die gebräuchliche σ–Algebra für die reellen Zahlen $\mathbb{R}$. Sie enthält alle offenen Intervalle $]a, b[\ ,\ a, b \in \mathbb{R}$.*

Um zufällige Ereignisse $\omega \in \Omega$ im Rahmen einer Analyse zahlenmäßig erfassen zu können, eignen sich als spezielle Abbildungen Zufallsvariablen.

Definition 3.3 (**Zufallsvariable, Verteilungsfunktion**). *Gegeben sei ein Wahrscheinlichkeitsraum (Ω, Σ, P). Eine auf Ω definierte Funktion $X : \Omega \to \mathbb{R}$ heißt eine reelle Zufallsvariable, wenn für $x \in \mathbb{R}$ gilt:*
$X^{-1}((-\infty, x]) = \{\omega \in \Omega : \ X(\omega) \leq x)\} \in \Sigma$, *wobei X^{-1} die inverse Funktion bezeichnet.*

Mit anderen Worten: Für jede reelle Zahl x ist eine Funktion $F_X : \mathbb{B} \to [0,1]$ erklärt.

$$F_X(x) = P(\{\omega : \ X(\omega) \leq x\}) = P(X \leq x) \tag{3.1}$$

Man nennt F_X die zu X gehörige Verteilungsfunktion bzw. Wahrscheinlichkeitsverteilung.

Werden mehrere Zufallsvariablen betrachtet, so ist der Begriff der Unabhängigkeit und der identischen Verteilung von Bedeutung.

Definition 3.4 (**Unabhängige, identisch verteilte Zufallsvariablen**). *Eine beliebige Folge von Zufallsvariablen $X_1, X_2, \ldots$ auf dem gleichen Wahrscheinlichkeitsraum (Ω, Σ, P) heißt unabhängig und identisch verteilt (kurz: **i.i.d.**[1]), wenn folgende zwei Bedingungen gelten:*

a) Die Zufallsvariablen sind unabhängig, d.h. es gilt

$$P(X_1 \in B_1, \ldots, X_k \in B_k) = P(X_1 \in B_1) \cdot \ldots \cdot P(X_k \in B_k) \quad \forall B_1, \ldots, B_k \in \mathbb{B}. \tag{3.2}$$

Die gemeinsame Verteilung ist das Produkt der einzelnen Verteilungen (Produktregel).

b) Die Zufallsvariablen besitzen dieselbe Verteilungsfunktion, d. h. $F_{X_i} = F_{X_j} \quad \forall i, j \in \mathbb{N}$.

Man unterscheidet in der Stochastik zwischen diskreten und stetigen Verteilungsfunktionen. Im Hinblick auf die späteren ökonomischen Anwendungen ist es an dieser Stelle ausreichend, stetige Verteilungen zu betrachten. Dies sind Verteilungen von Zufallsvariablen, die überabzählbar viele Werte annehmen können.
Dies führt zunächst auf den Begriff der Verteilungsdichtefunktion.

Definition 3.5 (**Dichtefunktion**). *Sei X eine Zufallsvariable auf dem Wahrscheinlichkeitsraum (Ω, Σ, P). Ihre Verteilung F_X besitzt genau dann eine Dichtefunktion p_X, wenn gilt:*

$$F_x(c) = \int_{-\infty}^{c} p_X(x)dx \quad \forall c \in \mathbb{R} \tag{3.3}$$

Die Dichtefunktion p_X als Ableitung der Verteilungsfunktion F_X besitzt die folgenden zwei Eigenschaften:

a) $p_X(x) \geq 0$

b) $\int_{-\infty}^{\infty} p_X(x)dx = 1$

Damit sind wir in der Lage zwei bedeutende Kenngrößen für Verteilungen zu definieren:
Erwartungswert und Varianz.

Definition 3.6 (**Erwartungswert**). *X sei eine Zufallsvariable auf dem Wahrscheinlichkeitsraum (Ω, Σ, P). Der Erwartungswert einer Zufallsvariablen X wird definiert als*

$$\mathbb{E}(X) := \int_{-\infty}^{\infty} x \cdot p_X(x)dx \tag{3.4}$$

[1] Abk. für independent and identically distributed

Definition 3.7 (**Varianz, Streuung**). *Als Varianz einer Zufallsvariablen X auf dem Wahrscheinlichkeitsraum (Ω, Σ, P) bezeichnet man*

$$Var(X) := \mathbb{E}\big([X - \mathbb{E}(X)]^2\big) = \int_{-\infty}^{\infty} (x - \mathbb{E}(X))^2 \cdot p_X(x) dx \tag{3.5}$$

Als weiterer Verteilungsparameter ist die Standardabweichung σ gegeben durch

$$\sigma(X) := \sqrt{Var(X)}. \tag{3.6}$$

Bemerkung 3.8. *Wir setzen in den obigen Definitionen jeweils voraus, dass die entsprechenden Integrale existieren. Nähere Einzelheiten zur Integrationstheorie, etwa zur Existenz der entsprechenden Momente einer Zufallsvariablen, finden sich in [1].*

Mit dem Erwartungswert charakterisieren wir den „typischen“ oder „mittleren“ Wert der Zufallsvariablen, die Varianz dient als Maß dafür, wie stark die Werte der Zufallsvariablen von $\mathbb{E}(X)$ abweichen. Als in praktischen Anwendungen am häufigsten auftretende stetige Verteilung gilt die Normalverteilung, die auch für unsere Zwecke sehr relevant ist.

Definition 3.9 (**Normalverteilung**). *Eine stetige Zufallsvariable X mit Erwartungswert μ und Varianz σ^2 heißt normalverteilt, wenn die Verteilungsdichtefunktion gegeben ist als*

$$p_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}. \tag{3.7}$$

Man schreibt für normalverteilte Zufallsvariablen $X \sim N(\mu, \sigma^2)$.
Eine Zufallsvariable $Y \sim N(0, 1)$ heißt standardnormalverteilt, ihre Dichtefunktion p_Y wird als „Gauß'sche Glockenkurve“ bezeichnet.

Nachfolgende Abbildung (3.1) zeigt einige Dichtefunktionen von Normalverteilungen für verschiedenen Varianzen, die blaue Kurve den Graphen der Gauß'schen Glockenkurve.

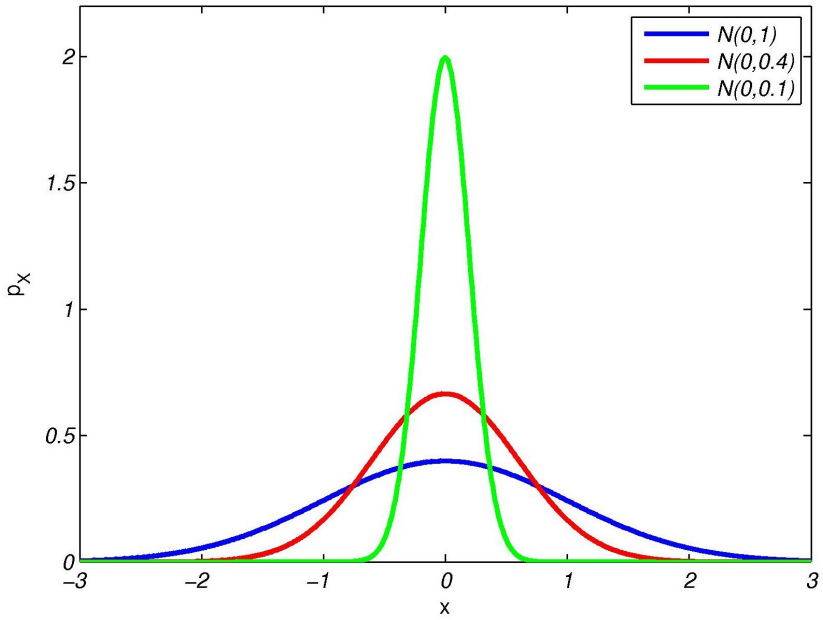

Abbildung 3.1: Dichtefunktion von Normalverteilungen

3.1.2 Stochastische Prozesse

Zur Erfassung möglicher Abhängigkeiten zufälliger Größen von deterministischen Parametern geht man vom Begriff der Zufallsvariable zu dem des stochastischen Prozesses über. Mit solchen „Zufallsprozessen“ lassen sich z.B. die Bewegung eines Moleküls auf der Wasseroberfläche in der Physik (Brownsche Bewegung) oder, um im ökonomischen Kontext zu bleiben, das Ausmaß technologischer Schocks in Abhängigkeit von der Zeit ausdrücken.

Definition 3.10 (Stochastischer Prozess). *Eine Familie von Zufallsvariablen über einem Wahrscheinlichkeitsraum (Ω, Σ, P), abhängig von einem Parameter $t \in T$, heißt stochastischer Prozess. Ein reellwertiger stochastischer Prozeß X ist definiert als Funktion*

$$X : \Omega \times T \to \mathbb{R}, \quad (\omega, t) \to X(\omega, t). \tag{3.8}$$

Im folgenden ist t immer der Zeitparameter. Wird ein Elementarereignis ω^* festgehalten, so nennt man $X(\omega^*, \cdot)$ eine Realisierung (oder Pfad) des stochastischen Prozesses. Betrachtet man einen festen Zeitpunkt $\hat{t} \in T$, so ist $X(\cdot, \hat{t})$ eine gewöhnliche Zufallsvariable.

Bemerkung 3.11. *Für $T = [a, b]$ mit $a, b \in \mathbb{R}, -\infty < a < b < \infty$ spricht man von einem zeitstetigen stochastischen Prozess. Wir betrachten in dieser Arbeit aber diskrete Prozesse, also $T = \mathbb{N}_0$. Die in Abschnitt (3.1.1) eingeführten Begriffe und zentralen Parameter sind*

auf stochastische Prozesse übertragbar.
So ist z.B. durch $F_X(x,t) = P(X(t) \leq x)$ die Verteilungsfunktion eines stochastischen Prozesses gegeben.
Unter anderem lassen sich stochastische Prozesse nach dem Typ der Verteilung klassifizieren. Bekannte Prozesstypen sind folgende:

- *Markov'sche Prozesse*
- *Wiener'sche Prozesse*
- *Gauß'sche Prozesse.*

Bei letztgenannter Klasse genügen die Zufallsgrößen $X(\cdot, t)$ einer Normalverteilung.

3.1.3 Das zeitdiskrete, stochastische Kontrollsystem

Nachdem die relevanten Grundbegriffe der Wahrscheinlichkeitstheorie geklärt wurden, wird in diesem Abschnitt ein zeitdiskretes, stochastisches Kontrollsystem definiert, auf dem die weiteren Ausführungen basieren. Dabei werden auch die Begriffe Kontrolle bzw. Steuerung näher erklärt.

Definition 3.12 (**Zeitdiskretes, stochastisches Kontrollsystem**). *Ein zeitdiskretes, stochastisches Kontrollsystem ist gegeben durch eine Abbildung*

$$\varphi : \mathbb{R}^n \times U \times \mathbb{R}^m \to \mathbb{R}^n, \qquad (x, u, z) \mapsto \varphi(x, u, z).$$

Hierbei bezeichnet $x \in \mathbb{R}^n$ den Zustand, $u \in U \subseteq \mathbb{R}^d$ die Kontrolle (Synonym: Steuerung) und $z \in \mathbb{R}^m$ die stochastische Störung. φ bezeichne die Dynamik des Systems.

Zum Anfangswert $x_0 \in \mathbb{R}^n$, einer Kontrolle $u = (u_t, t \in \mathbb{N}_0)$ und einer Folge von Zufallsvariablen $Z_0, Z_1, \ldots$ (i.i.d.) definieren wir die Lösung $X_t = X_t(x_0, u)$ des Systems für $t \geq 0$ induktiv mittels

$$X_0 = x_0, \quad X_{t+1} = \varphi(X_t, u_t, Z_t).$$

Das definierte System kann im Zeitablauf verschiedene Zustände annehmen. Die Zustandsvariablen fassen jene Größen zusammen, welche das Verhalten des Systems beschreiben.
In jedem Zeitpunkt kann der Entscheidungsträger den Wert der Kontrollvariablen u wählen und steuert somit das System.

Bemerkung 3.13. *Der Zustandsraum wird mit $X \subset \mathbb{R}^n$ bezeichnet und als kompakt angenommen.*
Für die Lösung wird nachfolgend die kürzere Notation X_t verwendet. Nur dort, wo es notwendig ist, wird das Argument (x_0, u) aufgeführt.

Die Kontrolle u des Systems ist selbst wie Z ein stochastischer Prozess (Kontrollprozess), da u zeitabhängig ist und die stochastische Information zu jedem Zeitpunkt die Wahl von u maßgeblich beeinflusst. Denn wählt man beispielsweise den Kontrollwert u_0 fest, so soll die Möglichkeit bestehen, zu einem Zeitpunkt $t \geq 1$ auf die dann bekannte, jetzt aber noch ungewisse zukünftige Situation zu reagieren, d.h. die Kontrollfunktion hängt von der anfangs unbekannten Zukunft ab.

Die Wahl des Kontrollwerts u soll zu jedem Zeitpunkt t aber realistischerweise nur von den vergangenen und dem gegenwärtigen Zustand $X_0, X_1, \ldots X_t$ abhängen.
Dies beschreibt folgende kontrolltheoretische Definition von der Zulässigkeit eines Kontrollprozesses.

Definition 3.14 (Zulässiger Kontrollprozess). *Ein Kontrollprozess u heißt zulässig, wenn Abbildungen $h_t : (\mathbb{R}^n)^{t+1} \to U, t \in \mathbb{N}_0$ existieren, so dass*

$$u_t = h_t(X_0, X_1, \ldots, X_t) \qquad \forall t \in \mathbb{N}_0. \tag{3.9}$$

Der ökonomische Agent wählt u_t demnach in geeigneter Feedbackform, d.h. die Kontrolle ist abhängig vom gegenwärtigen und den vergangenen Zuständen sowie von der Zeit.

Bemerkung 3.15 (Standardisierung der Z_t). *Für den stochastischen Einfluss im System gilt die Prämisse, dass die Z_t normalverteilte Zufallsvariablen mit $\mathbb{E}(Z_t) = 0$ und $Var(Z_t) = \sigma^2$ sind.*
Die $Z_t \sim N(0, \sigma^2)$ lassen sich zu $\epsilon_t := \frac{Z_t}{\sigma}$ standardisieren.
Es gilt dann $\epsilon_t \sim N(0, 1)$ und $\sigma \cdot \epsilon_t \sim Z_t$.[2]

Optimales Kontrollproblem

Nachdem das diskrete, stochastische Kontrollsystem eingeführt wurde, wird nun die für die späteren Lösungsansätze relevante Klasse dynamischer Optimierungsprobleme, das sogenannte optimale Kontrollproblem betrachtet. Die Kontrollvariable u wird im Normalfall nicht willkürlich, sondern einem Optimierungsziel folgend gewählt. Dies macht folgende Definition ersichtlich.

Definition 3.16 (Optimales Kontrollproblem). *Gegeben sei ein diskretes, stochastisches Kontrollsystem gemäß (3.12).*
Man definiert eine Ertragsfunktion $l : \mathbb{R}^n \times U \to \mathbb{R}$ und eine Diskontrate $\beta \in (0, 1]$.
Unter einem optimalen Kontrollproblem auf unendlichem Zeithorizont $T = \mathbb{N}_0 = \{0, 1, 2, \ldots\}$ versteht man folgende Problemstellung:

$$\boxed{\max_{u \in U(x_0)} J_\infty(x_0, u) := \mathbb{E}\left(\sum_{t=0}^{\infty} \beta^t \cdot l(x_t, u_t)\right)}$$

$U(x_0)$ ist die zum Anfangswert x_0 zulässige, beschränkte Kontrollmenge.

[2] Die ϵ_t treten im vierten Kapitel beim Perturbationsmodell wieder auf.

Die optimale Wertefunktion

Zum optimalen Kontrollproblem führen wir die folgenden Bezeichnungen ein.

Definition 3.17 (**Optimale Wertefunktion**). *Die optimale Wertefunktion V_∞ zum Kontrollproblem (3.16) wird definiert als*

$$V_\infty(x_0) := \sup_{u \in U(x_0)} J_\infty(x_0, u). \tag{3.10}$$

Die Funktion ordnet jedem Anfangswert x_0 den maximal erzielbaren Zielfunktionswert zu.

Definition 3.18 (**Optimaler Kontrollprozess**). *Ein Kontrollprozess $\widehat{u}$ heißt optimaler Kontrollprozess für J_∞, wenn gilt:*

$$J_\infty(x_0, \widehat{u}) = V_\infty(x_0) \tag{3.11}$$

Die zugehörige Lösung $X_t(x_0, \widehat{u})$ wird folglich optimale Lösung bzw. Trajektorie genannt.

3.2 Der Ansatz der dynamischen Programmierung

Ziel dieses Abschnitts ist es, das Konzept der dynamischen Programmierung, insbesondere das Bellman'sche Optimalitätsprinzip, darzulegen. Die das Prinzip verkörpernde, berühmte Bellman–Gleichung dient der Lösung optimaler Kontrollprobleme. Die Unterabschnitte erläutern die numerische Umsetzung, von der Diskretisierung des Zustandsraums mit Gittern, über die Fehlerabschätzung bis hin zum Entwurf eines Algorithmus, der adaptive Gitter verwendet. Durch deren Einsatz kann numerisch die optimale Wertefunktion des betrachteten Optimierungsproblems auf unendlichem Zeithorizont berechnet werden.
Zunächst wird das wegweisende Optimalitätsprinzip, welches auf dem amerikanischen Mathematiker Richard Bellman[3] beruht, in Form eines Satzes bewiesen.

3.2.1 Bellman'sches Optimalitätsprinzip

Satz 3.19 (**Bellman'sches Optimalitätsprinzip**). *Die optimale Wertefunktion V_∞ erfüllt für alle Anfangswerte x_0 und alle Zeitpunkte $k \in \mathbb{N}_0$ die Gleichung*

$$V_\infty(x_0) = \sup_{u_t \in U(x_0)} \mathbb{E}\left\{ \sum_{t=0}^{k} \beta^t l(X_t, u_t) + \beta^{k+1} V_\infty(X_{k+1}) \right\}, \tag{3.12}$$

mit $X_t = X_t(x_0, u)$. Für den Spezialfall $k = 0$ ergibt sich:

$$V_\infty(x_0) = \sup_{u \in U(x_0)} \mathbb{E}\left\{ l(x_0, u) + \beta V_\infty(\varphi(x_0, u, Z_0)) \right\}. \tag{3.13}$$

[3] geb. 29. August 1920, † 19. März 1984. Seine Erfindung des Prinzips der dynamischen Programmierung in den fünfziger Jahren war ein entscheidender Durchbruch auf dem Gebiet der Entscheidungsprozesse.

Beweis:
Der Beweis wird für (3.13) ausgeführt, da (3.12) mittels Induktion über k folgt.

„$\leq$“ : Es seien $x_0 \in \mathbb{R}^n$ und $u \in U(x_0)$ beliebig. Dann gilt:

$$\begin{aligned} J_\infty(x_0,u) &= \mathbb{E}\left\{\sum_{t=0}^{\infty}\beta^t l(X_t,u_t)\right\}(Definition) \\ &= \mathbb{E}\left\{l(x_0,u_0)+\sum_{t=1}^{\infty}\beta^t l(X_t,u_t)\right\} = l(x_0,u_0)+\beta\mathbb{E}\left\{\sum_{t=0}^{\infty}\beta^t l(X_{t+1},u_{t+1})\right\} \\ &\leq \mathbb{E}\left\{l(x_0,u_0)+\beta V_\infty(X_1)\right\} \\ &= \mathbb{E}\left\{l(x_0,u_0)+\beta V_\infty(\varphi(x_0,u_0,Z_0))\right\} \\ &\leq \sup_{u\in U(x_0)} \mathbb{E}\left\{l(x_0,u_0)+\beta V_\infty(X_1)\right\} \end{aligned}$$

Diese Ungleichung gilt für alle zulässigen u, somit auch für $V_\infty(x_0) = \sup_{u\in U(x_0)} J_\infty(x_0,u)$.

„$\geq$“ : Sei $\epsilon > 0$. Zu beliebigem, aber festem $x \in \mathbb{R}^n$ wird ein Kontrollprozess $u^x \in U(x)$ gewählt, für den gilt:

$$J_\infty(x,u^x) \geq V_\infty(x) - \epsilon \tag{3.14}$$

Wähle einen Kontrollwert $u^* \in U$ mit

$$\mathbb{E}\left\{l(x_0,u^*)+\beta V_\infty(\varphi(x_0,u^*,Z_0))\right\} \geq \sup_{u\in U}\mathbb{E}\left\{l(x_0,u)+\beta V_\infty(\varphi(x_0,u,Z_0))\right\} - \epsilon.$$

Zudem wählt man einen Kontrollprozess $u \in U(x_0)$ mit Startkontrollwert $u_0 = u^* = h_0(x_0)$. Definiere einen aus u und u^{X_1} zusammengefügten Kontrollprozess $\tilde{u}$ [4]:

$$\tilde{u}_t := \begin{cases} u_0, & t=0 \\ u_{t-1}^{X_1}, & t=1,2,\ldots \end{cases}$$

Dieser ist zulässig für den Anfangswert x_0 gemäß (3.9), da entsprechende Feedbackabbildungen $\tilde{h}_t$ gegeben sind durch:

$$\tilde{h}_t(x_0,\ldots,x_t) = \begin{cases} h_0(x_0), & t=0 \\ h_{t-1}^{x_1}(x_1,\ldots,x_t), & t=1,2,\ldots \end{cases}$$

Die zu $\tilde{u}$ gehörige Lösung werde mit $\tilde{X}_t = X_t(x_0,\tilde{u})$ bezeichnet.

[4] Wird als Konkatenation bezeichnet.

Dann ergibt sich

$$\begin{aligned}
V_\infty(x_0) &= \sup_{u\in U(x_0)} J_\infty(x_0,\tilde{u}) \\
&= \sup_{u\in U(x_0)} \mathbb{E}\left\{\sum_{t=0}^{\infty}\beta^t l(X_t,u_t)\right\} \\
&= \sup_{u\in U(x_0)} \mathbb{E}\left\{l(x_0,u_0)+\sum_{t=1}^{\infty}\beta^t l(X_t,u_t)\right\} \\
&\geq \mathbb{E}\left\{l(x_0,\tilde{u}_0)+\beta\sum_{t=0}^{\infty}\beta^t l(\tilde{X}_{t+1},\tilde{u}_{t+1})\right\} \\
&= l(x_0,u^*)+\beta\mathbb{E}\left\{\mathbb{E}_1\left(\sum_{t=0}^{\infty}\beta^t l(\tilde{X}_{t+1},u_t^{X_1})\right)\right\} \\
&\geq \sup_{u\in U}\mathbb{E}\left\{l(x_0,u)+\beta V_\infty(\varphi(x_0,u,Z_0))\right\}-2\epsilon
\end{aligned}$$

Im letzten Schritt wurde die Wahl von u und (3.14) ausgenutzt.
Mit $\epsilon\to 0$ folgt schließlich die Ungleichung und somit die Behauptung (3.13). □

Das Bellman'sche Optimalitätsprinzip verdeutlicht anschaulich, dass Endstücke optimaler Trajektorien selbst wieder optimale Trajektorien darstellen. Anders ausgedrückt bedeutet das: Steuert man vom Anfangszustand x_0 ausgehend bis zu einem gewissen Zeitpunkt k optimal, so ist für den erreichten Zustand x_k , den man dann als neuen Anfangswert auffasst, die restliche Kontrollwertfolge $u_k, u_{k+1}\ldots$ optimal.
Um den Satz dahingehend zu verstärken, dass V_∞ die *eindeutige* Lösung der Bellman-Gleichung (3.13) darstellt, benötigt man eine weitere Bedingung, die geeignetes Grenzverhalten der rechten Seite des Optimalitätsprinzips beschreibt.

Definition 3.20 (Transversalitätsbedingung). *Eine Funktion $W:\mathbb{R}^n\to\mathbb{R}$ erfüllt die sog. Transversalitätsbedingung, falls für $x_0\in\mathbb{R}^n$ und jede Folge von Kontrollprozessen $u^T\in U(x_0)$ mit zugehörigen Lösungen $X_t^T=X_t(x_0,u^T)$ die nachfolgenden Implikationen gelten:*

$$\limsup_{T\to\infty}\mathbb{E}\left\{\sum_{t=0}^{T-1}\beta^t l(X_t,u_t)+\beta^T W(X_T)\right\}<\infty\Rightarrow\liminf_{T\to\infty}\mathbb{E}\{\beta^t W(X_t)\}\geq 0 \qquad (3.15)$$

$$\liminf_{T\to\infty}\mathbb{E}\left\{\sum_{t=0}^{T-1}\beta^t l(X_t,u_t)+\beta^T W(X_T)\right\}>-\infty\Rightarrow\limsup_{T\to\infty}\mathbb{E}\{\beta^t W(X_t)\}\leq 0 \qquad (3.16)$$

Bemerkung 3.21. *Die Transversalitätsbedingung wird von der optimalen Wertefunktion V_∞ erfüllt, falls für die Diskontrate $\beta<1$ gilt und die Ertragsfunktion l beschränkt ist.*

Damit ist nun folgende Eindeutigkeitsaussage möglich.

Satz 3.22. *Die optimale Wertefunktion V_∞ erfülle die Transversalitätsbedingung (3.15), (3.16). Dann stimmt jede weitere Funktion $\hat{W}$, die (3.12) erfüllt, mit V_∞ überein.*

Beweis
Der Beweis ist in [13] nachzulesen. □

Damit ist die Wertefunktion V_∞ als eindeutige Lösung der Bellman-Gleichung gekennzeichnet und kann, wie in den nachfolgenden Abschnitten aufgezeigt wird, ausgehend von (3.13) berechnet werden.
Mithilfe des Optimalitätsprinzips ist es außerdem möglich, optimale Kontrollprozesse zu charakterisieren. Für diese ergibt sich, wie der folgende Satz zeigt, eine sehr schöne Darstellung in Feedback-Form, die noch prägnanter ist als diejenige von (3.9).

Satz 3.23 (Optimaler Kontrollprozess als Feedback). *Für alle $x \in \mathbb{R}^n$ sei das Supremum in (3.13) ein Maximum. Dann existiert eine Funktion $F_\infty : \mathbb{R}^n \to U$, so dass folgende Gleichung gilt:*

$$\sup_{u\in U} \mathbb{E}\left\{l(x,u) + \beta V_\infty(\varphi(x,u,Z_0))\right\} = \mathbb{E}\left\{l(x,F_\infty(x)) + \beta V_\infty(\varphi(x,F_\infty(x),Z_0))\right\} \tag{3.17}$$

Dadurch ist für jeden beliebigen Anfangswert x_0 induktiv eine optimale Lösung $X_t, t \in \mathbb{N}_0$, definiert mittels

$$X_0 = x_0,\ X_{t+1} = f(X_t, F_\infty(X_t), Z_t).$$

D.h. für den zugehörigen optimalen Kontrollprozess $u_t^\infty = F_\infty(X_t),\ \forall t \in \mathbb{N}_0$ gilt

$$J_\infty(x_0, u^\infty) = V_\infty(x_0),$$

unter der Prämisse, dass V_∞ und $W(x_0) := J_\infty(x_0, u^\infty)$ die Transversalitätsbedingung erfüllen.

Beweis:
Aus der Existenz der Maxima folgt unmittelbar die Existenz der Funktionen F_∞ . Die Tatsache, dass die angegebene Lösung durch den Kontrollprozess u_∞ erzeugt wird, ist durch (3.12) erwiesen. Der Beweis der Hauptaussage, sprich der Optimalität von X_t und u^∞, ist in [13] nachzulesen und wird an dieser Stelle nicht ausgeführt. □

Bemerkung 3.24. *Waren die zulässigen Kontrollprozesse noch in Feedback-Form (3.9) in Abhängigkeit aller vergangenen Zustände, des aktuellen Zustands und der Zeit t gegeben, so ist hingegen die optimale Kontrollstrategie wegen $u_t = F_\infty(X_t)$ nur noch an den gegenwärtigen Zustand des Systems gekoppelt (Zustandsfeedback).*

3.2.2 Iterative, numerische Bestimmung der optimalen Wertefunktion

Dieser Abschnitt erläutert die numerische Umsetzung der dynamischen Programmierung. Ziel ist es, eine Approximation $\tilde{V}_\infty$ der optimalen Wertefunktion eines zeitdiskreten, optimalen Kontrollproblems zu berechnen.
Die Berechnung der optimalen Wertefunktion, die eindeutige Lösung von (3.13) ist, erfolgt iterativ ausgehend vom eben bewiesenen Optimalitätsprinzip. Zunächst wird eine geeignete Funktionenklasse definiert.

Definition 3.25. *Mit $F(A,B)$ wird die Menge aller Funktionen von einer Menge A in eine Menge B bezeichnet. Der Projektionsoperator $\Pi : F(A,B) \to \mathcal{W}$ bildet eine beliebige Funktion W aus $F(A,B)$ auf ihre numerische Approximation $\tilde{W}$ ab, $\tilde{W} = \Pi W$.*

Für die rechte Seite des Optimalitätsprinzips in (3.13) wird der sogenannte Dynamische Programmierung-Operator $\mathcal{T} : F(\mathbb{R}^n, \mathbb{R}) \to F(\mathbb{R}^n, \mathbb{R})$ eingeführt.
Es gilt:

$$\mathcal{T}(W)(x) := \sup_{u \in U(x)} \mathbb{E}\{l(x,u) + \beta W(\varphi(x,u,z)\} \tag{3.18}$$

Das Optimalitätsprinzip lässt sich damit kurz schreiben als:

$$V_\infty(x) = \mathcal{T}(V_\infty)(x) \quad \forall\, x \in X \tag{3.19}$$

Die gesuchte optimale Wertefunktion ist demnach charakteristischer Fixpunkt des Operators $\mathcal{T}$. Mit der Iteration

$$\tilde{V}_0 \equiv 0, \quad \tilde{V}_{k+1} = \Pi\mathcal{T}(\tilde{V}_k), \quad k = 0, 1, \ldots \tag{3.20}$$

erhält man numerische Approximationen $\tilde{V}_k$ der optimalen Wertefunktion.
Es lässt sich leicht zeigen, dass $\lim_{k\to\infty} \tilde{V}_k = \tilde{V}_\infty$.

Im Rahmen einer numerischen Umsetzung iteriert man solange, bis sich die Iterierten nicht mehr wesentlich unterscheiden, also $\| \tilde{V}_{k+1} - \tilde{V}_k \|_X \leq \epsilon$ bei Vorgabe eines ϵ. Dieses Vorgehen wird als **Werteiteration** bezeichnet.
In den nachfolgenden Abschnitten wird dieser abstrakt eingeführte Ansatz konkretisiert.

3.2.3 Diskretisierung des Zustandsraums

Eine Diskretisierung des Zustandsraums X ist nötig, um eine numerische Umsetzung auf dem Rechner zu ermöglichen. Die Bellman–Gleichung in Form von (3.19) gilt für alle $x \in X$, ist aber je Iterationsschritt aus Speicherplatz- und Rechenzeitgründen unmöglich für alle Zustände - es sind unendlich viele - auszuwerten. Durch den Ansatz einer räumlichen Diskretisierung über Gitter wird der Raum der Approximationen endlich dimensionale Gestalt aufweisen. Der numerische

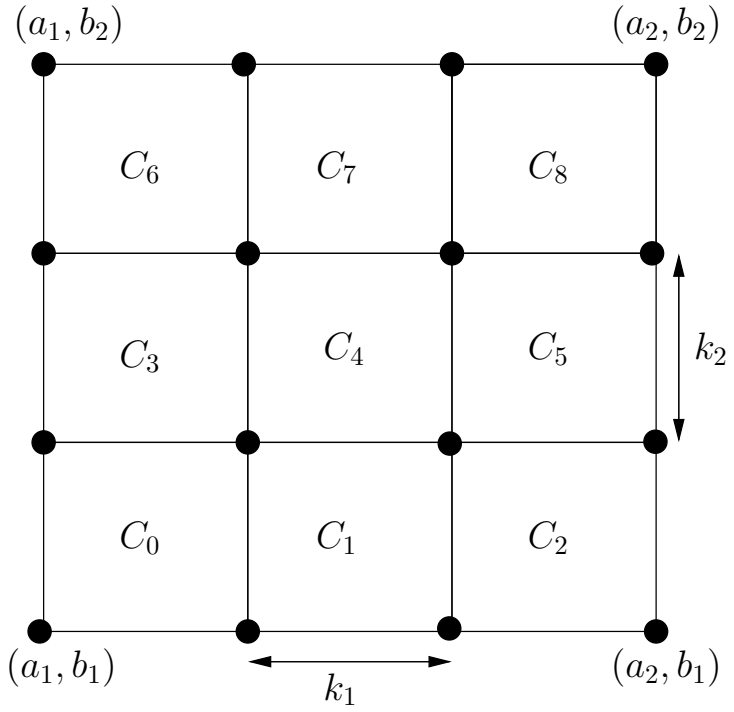

Abbildung 3.2: Rechteckgitter

Aufwand steigt allerdings mit der Dimension n des Zustandsraums, weshalb man vom „curse of dimensionality“ [5] spricht.

Bemerkung 3.26. *Der kompakte Zustandsraum $X \subset \mathbb{R}^n$ wird nun als zweidimensional $(n = 2)$ vorausgesetzt. Nachfolgende Definitionen und Sätze beziehen sich darauf, lassen sich jedoch auch auf $n \geq 3$ verallgemeinern.*

Definition 3.27 (Rechteckgitter). *Sei $X \subset \mathbb{R}^2$ durch $X = [a_1, b_1] \times [a_2, b_2]$, $a_1 < b_1$, $a_2 < b_2$ gegeben. Ein (regelmäßiges) Rechteckgitter Γ auf X ist eine Menge von Rechtecken C_i, $i = 0, \ldots, P-1$, $P = P_1 \cdot P_2$ mit Kantenlängen $k_1 = \frac{b_1 - a_1}{P_1}$ und $k_2 = \frac{b_2 - a_2}{P_2}$, so dass*

$$\bigcup_{i=0}^{P-1} C_i = X \quad \text{und} \quad int\, C_i \cap int\, C_j = \emptyset \quad \forall i, j = 0, \ldots, P-1, i \neq j.$$

Als Eckpunkte bzw. Knoten definieren wir die Punkte $E_i, i = 0, \ldots, N-1$, $N = (P_1 + 1) \cdot (P_2 + 1)$. Der maximale Durchmesser eines Rechtecks wird mit $k = \sqrt{k_1^2 + k_2^2}$ bezeichnet.

Abbildung (3.2) zeigt beispielhaft ein regelmäßiges Gitter mit 16 Knoten. Als Elemente des Funktionenraums für die Approximation von V_∞ , zu welchem die Iterierten in (3.20) gehören, eignen sich affin bilineare Funktionen.

[5] Der Begriff wurde von Richard Bellman selbst kreiert. Er beschreibt damit die Komplexitätszunahme bei dynamischen Optimierungsproblemen aufgrund größerer Raumdimension.

Definition 3.28 (Affin bilineare Funktion). *Sei $X \subset \mathbb{R}^2$. Eine Funktion $W : X \to \mathbb{R}$ heißt affin bilinear, falls Konstanten $\alpha_0, \ldots, \alpha_3 \in \mathbb{R}$ existieren, so dass die Gleichung*

$$W(x) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_1 x_2 \tag{3.21}$$

für alle $x = (x_1, x_2)^T \in \mathbb{R}^2$ gilt.
Für eine rechteckförmige Menge $X \subset \mathbb{R}^2$ definiert man den Raum der stetigen und stückweise affin bilinearen Funktion auf X bezüglich Γ als:

$$\mathcal{W} := \{W : X \to \mathbb{R} | W \text{ ist stetig und } W|_{C_i} \text{ ist affin bilinear für jedes } i\} \tag{3.22}$$

Der Funktionenraum $\mathcal{W}$ weist einige besondere Eigenschaften auf.

Satz 3.29. *Für die Elemente aus $\mathcal{W}$ gilt:*

a) Jedes W aus $\mathcal{W}$ ist eindeutig durch die Werte $W(E_i)$ in den Gittereckpunkten bestimmt.

b) Für jedes Rechteck $C_i = [c_1, d_1] \times [c_2, d_2]$ mit den Eckpunkten $E_{i_0} = (c_1, d_1)^T, E_{i_1} = (d_1, c_2)^T, E_{i_2} = (c_1, d_2)^T, E_{i_3} = (d_1, d_2)^T$ lässt sich $W|_{C_i}$ schreiben als

$$W(x) = \sum_{j=0}^{3} \mu_j(x) W(E_{i_j}). \tag{3.23}$$

*Hierbei gilt für die sogenannten **baryzentrischen Koordinaten** μ_j:*

$$\begin{aligned} \mu_0 &= (1 - y_1(x))(1 - y_2(x)) & \mu_1 &= y_1(x)(1 - y_2(x)) \\ \mu_2 &= (1 - y_1(x)) y_2(x) & \mu_3 &= y_1(x) y_2(x) \end{aligned}$$

mit

$$y_k(x) = \frac{x_k - c_k}{d_k - c_k}, \ k = 1, 2.$$

Insbesondere gilt $\mu_j \geq 0 \ \forall j = 0, \ldots, 3$ und $\sum_{j=0}^{3} \mu_j(x) = 1$.

Beweis:
a): Aus der Definition (3.21) resultiert folgendes 4×4 - Gleichungssystem:

$$\begin{pmatrix} 1 & c_1 & c_2 & c_1 \cdot c_2 \\ 1 & d_1 & c_2 & d_1 \cdot c_2 \\ 1 & c_1 & d_2 & c_1 \cdot d_2 \\ 1 & d_1 & d_2 & d_1 \cdot d_2 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} W(E_{i_0}) \\ W(E_{i_1}) \\ W(E_{i_2}) \\ W(E_{i_3}) \end{pmatrix}$$

Die Determinante ergibt sich zu $(d_1 - c_1)^2 \cdot (d_2 - c_2)^2 \neq 0$.
Da die Eckpunkte paarweise verschieden sind, besitzt das lineare Gleichungssystem genau eine eindeutige Lösung $\alpha \in \mathbb{R}^4$.

b): Durch Rechnung sieht man, dass das angegebene W affin bilinear auf C_i ist und in den Eckpunkten die Werte $W(E_{i_j})$ annimmt.
Zu zeigen bleiben die beiden Eigenschaften für die μ_j:
Dass $\mu_j \geq 0$ gilt, folgt aus der Tatsache, dass die $y_k(x) \in [0,1]$ liegen.
Zudem gilt $\sum_{j=0}^{3} \mu_j(x) = (1 - y_1(x)) \cdot [(1 - y_2(x)) + y_2(x)] + y_1(x) \cdot [(1 - y_2(x)) + y_2(x)] =$
$1 - y_1(x) + y_1(x) = 1$. □

Die Eigenschaft, dass sich jede affin bilineare Funktion durch die Werte in den Knoten bestimmen lässt, ist für die numerische Umsetzung wertvoll. Bei gegebenen N Knoten des Gitters Γ, welches als Rechengebiet dient, lässt sich $\mathcal{W}$ als Vektorraum der Dimension N auffassen. Das heißt also, man wertet bei der Implementierung des Algorithmus die diskrete Hamilton–Jacobi–Bellman–Gleichung (3.13) nur für die Eckpunkte E_i des Gitters aus und führt die Iterationsvorschrift gemäß

$$\tilde{V}_{k+1}(E_i) = \tilde{\mathcal{T}}(\tilde{V}_k)(E_i),^{6} \qquad i = 0, \ldots, N-1,\ k \in \mathbb{N}_0 \tag{3.24}$$

durch.

3.2.4 Fehlerschätzung und adaptive Gitter

Die in (3.27) eingeführten Gitter sind regelmäßig, d.h. alle Rechtecke C_i sind gleich groß. Mithilfe adaptiver Gitter, deren Verwendung in [13] und [14] geschildert wird, lässt sich die Effizienz der numerischen Approximation durch affin bilineare Funktionen deutlich steigern.
Adaptive Gitter haben die Eigenschaft, dass die Rechtecke in manchen Bereichen von Γ feiner, in anderen wiederum gröber sind. Größere Feinheit ist besonders dann vorteilhaft, wenn die Wertefunktion auf diesem Zustandsgebiet stark gekrümmt ist und infolgedessen größere Abweichungen zwischen exakter und approximativer Lösung zu erwarten sind. Die Verfeinerung ist abhängig von der Bestimmung des lokalen Fehlers in den Rechtecken des zugrundeliegenden Gitters Γ.

Daher beginnt dieser Abschnitt, der sich an [10] und [11] orientiert, einleitend mit der Fehleranalyse. Das Ziel besteht darin, den echten Fehler $\| V_\infty - \tilde{V}_\infty \|_\infty$ in der Maximumsnorm $\| \cdot \|_\infty$ abzuschätzen, ohne dabei die exakte Lösung V_∞ zu kennen, was in der Praxis sehr oft der Fall ist. Dabei spielen die in folgender Definition eingeführten lokalen a–posteriori Fehlerschätzer eine wesentliche Rolle.

[6] $\tilde{\mathcal{T}} = \Pi\mathcal{T}$, numerische Projektion von $\mathcal{T}$

Definition 3.30 (Lokaler a–posteriori Fehlerschätzer). *Betrachtet wird das diskrete, optimale Kontrollproblem nach (3.16) auf einem Rechteckgitter Γ mit P Rechtecken C_i. Ein lokaler a-posteriori Fehlerschätzer in der ∞–Norm ist eine Menge von Werten $\eta_1, \ldots, \eta_P$ mit den beiden Eigenschaften*

a) Der Wert η_i ergibt sich aus den Daten des optimalen Kontrollproblems und aus der Funktion $\tilde{V}_\infty$ in einer Umgebung $N(C_i)$ des Rechtecks C_i.

b) Es existieren Konstanten $C_1, C_2 > 0$, so dass für den Wert $\eta := \max_{i=0,\ldots,P-1} \eta_i$ die Abschätzungen

$$C_1\eta \leq \| V_\infty - \tilde{V}_\infty \|_\infty \leq C_2\eta \tag{3.25}$$

gelten. Man nennt den Fehlerschätzer effizient und zuverlässig.

Von lokaler Effizienz des Fehlerschätzers spricht man, falls folgende Ungleichung gilt:

$$C_1\eta_i \leq \sup_{x\in N(C_i)} |V_\infty(x) - \tilde{V}_\infty(x)| \tag{3.26}$$

Unter Effizienz versteht man hierbei, dass aus einem großen Fehlerschätzer eine große Differenz zwischen Approximation und exakter Lösung gefolgert werden kann. Zuverlässig bedeutet, dass ein kleiner Fehlerschätzer einen kleinen Fehler impliziert.
Die Definition deutet an, wie ein lokaler Fehlerschätzer zu einem auf adaptiven Gittern basierenden Algorithmus herangezogen wird. Die lokale Effizienz lässt von einem großen Wert η_i auf einen großen Fehler in der Umgebung des Rechtecks schließen. Dies legt nahe, dass im Gitter Regionen mit großem Schätzwert entsprechend zu verfeinern sind.

Um einen Algorithmus zu verwirklichen, der mittels adaptiver Gitter iterativ die optimale Wertefunktion approximiert, wird ein geeigneter lokaler Fehlerschätzer benötigt. In dessen nachstehende Definition geht der DP Operator $\mathcal{T}$ ein.

Definition 3.31. *Die Funktion $\eta : X \to \mathbb{R}_0^+$ sei gegeben durch*

$$\begin{aligned}\eta(x) &:= \left|\tilde{V}_\infty(x) - \tilde{\mathcal{T}}(\tilde{V}_\infty)(x)\right| \\ &= \left|\tilde{V}_\infty(x) - \left(\sup_{u\in U(x)} \mathbb{E}\{l(x,u) + \beta\tilde{V}_\infty(\varphi(x,u,z)\}\right)\right|.\end{aligned}$$

Auf $\eta(x)$ basierend sei dann der lokale Fehlerschätzer definiert als

$$\eta_i := \max_{x\in C_i} \eta(x) \quad , i = 0, \ldots, P-1. \tag{3.27}$$

Mit genau diesem Fehlerschätzer erhält man für die Abweichung von exakter Lösung und Approximation eine nützliche Abschätzung nach unten und oben.

Satz 3.32 (Fehlerabschätzung). *Es sei $\beta < 1$. Für $\eta = \max_{i=0,\dots,P-1} \eta_i$ mit den lokalen Fehlerschätzern η_i gemäß (3.27) gilt*

$$\frac{1}{2}\eta \leq \| V_\infty - \tilde{V}_\infty \|_\infty \leq \frac{1}{1-\beta}\eta. \tag{3.28}$$

Außerdem gilt

$$\frac{1}{2}\eta_i \leq \sup_{x \in N(C_i)} |\tilde{V}_\infty(x) - \mathcal{T}(\tilde{V}_\infty)(x)| \tag{3.29}$$

mit $N(C_i) := \{y \in X | es\ existiert\ ein\ x \in C_i\ mit\ \| y - x \| \leq M\}$. M ist obere Schranke für $\| \varphi(x,u,z) \|$.

Beweis:
Der Beweis kann in [11] in Kapitel 4 nachgelesen werden.

Die untere Grenze gewährleistet, dass große lokale Fehler eine große Abweichung der Approximation von der exakten Lösung erahnen lassen. Analog impliziert ein geringer oberer Schrankenwert einen kleinen echten Fehler. Zu beachten ist, dass für $\beta \approx 1$ die Lücke zwischen oberer und unterer Schranke groß werden kann. Die Unabhängigkeit der unteren Grenze vom Problemparameter β garantiert jedoch die Effizienz der Methode.
Eine Schwierigkeit bleibt jedoch bestehen. Bei der Berechnung der in (3.27) definierten Fehlerschätzer tritt erneut das Problem auf, dass $\eta(x)$ an unendlichen vielen Punkten auszuwerten ist. Abhilfe schafft hierbei die approximative Maximierung über eine endliche Menge an Textpunkten pro Rechteck. Darauf wird beim Algorithmus eingegangen.

Adaptive Gitter

Kehren wir nun zum Ausgangspunkt der numerischen dynamischen Programmierung zurück, der Iteration gemäß (3.20) zur Bestimmung der optimalen Wertefunktion.

Adaptive Gitter sind Folge einer im Laufe der Iteration durchgeführten Verfeinerung und Vergröberung, die in Abhängigkeit der lokalen Fehlerschätzungen im Gitter erfolgen.
In jedem Iterationsschritt $k \in \mathbb{N}_0$ sollte idealerweise die Gleichung

$$\tilde{V}_{k+1}(x) = \tilde{\mathcal{T}}\tilde{V}_k(x) \quad \forall\, x \in X \tag{3.30}$$

erfüllt sein, jedoch aber wird (3.30) nur von den Eckpunkten E_i des Gitters erfüllt.
Für alle Punkte $x \neq E_i$ gilt:

$$\tilde{V}_{k+1}(x) \neq \tilde{\mathcal{T}}\tilde{V}_k(x).$$

Um zu überprüfen, wie stark Gleichung (3.30) verletzt ist, wählt man im Gitter Γ_k eine ausgezeichnete endliche Menge von Testpunkten $x_j, j = 0, \dots M$ und wertet die lokalen Fehlerschätzer

$$\eta_j := |\tilde{V}_{k+1}(x_j) - \tilde{\mathcal{T}}\tilde{V}_k(x_j)|$$

aus. Übersteigt der Fehler eine im Algorithmus vorgegebene Fehlerschranke, so wird der Testpunkt als neuer Knoten in das Gitter eingefügt und ein entsprechend feineres Gitter resultiert. Um die Gittergeometrie aufrecht zu erhalten, werden typischerweise Testpunkte als Kantenmittelpunkte oder Mittelpunkt des Rechtecks C_i gewählt. Abbildung (3.3) verdeutlicht die Auswahl der Testpunkte.

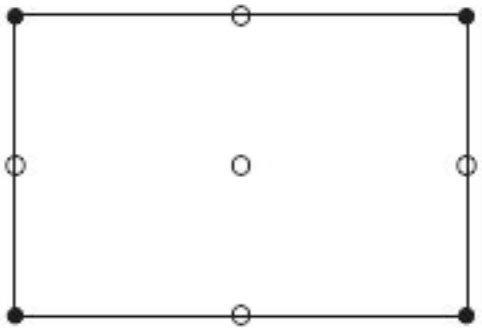

Abbildung 3.3: Mögliche Testpunkte eines Rechtecks C_i

Entsprechend lässt sich auswerten, ob ein existierender Knoten E_j notwendig ist, indem man testet, wie sein Wert $\tilde{V}_k(E_j)$ vom interpolierten Wert $\tilde{V}_k^*(E_j)$ des gröberen Gitters $\Gamma^* = \Gamma \setminus \{E_j\}$ abweicht.
Ist die Abweichung

$$d_j := |\tilde{V}_k^*(E_j) - \tilde{V}_k(E_j)|$$

kleiner als eine vorgegebene Vergröberungstoleranz $ctol > 0$, so wird der Eckpunkt E_j entfernt, ohne dass die Approximation qualitativ schlechter wird. Dafür wird Speicherplatz gespart und die Rechenzeit verkürzt.

Die Gitteradaption führt somit auf den folgenden Algorithmus, der angelehnt ist an eine Variante aus [14].[7]

[7] Implementierungsdetails werden in Kapitel 5 erklärt.

Algorithmus zur Berechnung der optimalen Wertefunktion (3.31)

1. Wähle ein Startgitter Γ_0 und setze $k := 0$. Definiere einen Verfeinerungsparameter $\rho \in (0, 1)$ und eine Fehlerschranke $rtol$.
2. Berechne $\mathcal{T}(\tilde{V}_k)(E_i)$ auf dem Gitter Γ_k für alle Knoten E_i (Anwendung des Optimalitätsprinzips).
3. Werte die lokalen Fehlerschätzer η_j für die erzeugten Testpunkte x_j aus. Gilt $\eta_j < rtol \ \forall j$, beende den Algorithmus.
4. Füge diejenigen Testpunkte mit $\eta_j \geq \rho \cdot \max_j \eta_j$ in das Gitter Γ_k ein und setze $k := k + 1$. Gehe zu Schritt (1).

Alternativ zur Spezifikation von $rtol$, kann man die Verfeinerung abbrechen, falls eine Maximalzahl an Rechtecken (Knoten) erreicht wurde. Sicherheitshalber sollte eine solche obere Schranke zu Beginn des Algorithmus festgelegt sein.
Zudem kann der Algorithmus dahingehend abgeändert werden, dass nach Schritt (2) eine Gittervergröberung erfolgt. Im Falle der späteren Anwendungsbeispiele wird diese Modifikation nicht vorgenommen.

Auswertung des DP Operators $\mathcal{T}$

Die zentrale Rechenleistung in obigem Algorithmus stellt die Auswertung des DP Operators (3.18) dar. Für jeden Knotenpunkt und für die Testpunkte ist in jedem Schritt $k \in \mathbb{N}_0$ die rechte Seite der Bellman–Gleichung auszuwerten.
Dabei treten zwei Probleme auf, die aber gelöst werden können.

Auswertungsprobleme:

1. Auswertung des Maximums über alle zulässigen Kontrollen u
2. Berechnung des Erwartungswerts bei gegebener Dynamik φ mit stochastischem Einfluss

Das erste Problem wird dadurch beseitigt, dass die als kompakt vorausgesetzte Steuerungsmenge $U(x)$ diskretisiert wird. Betrachtet wird eine endliche Menge $\tilde{U} \subseteq U(x_0)$ mit der Eigenschaft

$$\epsilon_u := \max_{u \in U(x)} \min_{\tilde{u} \in \tilde{U}} \| u - \tilde{u} \|$$

Zugleich lässt sich damit numerisch eine approximative Kontrolle $\tilde{u}$ berechnen. Für einen gegebenen Zustand x erhält man „online“ den optimalen Kontrollwert $\tilde{u}(x)$ durch

$$\tilde{u}(x) = \underset{\tilde{u}\in\tilde{U}}{\operatorname{argmax}}\, \mathbb{E}\{l(x,\tilde{u}) + \beta\tilde{V}_k(\varphi(x,\tilde{u},z)\}. \tag{3.32}$$

Das maximierende Argument lässt sich meist schnell berechnen, weswegen der Rechenaufwand für die Gewinnung approximativer, optimaler Feedbacks vertretbar ist.

Um das zweite Problem bei der Berechnung von $\mathcal{T}$ zu beheben, wird der Erwartungswertausdruck

$$\mathbb{E}\{l(x,\tilde{u}) + \beta\tilde{V}_k(\varphi(x,\tilde{u},z)\}$$

genauer betrachtet.

Der stochastische Einfluss z als Argument der Dynamik φ ist normalverteilt. Die stetige Verteilungsdichtefunktion $p_Z(x)$ ist numerisch auswertbar und somit ergibt sich

$$\mathbb{E}\{l(x,\tilde{u}) + \beta\tilde{V}_k(\varphi(x,\tilde{u},z)\} = l(x,\tilde{u}) + \beta\cdot\int_z \tilde{V}_k(\varphi(x,\tilde{u},z))p_Z(z)dz.$$

Um den Wert des Integrals zu approximieren, wird eine numerische Quadraturregel herangezogen. Im Rahmen dieser Arbeit wird die zusammengesetzte Trapezregel verwendet.[8]

3.3 Die Hamilton-Funktion und das stochastische Maximumprinzip

Dieser Abschnitt, der als Unterbau für das nächste Kapitel aufzufassen ist, erläutert das Konstrukt der Hamilton-Funktion und leitet ein Optimalitätsprinzip her, welches den Namen stochastisches Maximumprinzip trägt.
Es wurde laut [7], auf dessen Arbeit in diesem Abschnitt verwiesen wird, seit Mitte der Sechzigerjahre auf ökonomische Problemstellungen angewendet und ist seitdem als weitverbreitete Lösungsmethode bekannt. Im Rahmen dieser Arbeit stellt sie neben der Dynamischen Programmierung den zweiten Lösungsansatz dar. Zur Formulierung des Prinzips, welches später im Rahmen der Perturbationsmethoden als Ausgangspunkt für die Approximationen der Kontrolle und der Dynamik dient, benötigen wir vorab die Definition der Hamilton-Funktion.

Definition 3.33 (Hamilton-Funktion). *Für $x,\lambda\in\mathbb{R}^n$ und $u\in U\subseteq\mathbb{R}^d$ definieren wir*

$$H^t(x,\lambda,u) = \mathbb{E}_t\left\{\beta^t\cdot l(x,u) + \lambda^T\varphi(x,u,Z_t)\right\}, \tag{3.33}$$

wobei φ, l und die Z_t aus der Definition des optimalen Steuerungsproblems (3.16) stammen.

[8] Eine kurze Erläuterung ist in Abschnitt (5.2) bei den Implementierungsdetails zu finden.

Bemerkung 3.34. *Der Vektor $\lambda \in \mathbb{R}^n$ wird **Kozustand** bzw. **adjungierte Variable** genannt. H wird genauer auch als Hamilton-Funktion in Gegenwartsschreibweise (present-value) bezeichnet. Dies ist im Faktor β^t begründet.*
Der Operator $\mathbb{E}_t$ bezeichnet den Erwartungswert, abhängig von allen zum Zeitpunkt t vorhandenen Informationen, d.h. es liegen die Zustände $x_0, x_1, \dots x_t$ vor.

Nachstehend wird angenommen, dass sich die Bildung des Erwartungswertes und die Ableitung vertauschen lassen. Unter Verwendung der Hamilton-Funktion lassen sich notwendige Optimalitätsbedingungen herleiten.

Satz 3.35 (Stochastisches Maximumprinzip). *Die optimale Wertefunktion V_∞ und die Hamiltonfunktionen H^t seien stetig differenzierbar und u_t sei eine optimale Entscheidungsfolge sowie X_t die zugehörige Zustandsfolge. Dann existiert ein $\mathbb{R}^n$ - wertiger Prozess λ_t, so dass für alle Zeitpunkte $t \in \mathbb{N}_0$ gilt:*

$$E_t\{X_{t+1}\} = \left(\frac{\partial H^t}{\partial \lambda}\right)(X_t, \lambda_{t+1}, u_t) \qquad \textbf{\textit{(Zustandsgleichung)}}$$

$$E_t\{\lambda_t\} = \left(\frac{\partial H^t}{\partial x}\right)(X_t, \lambda_{t+1}, u_t) \qquad \textbf{\textit{(Kozustandsbedingung)}}$$

$$0 = \left(\frac{\partial H^t}{\partial u}\right)(X_t, \lambda_{t+1}, u_t) \qquad \textbf{\textit{(Extremalbedingung)}}$$

Beweis:
Zuerst wird die Zustandsgleichung bewiesen. Es gilt:

$$\left(\tfrac{\partial H^t}{\partial \lambda}\right)(X_t, \lambda_{t+1}, u_t) = \mathbb{E}_t\{\varphi(X_t, u_t, Z_t)\} = \mathbb{E}_t\{X_{t+1}\}$$

Beweis der Kozustandsbedingung:
Zunächst setzen wir für die Kozustandsvariable λ_t[9]:

$$\lambda_t := \beta^t \left(\frac{\partial V_\infty}{\partial x}(X_t)\right)^T \tag{3.34}$$

Damit ergibt sich:

$$\begin{aligned}
\left(\frac{\partial H^t}{\partial x}\right)(X_t, \lambda_{t+1}, u_t) &= \mathbb{E}_t\left\{\beta^t \frac{\partial l}{\partial x}(X_t, u_t) + \lambda_{t+1}^T \frac{\partial \varphi}{\partial x}(X_t, u_t, Z_t)\right\} \\
&= \mathbb{E}_t\left\{\beta^t \frac{\partial l}{\partial x}(X_t, u_t) + \beta^{t+1}\frac{\partial V_\infty}{\partial x}(X_{t+1})\frac{\partial \varphi}{\partial x}(X_t, u_t, Z_t)\right\} \\
&= \frac{\partial}{\partial x}\mathbb{E}_t\left\{\beta^t\, l(X_t, u_t) + \beta^{t+1} V_\infty(\varphi(X_t, u_t, Z_t))\right\} \\
&= \frac{\partial}{\partial x}\max_{u\in U}\mathbb{E}_t\left\{\beta^t\, l(X_t, u) + \beta^{t+1} V_\infty(\varphi(X_t, u_t, Z_t))\right\} \\
&= \frac{\partial}{\partial x}\mathbb{E}_t\left(\beta^t \cdot V_\infty(X_t)\right) = \mathbb{E}_t \lambda_t^T.
\end{aligned}$$

[9] Wahl von λ_t wird auf Seite 35 näher erläutert

Im letzten Schritt wurde das Bellman'sche Optimalitätsprinzip für V_∞ verwendet.
Abschließend der Beweis der Extremalbedingung:

$$\begin{aligned}\left(\frac{\partial H^t}{\partial u}\right)(X_t, \lambda_{t+1}, u_t) &= \mathbb{E}_t\left\{\beta^t \frac{\partial l}{\partial u}(X_t, u_t) + \lambda_{t+1}^T \frac{\partial \varphi}{\partial u}(X_t, u_t, Z_t)\right\} \\ &= \mathbb{E}_t\left\{\beta^t \frac{\partial l}{\partial u}(X_t, u_t) + \beta^{t+1}\frac{\partial V_\infty}{\partial x}(X_{t+1})\frac{\partial \varphi}{\partial u}(X_t, u_t, Z_t)\right\} \\ &= \frac{\partial}{\partial u}\mathbb{E}_t\{\beta^t\, l(X_t, u_t) + \beta^{t+1} V_\infty(\varphi(X_t, u_t, Z_t))\} = 0,\end{aligned}$$

da u_t den letzten Ausdruck maximiert. □

Bemerkung 3.36. *Dieser Satz gilt unter der Bedingung, dass keine Beschränkung der Steuervariablen u vorliegt, d.h. $U = \mathbb{R}^d$. Die geforderte Differenzierbarkeit der optimalen Wertefunktion V_∞ erleichtert hier den Beweis, wird in anderen Fassungen des Satzes (in [7] nachzulesen) aber nicht vorausgesetzt. Wird die Hamilton-Funktion als konkav vorausgesetzt, so sind die genannten Bedingungen sogar hinreichend.*

Der wesentliche Nutzen des Satzes besteht für uns darin, dass sich für ein optimales Kontrollproblem (3.16) Bedingungen für das Gleichgewicht ergeben. Es zeigt sich in vielen Kontrollproblemen, dass die Folge $(x(t), u(t))$ für $t \longrightarrow \infty$ gegen einen stationären Punkt $(\bar{x}, \bar{u})$ konvergiert. Bei Problemen auf unendlichem Zeithorizont hält sich die Lösung einen Großteil des Zeitintervalls nahe des Gleichgewichts $(\bar{x}, \bar{u})$ auf.
Das Gleichgewicht ist mit den notwendigen Optimalitätsbedingungen des stochastischen Maximumprinzips berechenbar. Die Bedingungen des Satzes (3.35) lassen sich derart umstellen, dass sich die folgenden Gleichgewichtsbedingungen herleiten lassen.

Definition 3.37 (Gleichgewichtsbedingungen). *Die Gleichungen*

$$\mathbb{E}_t\left\{f(X_t, X_{t+1}, u_t, u_{t+1}, \lambda_t, \lambda_{t+1})\right\} = 0 \quad \forall t \in \mathbb{N}_0, \tag{3.35}$$

wobei f definiert ist als

$$f := \begin{bmatrix} X_{t+1} - \left(\frac{\partial H^t}{\partial \lambda}\right)(X_t, \lambda_{t+1}, u_t) \\ \lambda_t - \left(\frac{\partial H^t}{\partial x}\right)(X_t, \lambda_{t+1}, u_t) \\ \frac{\partial H^t}{\partial u}(X_t, \lambda_{t+1}, u_t) \end{bmatrix}, \tag{3.36}$$

stellen die Gleichgewichtsbedingungen für stochastische, dynamische Kontrollsysteme dar.

Diese Bedingungen für den „steady state" stellen die Schnittstelle zur Theorie der Perturbationsmethoden dar und werden im vierten Kapitel aufgegriffen.

Ökonomische Deutung des Maximumprinzips

Das Maximumprinzip lässt sich mit Blick auf die Anwendung bei Wachstumsmodellen ökonomisch deuten. Es gilt vorrangig, die zur Formulierung der Hamilton-Funktion neu eingeführte Kozustandsvariable λ zu interpretieren. Sie wurde im Beweis zum stochastischen Maximumprinzip gemäß Gleichung (3.34) gesetzt.

Wir treffen hierzu folgende Annahmen:

- Der Zustand $x \in \mathbb{R}^1$ bezeichnet den Kapitalstock.
- Die eindimensionale Kontrollvariable $u \in \mathbb{R}^1$ entspricht dem Konsum.
- Die Ertragsfunktion $l(x, u)$ sei eine Nutzenfunktion.

Demnach lässt sich λ_t als **Schattenpreis** auffassen. Die Variable bewertet die Änderung des optimalen Prozesswertes $V_\infty(X_t)$, welche aus einer Veränderung der Zustandsvariable x_t um eine Einheit resultiert.
Dabei wird unterstellt, dass sich der Entscheidungsträger des Kontrollproblems für den restlichen Planungszeitraum optimal verhält.

Der zweite Summand der Hamilton-Funktion H^t

$$\lambda^T \varphi(x, u, Z_t) = \sum_{j=1}^{n} \lambda_j \cdot \varphi_j(x, u, Z_t)$$

beschreibt die Wertänderung des Kapitalstocks in Abhängigkeit von der Wahl des Konsums u.

Die Hamilton-Funktion H^t bildet die „totale Auswirkung“ der Konsumentscheidung u zum Zeitpunkt t nach. Sie lässt sich in einen direkten und indirekten Effekt zerlegen.

- Der direkte Effekt der Entscheidung, u Einheiten zu konsumieren besteht in Verbindung mit dem herrschenden Zustand x darin, dass der Nutzen $l(x, u)$ erzielt wird, der auf den Zeitpunkt $t = 0$ diskontiert wird (erster Summand von H^t).
- Indirekten Einfluss hat die Wahl von u auf die Änderung des Kapitalbestands. Es ergibt sich nach Wahl von u_t der neue Zustand x_{t+1}, der ebenfalls diskontiert zum Schattenpreis λ_t bewertet wird. Das Produkt beider Größen ergibt den zweiten Summanden der Hamilton-Funktion.

Somit bemisst die Hamilton-Funktion den Gesamtnutzen (verallgemeinert einen ökonomischen Wert), der sich aus Addition des unmittelbaren Nutzens und dem indirekten Wertzuwachs aufgrund der Zustandsänderung ergibt. Die Extremalbedingung verdeutlicht somit, dass bei der Wahl der Steuerung u zu jedem Zeitpunkt t dasjenige u_t zu wählen ist, welches beiden Effekten gerecht wird.

4 Perturbationsmethoden zweiter Ordnung

In diesem Kapitel wird das Perturbationsverfahren, welches als Alternative zur dynamischen Programmierung eine weitere Lösungsmöglichkeit für optimale Kontrollprobleme darstellt, erläutert. Das Kapitel ist in drei Abschnitte unterteilt. Der erste Abschnitt stellt die getroffenen Annahmen und Notationen des Perturbationsmodells vor, der zweite Teil erläutert die Herleitung der Koeffizienten für die Taylor–Approximationen von Dynamik und Kontrolle sowie mit besonderem Augenmerk diejenige für die optimale Wertefunktion des Modells. Bei der Annäherung von V_∞ wird zwischen einer indirekten und einer neuartigen, direkten Approximationsmethode unterschieden. Die numerische Umsetzung auf dem Rechner in Form der Perturbationsroutinen in MATLAB und MAPLE behandelt abschließend der letzte Teilabschnitt.
Die Ausarbeitungen [22], [23] und [24] von Schmitt–Grohe und Uribé sowie die Arbeit [5] von Collard & Juillard wurden als maßgebende Literaturquellen herangezogen.

4.1 Annahmen des Perturbationsmodells

Die aus den Bedingungen des stochastischen Maximumprinzips hergeleiteten Gleichgewichtsbedingungen (3.35) von Abschnitt (3.3) werden an dieser Stelle aufgegriffen. Sie sind der Ausgangspunkt für das Perturbationsmodell, das auf den Ausführungen von Schmitt–Grohe und Uribé in [24] fußt. Voraussetzung für die Anwendbarkeit der Perturbationsmethoden ist nämlich die Kenntnis eines Gleichgewichts.

Die Gleichgewichtsbedingungen für dynamische, stochastische Gleichgewichtsmodelle sind allgemein formuliert als

$$\mathbb{E}_t f(x_t, x_{t+1}, u_t, u_{t+1}) = 0 \quad \forall t \in \mathbb{N}_0. \tag{4.1}$$

Die Gleichungen entsprechen den Gleichungen von (3.35), die über die Hamilton–Funktion aufgestellt wurden. Jedoch fehlt der Prozess λ als weiteres Argument. Dies ist darin begründet, dass sich für viele Modelle, auch für das Ramsey Modell, die Gleichungen derart zusammenfassen lassen, dass sich λ_t und λ_{t+1} herausheben.

Annahmen:
Analog wie in Definition (3.12) ist der reelle Zustandsvektor x_t von der Dimension n und der Kontrollvektor u_t im Vektorraum $\mathbb{R}^d$ beheimatet. Annahme ist nun, dass sich der Zustand x_t wie folgt in einen endogenen und einen exogenen Teil partitionieren lässt:

$$x_t = [x_t^1, x_t^2],\ x_t \in \mathbb{R}^{n_1+n_2};\ n = n_1 + n_2$$

x_t^1 fasst zu jedem Zeitpunkt die endogen bestimmten Variablen zusammen, x_t^2 die exogenen Zustandsvariablen.
Desweiteren wird angenommen, dass der exogene Zustandsvektor x_t^2 folgendem stochastischen Prozess folgt:

$$x_{t+1}^2 = Mx_t^2 + \sigma\Sigma\epsilon_{t+1}$$

Es handelt sich dabei genauer um einen **AR(1) Prozess**, einen sogenannten autoregressiven Prozess erster Ordnung[1]. Es gilt:

- Die Eigenwerte der Matrix M sind betragsmäßig kleiner als eins (Stationaritätsbedingung des Prozesses).
- Der Zufallsvektor ϵ_t ist $N(0, I)$ multivariat normalverteilt, wobei der Erwartungswert 0 ist und die Kovarianzmatrix der Einheitsmatrix I gleicht. Man bezeichnet den stochastischen Prozess ϵ_t als „Weißes Rauschen“.
- $\sigma \geq 0$ als Parameter für die Störung von außen und die Koeffizientenmatrix Σ sind bekannt.

Es wird weiterhin angenommen, dass f bekannt und mindestens zweimal stetig differenzierbar bezüglich der Variablen x, x', u, u' ist.
Hierbei ersetzt das Hochkomma $'$ den Zeitindex $t + 1$.

Annahmen an die Lösungen des Modells:
Mit Einführung der Notation für die Steuerungsfunktion $g : \mathbb{R}^n \times \mathbb{R}_0^+ \to \mathbb{R}^d$ und die Dynamik $h : \mathbb{R}^n \times \mathbb{R}_0^+ \to \mathbb{R}^n$ ist die Lösung des Modells von der Gestalt:

$$\begin{aligned} u_t &= g(x_t, \sigma) \\ x_{t+1} &= h(x_t, \sigma) + \eta\sigma \cdot \epsilon_{t+1} \quad \text{mit } \eta = \begin{pmatrix} 0 \\ \Sigma \end{pmatrix} \end{aligned} \tag{4.2}$$

Wir setzen voraus, dass g und h als Funktionen im Zustand $x \in \mathbb{R}^n$ und im Perturbationsparameter σ „hinreichend“ glatt sind. Sie seien mindestens zweimal stetig partiell nach ihren Argumenten

[1] Erste Ordnung: Der nachfolgende Zustand X_{t+1} hängt nur vom direkten Vorgänger X_t ab.

differenzierbar.
Damit kann die Ausgangsgleichung (4.1) umgeschrieben werden zu

$$\mathbb{E}_t \{f(x, h(x,\sigma) + \eta\sigma \cdot \epsilon', g(x,\sigma), g(h(x,\sigma) + \eta\sigma \cdot \epsilon', \sigma))\} = 0. \tag{4.3}$$

Man definiert dafür zusammenfassend die „Nullfunktion“ F,

$$F(x,\sigma) := \mathbb{E}_t \{f(x, h(x,\sigma) + \eta\sigma \cdot \epsilon', g(x,\sigma), g(h(x,\sigma) + \eta\sigma \cdot \epsilon', \sigma))\} = 0. \tag{4.4}$$

4.2 Herleitung der Koeffizienten der Taylor - Approximationen

In diesem Abschnitt werden die einzelnen Koeffizienten für die Lösungen g und h des Perturbationsmodells (4.2) hergeleitet. Beide Funktionen werden in Form eines Taylorpolynoms zweiten Grades angenähert.
Sobald beide Approximationen bekannt sind, kann die optimale Wertefunktion des betrachteten Kontrollproblems berechnet werden, ebenfalls als eine Taylorreihe.
Eine Gemeinsamkeit zu den Methoden der dynamischen Programmierung besteht darin, dass hierfür auch die Bellman–Gleichung (3.13) ausgenutzt wird. Jedoch ist sie nicht Ausgangspunkt einer Iteration wie in (3.20), sondern sie wird auf beiden Seiten quadratisch um den Gleichgewichtspunkt entwickelt. Da es sich bei den Approximationen um Taylorpolynome handelt, folgt einleitend die Aussage des Taylor'schen Satzes, der einen der bedeutendsten Sätze der Analysis verkörpert.

Satz 4.1 (Satz von Taylor). *Sei $f : I \to \mathbb{R}^1$, $I \subseteq \mathbb{R}^n$ offen. Sei $f \in C^{k+1}(I)$, d.h $(k+1)$–mal stetig differenzierbar, sowie $x_0 \in I$. Dann gilt für alle $x \in I$:*

$$\begin{aligned} f(x) =& f(x_0) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x_0)(x_i - x_i^0) + \\ &+ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j}(x_0)(x_i - x_i^0)(x_j - x_j^0) \\ &\vdots \\ &+ \frac{1}{k!}\sum_{i_1=1}^{n} \dots \sum_{i_k=1}^{n} \frac{\partial^k f}{\partial x_{i_1} \dots \partial x_{i_k}}(x_0)(x_{i_1} - x_{i_1}^0) \dots (x_{i_k} - x_{i_k}^0) \\ &+ \Theta(\| x - x_0 \|^{k+1}) \end{aligned}$$

Bemerkung 4.2. *Die Taylor-Approximation von f basiert auf dem Entwicklungspunkt x_0 und benutzt Ableitungsinformationen an dieser Stelle zur Konstruktion eines Polynoms vom Grad k. Für die nachfolgenden Ausführungen werden die Fälle $\mathbf{k} \in \{\mathbf{1}, \mathbf{2}\}$ in Betracht gezogen.*

4.2.1 Taylor-Approximation von Kontrolle g und Dynamik h

Ziel der Perturbationsmethoden ist es, lokale Approximationen der Kontrollfunktion g und der Dynamik h aus dem Perturbationsmodell (4.2) zu finden. Lokal drückt dabei aus, dass die Approximationen in der Umgebung eines speziellen Punktes $(\bar{x}, \bar{\sigma})$, dessen Existenz Voraussetzung für die Anwendung dieser Methode ist, gültig sind.

Der erste Schritt, die Berechnung des optimalen Gleichgewichts $(\bar{x}, \bar{u})$, besteht darin, den steady state $\bar{x}$ sowie $\bar{u}$ für das ungestörte, deterministische System, d.h. $\sigma = 0$, zu bestimmen. Dies geschieht über die Optimalitätsbedingungen des Maximumprinzips (3.35).
Im Perturbationsmodell gilt dann:

$$\bar{u} = g(\bar{x}, 0) \quad \text{und} \quad \bar{x} = h(\bar{x}, 0)$$

Das Paar $(\overline{x}, 0) \in \mathbb{R}^n \times \mathbb{R}_0^+$ dient als Entwicklungspunkt der Taylor-Approximationen g und h.

Um den Überblick über die auftretenden Ableitungen im mehrdimensionalen Fall zu behalten, wird vorab folgende Notation [2] eingeführt und anschließend konsequent beibehalten.

Notation. *Sei f eine beliebig oft differenzierbare mehrdimensionale Funktion, $f : \mathbb{R}^n \to \mathbb{R}^m$, dann werden deren partielle Ableitungen wie folgt notiert:*

$$[f_x]^i_j := \frac{\partial f_i}{\partial x_j}$$

Sei g eine weitere differenzierbare Funktion mit gleichem Definitionsbereich. Für das Produkt zweier Ableitungen verwenden wir die Kurzform:

$$[f_x]^i_a [g_x]^a_j := \sum_a \frac{\partial f_i}{\partial x_a} \frac{\partial g_a}{\partial x_j}$$

Bei zweiten partiellen Ableitungen einer beliebigen Funktion f tritt folgende Notation auf:

$$[f_{xx}]^i_{ab} := \frac{\partial f_i}{\partial x_a \partial x_b}$$

Da das Argument x mehrdimensional ist, handelt es sich bei dem Ausdruck um einen Tensor dritter Stufe. Mit $[f_{xx}]^i_{ab}$ ist dann der Eintrag des dreidimensionalen Arrays f_{xx} in Zeile i, Spalte a und auf Seite b bezeichnet.

[2] Geht auf Kenneth Judd, Autor von [15] zurück.

Die gesuchte Approximation von g mittels einer Taylorreihe der Ordnung zwei hat dann in obiger Notation die Form:

$$\begin{aligned}[g(x,\sigma)]^i =& [g(\bar{x},0)]^i + [g_x(\bar{x},0)]^i_a \cdot (x-\bar{x})_a + [g_\sigma(\bar{x},0)]^i \cdot (\sigma) \\ &+ \frac{1}{2} \cdot [g_{xx}(\bar{x},0)]^i_{ab} \cdot (x-\bar{x})_a \cdot (x-\bar{x})_b \\ &+ \frac{1}{2} \cdot [g_{x\sigma}(\bar{x},0)]^i_a \cdot (x-\bar{x})_a \cdot (\sigma) \\ &+ \frac{1}{2} \cdot [g_{\sigma x}(\bar{x},0)]^i_a \cdot (x-\bar{x})_a \cdot (\sigma) \\ &+ \frac{1}{2} \cdot [g_{\sigma\sigma}(\bar{x},0)]^i \cdot (\sigma) \cdot (\sigma) \\ &\text{mit } i = 1,\dots,d \quad a,b = 1,\dots,n\end{aligned}$$

Die Approximation der Dynamik h des Perturbationsmodells nach Taylor sieht analog wie folgt aus:

$$\begin{aligned}[h(x,\sigma)]^j =& [h(\bar{x},0)]^j + [h_x(\bar{x},0)]^i_a \cdot (x-\bar{x})_a + [h_\sigma(\bar{x},0)]^j \cdot (\sigma) \\ &+ \frac{1}{2} \cdot [h_{xx}(\bar{x},0)]^j_{ab} \cdot (x-\bar{x})_a \cdot (x-\bar{x})_b \\ &+ \frac{1}{2} \cdot [h_{x\sigma}(\bar{x},0)]^j_a \cdot (x-\bar{x})_a \cdot (\sigma) \\ &+ \frac{1}{2} \cdot [h_{\sigma x}(\bar{x},0)]^j_a \cdot (x-\bar{x})_a \cdot (\sigma) \\ &+ \frac{1}{2} \cdot [h_{\sigma\sigma}(\bar{x},0)]^j \cdot (\sigma) \cdot (\sigma) \\ &\text{mit } j = 1,\dots,n \quad a,b = 1,\dots,n\end{aligned}$$

Die Ermittlung der entsprechenden Koeffizienten $[g_x(\bar{x},0)], [h_x(\bar{x},0)], [g_\sigma(\bar{x},0)], [h_\sigma(\bar{x},0)]$ der linearen Terme sowie der Koeffizienten $[g_{xx}(\bar{x},0)], [h_{xx}(\bar{x},0)]$, usw. der quadratischen Terme ist nötig zum Aufstellen der Polynome. Um dies zu bewerkstelligen, erinnert man sich an die „Nullfunktion“ F von (4.4) und deren besondere Eigenschaft:

$$F(x,\sigma) \equiv 0 \;\; \forall x \in \mathbb{R}^n, \sigma \geq 0.$$

Daraus folgt für deren Ableitungen:

$$F_{x^k\sigma^j}(x,\sigma) = 0 \;\; \forall x \in \mathbb{R}^n, \sigma \geq 0, k,j \in \{1,2\}. \tag{4.5}$$

Bemerkung 4.3. *$F_{x^k\sigma^j}$ bezeichnet die k-malige Ableitung in x–Richtung und die j-malige in σ–Richtung. Zu beachten ist ferner, dass für $\sigma = 0$, d.h. wenn das Modell störungsfrei ist, der bedingte Erwartungswert $\mathbb{E}_t$ bei F verschwindet.*

Durch Differentiation der Funktion F, die aus den Gleichgewichtsbedingungen des stochastischen Maximumprinzips hervorging, können wir nun die gesuchten Koeffizienten berechnen.

Bestimmung linearer Koeffizienten

Leitet man F nach der Zustandsvariablen x gemäß der Kettenregel ab, so ergeben sich folgende Gleichungen, in denen $[g_x], [h_x]$ [3] auftreten :

$$\begin{aligned}[F_x(\bar{x},0)]^i_j &= [f_{u\prime}]^i_a[g_x]^a_b[h_x]^b_j + [f_u]^i_a[g_x]^a_j + [f_{x\prime}]^i_b[h_x]^b_j + [f_x]^i_j \\ &= 0 \\ i &= 1,\ldots,n+d; \quad j,b = 1,\ldots,n; \quad a = 1,\ldots,d.\end{aligned} \tag{4.6}$$

Bemerkung 4.4. *Bei obigem Gleichungssystem handelt es sich um $(n+d)\cdot n$ Gleichungen mit den $(n+d)\cdot n$ Unbekannten, den gesuchten Koeffizienten $g_x(\bar{x},0)$ und $h_x(\bar{x},0)$. Die Ableitungen von f ausgewertet an der Stelle $(\bar{x},\bar{x},\bar{u},\bar{u})$ sind dabei bekannt.*
Um dieses Gleichungssystem zu lösen, wird es zunächst umgestellt:

$$\begin{pmatrix} f_{x\prime} & f_{u\prime} \end{pmatrix} \cdot \begin{pmatrix} I \\ g_x \end{pmatrix} \cdot h_x = -\begin{pmatrix} f_x & f_u \end{pmatrix} \cdot \begin{pmatrix} I \\ g_x \end{pmatrix}$$

Diese Zerlegung führt auf ein verallgemeinertes Eigenwertproblem. Wie es gelöst wird, kann in [2] nachgelesen werden.

Um die linearen Koeffizienten $[g_\sigma], [h_\sigma]$ zu erhalten, bildet man die erste Ableitung von F nach dessen zweitem Argument σ. Dies führt auf folgendes lineares Gleichungssystem:

$$\begin{aligned}[F_\sigma(\bar{x},0)]^i &= \mathbb{E}_t\{[f_{u\prime}]^i_a[g_x]^a_b[h_\sigma]^b + [f_{u\prime}]^i_a[g_x]^a_b[\eta]^b_s[\epsilon']^s + [f_{u\prime}]^i_a[g_\sigma]^a + [f_u]^i_a[g_\sigma]^a \\ &\quad + [f_{x\prime}]^i_b[h_\sigma]^b + [f_{x\prime}]^i_b[\eta]^b_s[\epsilon']^s\} \\ &= [f_{u\prime}]^i_a[g_x]^a_b[h_\sigma]^b + [f_{u\prime}]^i_a[g_\sigma]^a + [f_u]^i_a[g_\sigma]^a + [f_{x\prime}]^i_b[h_\sigma]^b \\ &= 0 \\ i &= 1,\ldots,n+d; \quad a = 1,\ldots,d; \quad b = 1,\ldots,n; \quad s = 1,\ldots,n_2.\end{aligned} \tag{4.7}$$

Bemerkung 4.5. *Dieses Gleichungssystem, das sich aufgrund der Tatsache $\mathbb{E}[\epsilon] = 0$ vereinfacht, ist linear und homogen in den Koeffizienten $[g_\sigma]$ und $[h_\sigma]$. Falls eine eindeutige Lösung existiert, folgt somit unmittelbar*

$$g_\sigma(\bar{x},0) = 0, \quad h_\sigma(\bar{x},0) = 0. \tag{4.8}$$

[3] Aus Platzgründen wird das Argument der Ableitung $(\bar{x},0)$ nicht mit aufgeführt.

Bestimmung quadratischer Koeffizienten

Analog zur Vorgehensweise bei den linearen Koeffizienten ergeben sich die quadratischen Koeffizienten, indem man die Nullfunktion F zweimal partiell differenziert und sie im Equilibrium $(\bar{x}, 0)$ auswertet.
Bei zweimaliger Ableitung nach der Zustandsvariablen x stößt man auf folgendes Gleichungssystem, in welchem die Unbekannten $[g_{xx}]$ sowie $[h_{xx}]$ vorkommen. Die linearen Koeffizienten sind dabei durch das vorherige Lösen des Gleichungssystems (4.6) bekannt.

$$
\begin{aligned}
[F_{xx}(\bar{x},0)]^i_{jk} = &([f_{u\prime u\prime}]^i_{ac}[g_x]^c_d[h_x]^d_k + [f_{u\prime u}]^i_{ac}[g_x]^c_k \\
&+ [f_{u\prime x\prime}]^i_{ad}[h_x]^d_k + [f_{u\prime x}]^i_{ak})[g_x]^a_b[h_x]^b_j \\
&+ [f_{u\prime}]^i_a[g_{xx}]^a_{bd}[h_x]^d_k[h_x]^b_j \\
&+ [f_{u\prime}]^i_a[g_x]^a_b[h_{xx}]^b_{jk} \\
&+ \left([f_{uu\prime}]^i_{ac}[g_x]^c_d[h_x]^d_k + [f_{uu}]^i_{ac}[g_x]^c_k + [f_{ux\prime}]^i_{ad}[h_x]^d_k + [f_{ux}]^i_{ak}\right)[g_x]^a_j \\
&+ [f_u]^i_a[g_{xx}]^a_{jk} \\
&+ \left([f_{x\prime u\prime}]^i_{bc}[g_x]^c_d[h_x]^d_k + [f_{x\prime u}]^i_{bc}[g_x]^c_k + [f_{x\prime x\prime}]^i_{bd}[h_x]^d_k + [f_{x\prime x}]^i_{bk}\right)[h_x]^b_j \\
&+ [f_{x\prime}]^i_b[h_{xx}]^b_{jk} \\
&+ [f_{xu\prime}]^i_{jc}[g_x]^c_d[h_x]^d_k + [f_{xu}]^i_{jc}[g_x]^c_k + [f_{xx\prime}]^i_{jd}[h_x]^d_k + [f_{xx}]^i_{jk} \\
&= 0 \\
&i = 1,\dots,n+d, \quad j,k,b,d = 1,\dots,n; \quad a,c = 1,\dots,d.
\end{aligned}
\tag{4.9}
$$

Bemerkung 4.6. *Es handelt sich hierbei um $(n+d)\cdot n^2$ lineare Gleichungen in den $(n+d)\cdot n^2$ unbekannten Koeffizienten $[g_{xx}]$, $[h_{xx}]$.*

Dieses sehr große linear homogene Gleichungssystem zur Bestimmung von zwei der quadratischen Koeffizienten besitzt eine eindeutige Lösung.
Detailliertere Informationen zur Existenz und Eindeutigkeit der Lösung der Gleichungssysteme können [2] entnommen werden.
Zur Berechnung der noch fehlenden Koeffizienten $[g_{\sigma\sigma}]$ und $[h_{\sigma\sigma}]$ bildet man die partielle Ableitung $F_{\sigma\sigma}(\bar{x}, 0)$ und setzt den Ausdruck, der sich durch Anwendung der mehrdimensionalen Kettenregel ergibt, gleich 0.

$$
\begin{aligned}
[F_{\sigma\sigma}(\bar{x},0)]^i =& [f_{u\prime}]^i_a [g_x]^a_b [h_{\sigma\sigma}]^b \\
&+ [f_{u\prime u\prime}]^i_{ac} [g_x]^c_d [\eta]^d_e [g_x]^a_b [\eta]^b_s [I]^s_e \\
&+ [f_{u\prime x\prime}]^i_{ad} [\eta]^d_e [g_x]^a_b [\eta]^b_s [I]^s_e \\
&+ [f_{u\prime}]^i_a [g_{xx}]^a_{bd} [\eta]^d_e [\eta]^b_s [I]^s_e \\
&+ [f_{u\prime}]^i_a [g_{\sigma\sigma}]^a \\
&+ [f_u]^i_a [g_{\sigma\sigma}]^a \\
&+ [f_{x\prime}]^i_b [h_{\sigma\sigma}]^b \\
&+ [f_{x\prime u\prime}]^i_{bc} [g_x]^c_d [\eta]^d_e [\eta]^b_s [I]^s_e \\
&+ [f_{x\prime x\prime}]^i_{bd} [\eta]^d_e [\eta]^b_s [I]^s_e \\
&= 0; \\
& i = 1, \ldots, n+d; \quad a, c = 1, \ldots, d; \quad b, d = 1, \ldots, n; \quad s, e = 1, \ldots, n_2.
\end{aligned}
\tag{4.10}
$$

Bemerkung 4.7. *Hier ergeben sich $n+d$ lineare Gleichungen mit $n+d$ Unbekannten, die den gesuchten Größen $[g_{\sigma\sigma}]$, $[h_{\sigma\sigma}]$ entsprechen.*

Um zuletzt die Koeffizienten $[g_{\sigma x}]$ und $[h_{\sigma x}]$ zu benennen, bildet man $[F_{x\sigma}(\bar{x},0)]$ und erhält folgendes System von Gleichungen.

$$
\begin{aligned}
[F_{\sigma x}(\bar{x},0)]^i_j =& [f_{u\prime}]^i_a [g_x]^a_b [h_{\sigma x}]^b_j + [f_{u\prime}]^i_a [g_{\sigma x}]^a_c [h_x]^c_j \\
&+ [f_u]^i_a [g_{\sigma x}]^a_j + [f_{x\prime}]^i_b [h_{\sigma x}]^b_j \\
&= 0; \\
& i = 1, \ldots, n+d; \quad a = 1, \ldots, d; \quad b, c, j = 1, \ldots, n.
\end{aligned}
\tag{4.11}
$$

Bemerkung 4.8. *In diesem Fall entstehen $(n+d) \cdot n$ Gleichungen mit ebenso vielen Unbekannten. Offensichtlich ist das System homogen in den Unbekannten $[g_{\sigma x}]$ und $[h_{\sigma x}]$. Aus Gründen der Eindeutigkeit der Lösung folgt:*

$$g_{\sigma x}(\bar{x},0) = 0 \tag{4.12}$$

$$h_{\sigma x}(\bar{x},0) = 0. \tag{4.13}$$

Die Erkenntnisse, die nach Analyse obiger Gleichungssysteme gewonnen wurden, fasst folgender Satz zusammen. Er drückt aus, dass für die quadratische Approximation der Lösungen des Modells neben dem bekannten Gleichgewicht nur noch ein Teil der linearen bzw. quadratischen Koeffizienten in die Näherungslösung eingehen.

Satz 4.9. *Gegeben sei ein dynamisches Gleichgewichtsmodell der Form* (4.1) *und dessen Lösungsfunktionen g und h gemäß (4.2). Dann gilt:*

$$\begin{aligned} g_\sigma(\bar{x}, 0) &= 0, \\ h_\sigma(\bar{x}, 0) &= 0, \\ g_{x\sigma}(\bar{x}, 0) &= 0, \quad \textit{und} \\ h_{x\sigma}(\bar{x}, 0) &= 0. \end{aligned}$$

Korollar 4.10. *Die quadratischen Approximationen für die Kontrollfunktion g bzw. Dynamik h des Perturbationsmodells (4.2) vereinfachen sich zu:*

$$\begin{aligned} [g(x,\sigma)]^i =& [g(\bar{x},0)]^i + [\mathbf{g_x}(\bar{\mathbf{x}},\mathbf{0})]^{\mathbf{i}}_{\mathbf{a}} \cdot (x-\bar{x})_a \\ &+ \frac{1}{2} \cdot [\mathbf{g_{xx}}(\bar{\mathbf{x}},\mathbf{0})]^{\mathbf{i}}_{\mathbf{ab}} \cdot (x-\bar{x})_a \cdot (x-\bar{x})_b \\ &+ \frac{1}{2} \cdot [\mathbf{g}_{\sigma\sigma}(\bar{\mathbf{x}},\mathbf{0})]^{\mathbf{i}} \cdot (\sigma) \cdot (\sigma) \\ &\textit{mit } i = 1,\ldots,d \quad a,b = 1,\ldots,n. \end{aligned}$$

bzw.

$$\begin{aligned} [h(x,\sigma)]^j =& [h(\bar{x},0)]^j + [\mathbf{h_x}(\bar{\mathbf{x}},\mathbf{0})]^{\mathbf{i}}_{\mathbf{a}} \cdot (x-\bar{x})_a \\ &+ \frac{1}{2} \cdot [\mathbf{h_{xx}}(\bar{\mathbf{x}},\mathbf{0})]^{\mathbf{j}}_{\mathbf{ab}} \cdot (x-\bar{x})_a \cdot (x-\bar{x})_b \\ &+ \frac{1}{2} \cdot [\mathbf{h}_{\sigma\sigma}(\bar{\mathbf{x}},\mathbf{0})]^{\mathbf{j}} \cdot (\sigma) \cdot (\sigma) \\ &\textit{mit } j = 1,\ldots,n \quad a,b = 1,\ldots,n. \end{aligned}$$

Wie die mit Fettschrift hervorgehobenen Koeffizienten unter Einsatz des Computers berechnet werden, schildert Abschnitt (4.3) über die Perturbationsroutinen. Mit Hilfe ausgereifter, mathematischer Software ist es möglich, die großen Gleichungssysteme, in denen kennzeichnend eine Vielzahl an partiellen Ableitungen auftreten, mit überschaubarem Rechenaufwand zu lösen und somit die Taylorreihen aufzustellen.

Es ist erwähnenswert, dass die quadratischen Approximationen den stochastischen Einfluss des Modells lediglich durch die beiden konstanten Terme $\frac{1}{2}[h_{\sigma\sigma}(\bar{x},0)] \cdot \sigma^2$ bzw. $\frac{1}{2}[g_{\sigma\sigma}(\bar{x},0)] \cdot \sigma^2$ wiedergeben. Würde man sich mit einer Taylorreihe erster Ordnung begnügen, so fiele der stochastische Anteil gänzlich weg und kein Verteilungsparameter, z.B. $\sigma^2 = Var(\epsilon)$, würde die Lösung beeinflussen. Auf einen Punkt gebracht halten dies COLLARD & JUILLARD in ihrem Artikel [5] wie folgt fest:

> „Indeed, one of the merits of higher orders perturbation methods is their ability to exploit the information contained in the higher order moments of distribution of the shocks that hit the economy.“

Höhere Ordnungen der Approximationen als die hier aufgezeigte quadratische sind durchaus möglich, führen allerdings zu einer weiteren Anzahl an linearen Gleichungssystemen, die mit Hilfe von Rechnern jedoch einfach und effizient zu lösen sind. So findet sich in eben erwähnter Arbeit [5] beispielsweise eine Approximation vierter Ordnung für ein spezielles finanzmathematisches Anwendungsbeispiel.

4.2.2 Indirekte Taylor-Approximation der optimalen Wertefunktion V_∞

Die optimale Wertefunktion V_∞ , die in (3.17) definiert wurde, wird ebenfalls durch ein Taylorpolynom angenähert.
Die indirekte Approximation wird fortan mit $\widehat{V}$ bezeichnet und erhält die Form einer Taylorreihe:

$$\begin{aligned}\widehat{V}(x,\sigma) = {} & V_0(\bar{x},0) + [V_x(\bar{x},0)]_a[(x-\bar{x})]_a + V_\sigma(\bar{x},0)\sigma \\ & + \frac{1}{2}[V_{xx}(\bar{x},0)]_{ab}[(x-\bar{x})]_a[(x-\bar{x})]_b + \frac{1}{2}[V_{x\sigma}(\bar{x},0)]_a[(x-\bar{x})]_a \cdot \sigma \\ & + \frac{1}{2}[V_{\sigma x}(\bar{x},0)]_a\sigma[(x-\bar{x})]_a + \frac{1}{2}[V_{\sigma\sigma}(\bar{x},0)] \cdot \sigma^2\end{aligned}$$

Im Unterschied zu Kontrolle und Dynamik des Perturbationsmodells, bei denen die Funktion F der Gleichgewichtsbedingungen und ihre Ableitungen als Ausgangspunkt zur Bestimmung der Taylor–Koeffizienten diente, ist dies für $\widehat{V}$ die Bellman–Gleichung.

Die Bellman–Gleichung für die optimale Wertefunktion lautet:

$$V_\infty(x) = \mathbb{E}\left\{l(x, F_\infty(x)) + \beta V_\infty(\varphi(x, F_\infty(x), z))\right\}$$

Es werden die gefundenen Perturbations–Approximationen g und h sowie die Approximation $\hat{V}$ in die Gleichung eingesetzt

$$\hat{V}(x,\sigma) = \mathbb{E}\left\{l(x, g(x,\sigma)) + \beta\hat{V}(h(x,\sigma) + \eta\sigma\cdot\epsilon', \sigma)\right\}.$$

Wir definieren für die rechte Seite die Funktion

$$R(x,\sigma) := l(x, g(x,\sigma)) + \beta\hat{V}(h(x,\sigma) + \eta\sigma\cdot\epsilon', \sigma)$$

Damit gilt:

$$\hat{V}(x,\sigma) = \mathbb{E}\left\{R(x,\sigma)\right\}.$$

Werden schließlich beide Seiten dieser zentralen Gleichung quadratisch in den Argumenten (x, σ) entwickelt, so ergibt sich:

$$\begin{aligned} & V_0(\bar{x},0) + [V_x(\bar{x},0)]_a[(x-\bar{x})]_a + [V_\sigma(\bar{x},0)]\sigma \\ & \quad + \frac{1}{2}[V_{xx}(\bar{x},0)]_{ab}[(x-\bar{x})]_a[(x-\bar{x})]_b + \frac{1}{2}[V_{x\sigma}(\bar{x},0)]_a[(x-\bar{x})]_a \cdot \sigma \\ & \quad + \frac{1}{2}[V_{\sigma x}(\bar{x},0)]_a\sigma[(x-\bar{x})]_a + \frac{1}{2}V_{\sigma\sigma}(\bar{x},0)\cdot\sigma^2 \\ & \quad = \\ \mathbb{E}\{ & R_0(\bar{x},0) + [R_x(\bar{x},0)]_a[(x-\bar{x})]_a + [R_\sigma(\bar{x},0)]\sigma\epsilon \\ & \quad + \frac{1}{2}[R_{xx}(\bar{x},0)]_{ab}[(x-\bar{x})]_a[(x-\bar{x})]_b + \frac{1}{2}[R_{x\sigma}(\bar{x},0)]_a[(x-\bar{x})]_a \cdot \sigma\epsilon \\ & \quad + \frac{1}{2}[R_{\sigma x}(\bar{x},0)]_a\sigma\epsilon[(x-\bar{x})]_a + \frac{1}{2}R_{\sigma\sigma}(\bar{x},0)\cdot\sigma^2\epsilon^2\}, \\ & \quad \text{mit } a,b = 1,\ldots,n. \end{aligned} \tag{4.14}$$

Die Auswertung der Ableitungen erfolgt gleichsam im Gleichgewichtspunkt $(\bar{x}, 0)$. Der Ausdruck der rechten Seite verkürzt sich dank der im Perturbationsmodell erfolgten Verteilungsannahme $\epsilon \sim N(0, I)$ für den stochastischen Prozess. Somit gilt:

$$\begin{aligned} \mathbb{E}(\epsilon) = \mathbb{E}(\epsilon x) = \mathbb{E}(x\epsilon) = 0 \\ \mathbb{E}(\epsilon^2)(= Var(\epsilon)) = 1. \end{aligned}$$

Die Linearitätseigenschaft des Erwartungswertes $\mathbb{E}$ und obige Folgerungen lassen die folgenden Terme der rechten Seite verschwinden:

$$\mathbb{E}(R_\sigma(\bar{x},0)\sigma\epsilon) = 0 \tag{4.15}$$

$$\mathbb{E}([R_{x\sigma}(\bar{x},0)]_a[(x-\bar{x})]_a \cdot \sigma\epsilon) = 0 \tag{4.16}$$

$$\mathbb{E}([R_{\sigma x}(\bar{x},0)]_a\sigma\epsilon[(x-\bar{x})]_a) = 0 \tag{4.17}$$

$$a = 1,\ldots,n \tag{4.18}$$

Weiterhin gilt $\mathbb{E}\{\frac{1}{2}R_{\sigma\sigma}(\bar{x},0)\cdot\sigma^2\epsilon^2\} = \frac{1}{2}R_{\sigma\sigma}(\bar{x},0)\cdot\sigma^2$.
Die damit verbliebene Taylor–Entwicklung von (4.14) führt durch entsprechendes Vergleichen der Terme der linken und rechten Seite auf die nachstehenden linearen Gleichungen $(a, b = 1,\ldots,n)$:

$$\begin{aligned} & V_0(\bar{x},0) = R_0(\bar{x},0) \\ & [V_x(\bar{x},0)]_a = [R_x(\bar{x},0)]_a \\ & [V_{xx}(\bar{x},0)]_{ab} = [R_{xx}(\bar{x},0)]_{ab} \\ & V_{\sigma\sigma}(\bar{x},0) = R_{\sigma\sigma}(\bar{x},0) \end{aligned} \tag{4.19}$$

Bemerkung 4.11. *Wir nehmen an, dass die Kontrollfunktion g eindimensional, die Dynamik h aus dem Perturbationsmodell zweidimensional ist. Dies wird auch in den späteren Anwendungen der Fall sein.*

Die Funktion $R(x, \sigma)$, die zur kompakten Schreibweise der rechten Seite eingeführt wurde, wird nach Summen- und Kettenregel nach x_a, $x_a x_b$ $(a, b = 1, 2)$ sowie $\sigma\sigma$ abgeleitet.

So ergibt sich ein lineares Gleichungssystem der Form

$$M \cdot x = b, \text{mit } x := (V_0, V_{x1}, V_{x2}, V_{x_1x_1}, V_{x_1x_2}, V_{x_2x_2}, V_{\sigma\sigma})^T. \tag{4.20}$$

$M \in \mathbb{R}^{7\times 7}$ ist eine quadratische Koeffizientenmatrix mit den Einträgen:

$$\begin{pmatrix}
1-\beta & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1-\beta\hat{x1}_{x1} & -\beta\hat{x2}_{x1} & 0 & 0 & 0 & 0 \\
0 & -\beta\hat{x1}_{x2} & 1-\beta\hat{x2}_{x_2} & 0 & 0 & 0 & 0 \\
0 & -\beta\hat{x1}_{x1x1} & -\beta\hat{x2}_{x1x1} & 1-\beta(\hat{x1}_{x1})^2 & -2\beta\hat{x1}_{x1}\hat{x2}_{x1} & -\beta(\hat{x2}_{x1})^2 & 0 \\
0 & -\beta\hat{x1}_{x1x2} & -\beta\hat{x2}_{x1x2} & -\beta\hat{x1}_{x1}\hat{x1}_{x2} & 1-\beta(\hat{x1}_{x1}\hat{x2}_{x2}+\hat{x2}_{x1}\hat{x1}_{x2}) & -\beta\hat{x2}_{x1}\hat{x2}_{x2} & 0 \\
0 & -\beta\hat{x1}_{x2x2} & -\beta\hat{x2}_{x2x2} & -\beta(\hat{x1}_{x2})^2 & -2\beta\hat{x1}_{x2}\hat{x2}_{x2} & 1-\beta(\hat{x2}_{x2})^2 & 0 \\
0 & -\beta\hat{x1}_{\sigma\sigma} & -\beta\hat{x2}_{\sigma\sigma} & -\beta(\hat{x1}_{\sigma})^2 & -2\beta\hat{x1}_{\sigma}\hat{x2}_{\sigma} & -\beta(\hat{x2}_{\sigma})^2 & 1
\end{pmatrix}$$

Bemerkung 4.12. *Hierbei bezeichnet die Funktion $\hat{x1}$ die erste Komponente der Dynamik h aus dem Modell (4.2), entsprechend notiert $\hat{x2} := h(x, \sigma)^2$ deren zweite.*

$b \in \mathbb{R}^7$ ist der Vektor der rechten Seite, dessen Einträge primär den Ableitungen der Ertragsfunktion $l(x, g(x, \sigma))$ entstammen, nachdem g eingesetzt wurde.

$$b := \begin{pmatrix}
l(\bar{x}, \bar{u}) \\
l_u \cdot g_{x1} \\
l_u \cdot g_{x2} \\
l_{uu}(g_{x1})^2 + l_u g_{x_1x_1} \\
l_{uu} g_{x1} g_{x2} + l_u g_{x_1x_2} \\
l_{uu}(g_{x2})^2 + l_u g_{x_2x_2} \\
l_{uu}(g_{\sigma})^2 + l_u g_{\sigma\sigma}
\end{pmatrix} \tag{4.21}$$

Ist man an einer rein linearen Approximation für $\widehat{V}$ interessiert, so ist das kleinere Gleichungssystem

$$M_{lin} \cdot x = b_{lin} \tag{4.22}$$

zu lösen, wobei

$$M_{lin} := (M_{ij}); i, j \in \{1, 2, 3\}$$

eine (3×3)–Teilmatrix von M und

$$b_{lin} := (b_1, b_2, b_3)^T$$

ein dreidimensionaler Teilvektor von b ist.
Als Lösung ergibt sich $x = (V_0, V_{x1}, V_{x2})^T$, der Vektor der Taylorkoeffizienten.

Jedoch ist diese in den Wirtschaftswissenschaften gebräuchlichste Form der Approximation, die sogenannte Standardlinearisierung, von wesentlich geringerer Approximationsgüte als die quadratische. Ferner wird der stochastische Einfluss bzw. die Volatilität der stochastischen Schocks, die die exogenen Zustandsvariablen betreffen, vernachlässigt, da σ nicht in die Berechnung eingeht. Die Lösung der Gleichungssysteme (4.22) und (4.20) erfolgt mittels eines in MATLAB implementierten Programms, dessen Funktionsweise in Teilabschnitt (4.3) beschrieben wird.

4.2.3 Direkte Taylor-Approximation der optimalen Wertefunktion V_∞

Dieser Teil befasst sich ebenfalls mit der Approximation der optimalen Wertefunktion V_∞, jedoch findet diesmal deren Bestimmung auf eine Art statt, die sich als „direkt“ bezeichnen lässt. Im vorherigen Abschnitt stellte die Bellman–Gleichung, die auf beiden Seiten quadratisch entwickelt wurde, den Ansatzpunkt für die Gewinnung einer Approximation für V_∞ dar. Dies hat auch für die direkte Methode Bestand. Jedoch wurden ferner die aus dem Perturbationsmodell (4.2) stammende Taylor–Näherung des optimalen Feedbacks g sowie die Dynamik h benötigt. Bei der direkten Taylor–Approximation vermeidet man die Verwendung der beiden Approximationen vollständig.

Das Bellman-Prinzip verkörpert die Gleichung, deren eindeutige Lösung die optimale Wertefunktion ist. Ohne Kenntnis der exakten bzw. einer approximierenden Kontrollfunktion lautet sie

$$V_\infty(x) = \max_{u \in U} \mathbb{E}\left\{ l(x,u) + \beta V_\infty(\varphi(x,u,\sigma,\epsilon)) \right\}. \tag{4.23}$$

Die Zufallsvariable z wurde in der Dynamik φ durch $\sigma \cdot \epsilon$ ($\epsilon \sim N(0,1)$) ersetzt, welches identisch verteilt ist (siehe Bemerkung (3.15)).
Da nicht bekannt ist, wie die optimale Kontrolle u in Abhängigkeit des Zustands x zu wählen ist, erfolgt auf der rechten Seite von (4.23) die Maximums-Bildung über alle zulässigen u.
Der Ansatz bei der direkten Methode besteht darin, die linke Seite quadratisch nach den Variablen x und σ zu entwickeln, die rechte Seite als Term in den Variablen x, u und σ zu betrachten und im Entwicklungspunkt $(\bar{x}, \bar{u}, 0)$ ebenfalls in zweiter Ordnung nach Taylor zu entwickeln. Die Berechnung von $(\bar{x}, \bar{u})$ geschieht über die Gleichgewichtsbedingungen, die für das durch die Setzung $\sigma = 0$ deterministisch gewordene Kontrollsystem aufgestellt werden.

Die linke Seite kann dann wie im vorangehenden Abschnitt in Form einer Taylorreihe zweiter Ordnung, die die Bezeichnung $\hat{V}_{dir}$ erhält, geschrieben werden [4]:

$$\begin{aligned} \hat{V}_{dir}(x,\sigma) &= V_0(\bar{x},0) + [V_x(\bar{x},0)]_a[(x-\bar{x})]_a \\ &\quad + \frac{1}{2}[V_{xx}(\bar{x},0)]_{ab}[(x-\bar{x})]_a[(x-\bar{x})]_b + \frac{1}{2}[V_{\sigma\sigma}(\bar{x},0)] \cdot \sigma^2, \quad a,b = 1,2. \end{aligned} \tag{4.24}$$

[4] Die Nullkoeffizienten (Verteilungsannahme ϵ) werden gleich ausgelassen (vgl. (4.15)ff.).

Den zu maximierenden Ausdruck auf der rechten Seite fassen wir als Funktion

$$G : \mathbb{R}^2 \times \mathbb{R} \times [0; \infty[\to \mathbb{R}, (x, u, \sigma) \mapsto G(x, u, \sigma)$$

mit den Argumenten x, u und σ.

$$G(x, u, \sigma) := l(x, u) + \beta \hat{V}_{dir}(\varphi(x, u, \sigma, \epsilon), \sigma). \tag{4.25}$$

So lässt sich (4.23) ingesamt schreiben als

$$\hat{V}_{dir}(x, \sigma) = \max_{u \in U} \mathbb{E}\{G(x, u, \sigma)\}. \tag{4.26}$$

Nun wird auch die mehrdimensionale Funktion G gemäß dem Satz von Taylor in zweiter Ordnung entwickelt.

Dann erhält man folgende Taylorreihe $\hat{G}_{\hat{y},2}$ zweiter Ordnung als Argument des Maximums:

$$\hat{G}_{\hat{y},2}(z) := \sum_{|j| \leq 2} \frac{D^j G(\hat{y})}{j!} (y - \hat{y})^j \tag{4.27}$$

wobei die Variable $y \in \mathbb{R}^4$ wie folgt definiert ist:

$$y = (x_1, x_2, u, \sigma), \ \hat{y} = (\bar{x}_1, \bar{x}_2, \bar{u}, 0). \tag{4.28}$$

Hierbei notiert j einen Multiindex mit den folgenden Schreibweisen:

$$j = \begin{pmatrix} j_1 \\ \vdots \\ j_4 \end{pmatrix} \in \mathbb{N}_0^4, \quad |j| := j_1 + \cdots + j_4, \quad j! := \prod_{i=1}^{4} j_i!$$

$$D^j G := \frac{\partial^{|j|} G}{\partial y_1^{j_1} \cdot \ldots \cdot \partial y_4^{j_4}}, \quad (y - \hat{y})^j := (y_1 - \hat{y}_1)^{j_1} \cdot \ldots \cdot (y_4 - \hat{y}_4)^{j_4}$$

Die Taylorreihe $\hat{G}_{\hat{y},2}$ gemäß (4.27) enthält die Variable u in mehreren Termen, z.B.

$$\frac{1}{2} \cdot \frac{\partial^2 G}{\partial u \partial u}(\bar{x}, \bar{u}, 0) \cdot (u - \bar{u})^2.$$

Sortiert man den ganzen Ausdruck nach Termen in den Potenzen $u^k, k = 0, 1, 2$, so ist die rechte Seite als eine quadratische Funktion $f(u)$ im Argument u aufzufassen:

$$f(u) := a_0 + a_1 \cdot u + a_2 \cdot u^2. \tag{4.29}$$

Zu beachten ist hierbei, dass deren Koeffizienten a_0, a_1, a_2 neben den Variablen x_1, x_2, σ auch die unbekannten Taylor-Koeffizienten $V_0, V_{x_1}, V_{x_2}, V_{x_1 x_1}, V_{x_1 x_2}, V_{x_2 x_2}, V_{\sigma\sigma}$ von $\hat{V}_{dir}$ beinhalten.

Die Funktion f wird nun nach der Variablen u differenziert und anschließend das mit u_{max} bezeichnete Extremum bestimmt, welches den Scheitelpunkt der quadratischen Funktion festlegt.
Da die Bedingung $f'(u) = 2a_2 \cdot u + a_1 = 0$ lediglich eine notwendige Bedingung für ein Maximum u_{max} darstellt, ist die zweite Ableitung $f''(u_{max})$ auf Negativität , also $2 \cdot a_2 < 0$, zu überprüfen.

Im nächsten Schritt wird in allen Termen, in welchen u auftritt, die allgemeine Kontrolle u durch das eben analytisch ermittelte $u_{max}\,(x_1, x_2, \sigma, V_0, V_{x_1}, \ldots, V_{\sigma\sigma})$ ersetzt.
Damit tritt die Steuerung u nicht mehr als Variable in der rechten Seite auf. Somit lassen sich beide Seiten der Gleichung, also $\hat{V}_{dir}$ und $\hat{G}_{\hat{y},2}$, in welche u_{max} eingesetzt wurde, nach Termen der Form

$$x_1^i \cdot x_2^j \cdot \sigma^k,\ i,j,k \in \{0,1,2\},\ i+j+k \leq 2,\ k \neq 1$$

sortieren. Diese Terme nennt man Monome in den Unbekannten x_1, x_2 und σ. Die linke Seite beispielsweise schreibt sich entsprechend umgestellt wie folgt:

$$\begin{aligned}\hat{V}_{dir} = &\frac{1}{2}V_{x_1x_1} \cdot x_1^2 + V_{x_1x_2} \cdot x_1x_2 + (V1 - V_{x_1x_2}\bar{x}_2 - V_{x_1x_1}\bar{x}_1) \cdot x_1 \\ &+ \frac{1}{2}V_{x_2x_2} \cdot x_2^2 + (V_{x_2} - V_{x_2x_2}\bar{x}_2 - V_{x_1x_2}\bar{x}_1) \cdot x_2 + \frac{1}{2}V_{\sigma\sigma}\sigma^2 \\ &+ (V_0 - V_{x_1}\bar{x}_1 - V_{x_2}\bar{x}_2 + \frac{1}{2}V_{x_1x_1}\bar{x}_1^2 + \frac{1}{2}V_{x_2x_2}\bar{x}_2^2 + V_{x_1x_2}\bar{x}_1\bar{x}_2).\end{aligned}$$

Somit kann man einen Vergleich der Koeffizienten der Monome $x_1^i \cdot x_2^j \cdot \sigma^k$ der linken und rechten Seite durchführen. Die Koeffizienten der linken Seite bezeichnen wir mit $lc_i, i = 1, \ldots, 7$, mit rc_i diejenigen der rechten Seite.
Die Koeffizienten von lc ergeben sich gemäß Tabelle (4.1). Für die rechte Seite werden aufgrund der Ersetzung von u durch u_{max} die Koeffizientenausdrücke wesentlich unübersichtlicher, aber sie sind ermittelbar.

$\mathbf{lc_i}$:	**zugehörige Monome:**
$\frac{1}{2}V_{x_1x_1}$	x_1^2
$\frac{1}{2}V_{x_2x_2}$	x_2^2
$V_{x_1x_2}$	$x_1 \cdot x_2$
$V1 - V_{x_1x_2}\bar{x}_2 - V_{x_1x_1}\bar{x}_1$	x_1
$V_{x_2} - V_{x_2x_2}\bar{x}_2 - V_{x_1x_2}\bar{x}_1$	x_2
$\frac{1}{2}V_{\sigma\sigma}$	σ^2
$V_0 - V_{x_1}\bar{x}_1 - V_{x_2}\bar{x}_2 + \frac{1}{2}V_{x_1x_1}\bar{x}_1^2 + \frac{1}{2}V_{x_2x_2}\bar{x}_2^2 + V_{x_1x_2}\bar{x}_1\bar{x}_2$	$1\ (= x^0y^0\sigma^0)$

Tabelle 4.1: Koeffizienten der Monome in $\hat{V}_{dir}$

Durch Gleichsetzen der Koeffizienten von linker und rechter Seite

$$lc_i = rc_i, \quad \forall i = 1, 2, \ldots, 7$$

erhält man ein Gleichungssystem mit 7 Gleichungen in den gesuchten Unbekannten $V_0, V_{x_1}, V_{x_2}, V_{x_1x_1}$ $V_{x_1x_2}, V_{x_2x_2}, V_{\sigma\sigma}$, das allerdings nichtlinear ist.
Definiert man

$$e_i := lc_i - rc_i, \quad \forall i = 1, 2, \ldots, 7 \tag{4.30}$$

und fasst

$$e : \mathbb{R}^7 \to \mathbb{R}^7, \quad x \mapsto \begin{pmatrix} e_1(x) \\ \vdots \\ e_7(x) \end{pmatrix}$$

als mehrdimensionale Funktion auf, dann ist das Problem der Bestimmung der Koeffizienten von $\hat{V}_{dir}$ äquivalent zur Lösung des nichtlinearen Nullstellenproblem

$$e(x) = 0_{\mathbb{R}^7} \tag{4.31}$$

mit $x := (V_0, V_{x_1}, V_{x_2}, V_{x_1x_1}, V_{x_1x_2}, V_{x_2x_2}, V_{\sigma\sigma})^T$. Als numerisches Verfahren zur Lösung dieses nichtlinearen Gleichungssystems findet das **multivariate Newton-Verfahren** Verwendung.
In diesem wird ausgehend von einer Startschätzung x_0 für die Taylor-Koeffizienten die Iteration gemäß

$$x_{n+1} = N_e(x_n) := x_n - \left(\frac{\partial e}{\partial x}(x_n)\right)^{-1} \cdot e(x_n), \quad n = 0, 1, \ldots \tag{4.32}$$

solange durchgeführt, bis sich die Iterierten hinsichtlich einer geeigneten Norm nur um weniger als ein vorgegebenes $\epsilon > 0$ unterscheiden, d.h. $\| x_{n+1} - x_n \| < \epsilon$.

Bemerkung 4.13. *Hierbei bezeichnet* $\left(\frac{\partial e}{\partial x}(x_n)\right)$ *die Jacobi-Matrix von* e *in der aktuellen Iterierten* x_n.

Mit dem mathematischen Softwaresystem MAPLE wird (4.31) dann gelöst. Der zur Lösung des Nullstellenproblems verwendete Befehl lautet `fsolve`.

Der große Vorteil der direkten Methode, die in einer MAPLE Routine implementiert ist [5], besteht wie anfangs erwähnt darin, dass man auf die beiden Approximationen $g(x, \sigma)$ und $h(x, \sigma)$ komplett verzichtet. Der Rechenaufwand reduziert sich somit durch das Wegfallen der Bestimmung der linearen und quadratischen Koeffizienten von g und h erheblich.

[5] Die Programmbeschreibung findet sich ebenso in Abschnitt (4.3)

4.3 Perturbationsroutinen

Das Auffinden der linearen und quadratischen Koeffizienten in der Taylorentwicklung der Steuerungsfunktion g und der Dynamik h ist wesentlicher Bestandteil der Perturbationsmethode.
Dies erfolgt mit Unterstützung der mathematischen Software MATLAB. Die im Internet frei erhältlichen Dateien nach [23] werden in diesem letzten Teilabschnitt vorgestellt. Ausführlicher sind sie im vierten Kapitel der Arbeit [2] erläutert, auf die verwiesen wird.
Zudem werden die Routinen zur Durchführung der indirekten bzw. direkten Taylor-Approximation von V_∞ vorgestellt.

Voraussetzung: Symbolic Math Toolbox

Zur erfolgreichen Ausführung der Routinen benötigt man zwingend die **Symbolic Math Toolbox** von MATLAB. In dem Ergänzungspaket wird ein neuer MATLAB Datentyp, das „symbolische Objekt“ , definiert. Symbolische Objekte werden genutzt, um symbolische Variablen, Ausdrücke und Matrizen zu definieren. Die Software verwendet für derartige Berechnungen den Befehlskern des Computeralgebrasystems MAPLE.
Im Rahmen der Perturbationsmethoden lassen sich dadurch beispielsweise die partiellen Ableitungen der Funktion f aus der Gleichgewichtsbedingung (4.1) effizient analytisch berechnen.

Die Routinen

Es werden die folgenden Routinen für die numerische Umsetzung des Perturbationsverfahrens eingesetzt:

`anal_deriv.m`

Funktionsweise:
Berechnung der ersten und optional zweiten partiellen Ableitungen der Gleichgewichtsfunktion f von (4.1)

Eingabeparameter:
Funktion f, die Variablen x, u, x', u' sowie der Parameter $approx$, der die Approximationsordnung vorgibt

---

Rückgabe:
Die partiellen Ableitungen $f_x, f_{x'}, f_u, f_{u'}$
sowie optional
$f_{u'u'}, f_{u'u}, f_{u'x'}, f_{u'x}, f_{uu'}, f_{uu}, f_{ux}, f_{x',u'}, f_{x'u}, f_{x'x'}, f_{x'x}, f_{xu'}, f_{xu}, f_{ux}, f_{xx'}, f_{xx}$

`num_eval.m`

Funktionsweise:
Auswertung der in `anal_deriv.m` gewonnenen analytischen Ableitungen im Gleichgewichtspunkt $(\bar{x}, \bar{u})$

Eingabeparameter:
keine

Rückgabe:
Die im Gleichgewicht ausgewerteten Ableitungen $f^n_x, f^n_{x'}, f^n_u, f^n_{u'}, f^n_{u'u'}, f^n_{u'u}$, $f^n_{u'x'}, f^n_{u'x}, f^n_{uu'}, f^n_{uu}, f^n_{ux}, f^n_{x',u'}, f^n_{x'u}, f^n_{x'x'}, f^n_{x'x}, f^n_{xu'}, f^n_{xu}, f^n_{ux}, f^n_{xx'}, f^n_{xx}$

Bemerkung 4.14. *Dabei bezeichnet f^n die numerische Auswertung der analytischen Ableitung im steady state $(\bar{x}, \bar{u})$.*

Nach Ausführung dieser beiden als elementar zu bezeichnenden Programmteile stehen alle numerischen Werte der Ableitung von f im Workspace von MATLAB zur Verfügung. Nun kann, auf den Resultaten aufbauend, mittels der nachfolgenden Routinen die Berechnung der nötigen Ableitungen der Kontrollfunktion g und der Dynamik h erfolgen, um eine lineare bzw. bei Wunsch nach einer höheren Ordnung quadratische Approximation zu bekommen.

`gxhx.m`

Funktionsweise:
Durchführung der linearen Approximation. Berechnung der Matrizen g_x und h_x über ein verallgemeinertes Eigenwertproblem.
Grundlage hierfür ist das Gleichungssystem (4.6).

Eingabeparameter:
Die ersten Ableitungen $f^n_x, f^n_{x'}, f^n_u, f^n_{u'}$ und der Parameter $stake$, der die obere Schranke für den größtmöglichen Betrag eines Eigenwerts vorgibt. Wird er nicht mitübergeben, so wird $stake$ im Programm gleich 1 gesetzt.

Rückgabe:
Die Jacobi–Matrizen g_x, h_x im Gleichgewichtspunkt $(\bar{x}, \bar{u})$

Bemerkung 4.15. *Im Internet existiert bei [23] eine alternative Routine zur linearen Approximation namens „gx_hx.m“. Sie basiert auf einer generalisierten Schur-Zerlegung der Ma-*

trizen zur Lösung des Eigenwertproblems. Einzelheiten zum Programm finden sich auf der Homepage von Paul Klein [18].

`gxx_hxx.m`

Funktionsweise:
Durchführung der quadratischen Approximation. Berechnung der zweiten Ableitungen g_{xx} und h_{xx} über ein aufgestelltes lineares Gleichungssystem.
Grundlegend hierfür ist das Gleichungssystem (4.9).

Eingabeparameter:
Sämtliche erste und zweite Ableitungen von f:
$f^n_x, f^n_{x'}, f^n_u, f^n_{u'}, f^n_{u'u'}, f^n_{u'u}, f^n_{u'x'}, f^n_{u'x}, f^n_{uu'}, f^n_{uu}, f^n_{ux}, f^n_{x',u'}, f^n_{x'u}, f^n_{x'x'}, f^n_{x'x}, f^n_{xu'}, f^n_{xu}, f^n_{ux}, f^n_{xx'}, f^n_{xx}$
sowie die beiden Matrizen g_x und h_x.

Rückgabe:
Zweite partielle Ableitungen g_{xx} und h_{xx} ausgewertet im Gleichgewichtspunkt $(\bar{x}, \bar{u})$. Rückgabe als dreidimensionales Array.

`gss_hss.m`

Funktionsweise:
Durchführung der quadratischen Approximation. Berechnung der Vektoren g_{ss} sowie h_{ss} als zweite partielle Ableitungen von g bzw. h über ein lineares Gleichungssystem.
Grundlage hierfür bilden die Gleichungen (4.10).

Eingabeparameter:
Sämtliche erste und zweite Ableitungen von f:
$f^n_x, f^n_{x'}, f^n_u, f^n_{u'}, f^n_{u'u'}, f^n_{u'u}, f^n_{u'x'}, f^n_{u'x}, f^n_{uu'}, f^n_{uu}, f^n_{ux}, f^n_{x',u'}, f^n_{x'u}, f^n_{x'x'}, f^n_{x'x}, f^n_{xu'}, f^n_{xu}, f^n_{ux}, f^n_{xx'}, f^n_{xx}$
sowie die beiden Matrizen g_x und h_x. Weiterhin g_{xx} und die Matrix η aus dem Perturbationsmodell.

Rückgabe:
Die beiden Vektoren g_{ss} und h_{ss}.

Bemerkung 4.16. *Der Buchstabe „s“ bezeichnet in obiger Routine den Störungsparameter σ des Perturbationsmodells.*

Bemerkung 4.17. *Die beiden Routinen für die quadratische Approximation* `gxx_hxx.m` *und* `gss_hss.m` *werden, wie an den Eingabeparametern ersichtlich ist, von der Reihenfolge her zuletzt aufgerufen.*

Berechnung der optimalen Wertefunktion V_∞

Die Ermittlung der optimalen Wertefunktion V_∞ ist im Rahmen dieser Arbeit das primäre Ziel bei der Wahl der Perturbationsmethoden als Lösungsmethode. Dies gelang Öhrlein in [19], wo die Bestimmung der Koeffizienten der Taylorreihe $\hat{V}$ mit Hilfe eines in MAPLE geschriebenen Programms namens „compute_Vcoeff.mw“ realisiert wurde.
Da alle anderen Routinen bereits als MATLAB Code vorliegen, wurde darauf aufbauend bzw. ergänzend die Berechnung der Approximation $\hat{V}$ ebenfalls mit dieser Software umgesetzt. Die programmtechnische Verwirklichung der indirekten Taylor-Approximation, insbesondere der Aufbau des Programms, wird nachstehend erläutert.[6]
Von der Erklärung MATLAB–spezifischer Befehle (z.B. syms, jacobian) wird dabei abgesehen, hierfür kann die interne Hilfe herangezogen werden.

`V_coeff.m`

Funktionsweise:
Berechnung der Koeffizienten der approximierenden Taylorreihe $\hat{V}$ durch Lösung eines linearen Gleichungssystems, das vorab aufgestellt wird

---

Eingabeparameter:
Diskontrate β, Abschreibungsrate d (beide modellabhängig) sowie die Ableitungen $g_x, h_x, g_{xx}, h_{xx}, g_{ss}, h_{ss}$, der Approximationsparameter $approx$ und der Parameter $model$ zur Modellauswahl

---

Ablauf:

1. Definition symbolischer Variablen für x_1, x_2, σ und u,
2. Zusammensetzen der Taylorpolynome g sowie $\hat{x1}, \hat{x2}$ der Ordnung $approx$ und jeweilige Bestimmung der Ableitungen im Gleichgewichtspunkt $(\bar{x}, 0)$, welcher aus Vereinfachungsgründen in den Ursprung $(0_{\mathbb{R}^2}, 0)$ verschoben wird,
3. Definition der Ertragsfunktion $l(x, u)$ des Modells und Bilden der partiellen Ableitungen l_u, l_{uu}, etc.,
4. Aufstellen des linearen Gleichungssystems $M \cdot x = b$ nach (4.20) bzw. $M_{lin} \cdot x = b_{lin}$, falls $approx = 1$ übergeben wurde,

[6] Der ausführliche Quellcode befindet sich im Anhang (B.1) dieser Arbeit auf Seite 111.

5. Berechnung der numerischen Lösung x mittels Gauß–Zerlegung,
6. Zusammensetzen des Arrays $Vcoeff = x$
   (Einträge: gesuchte Koeffizienten der Taylorreihe).

Rückgabe:
Zusammengesetzter Vektor $Vcoeff$ mit den gesuchten Koeffizienten $V_0, V_{x_1}, V_{x_2}, V_{x_1x_1}$, $V_{x_1x_2}, V_{x_2x_2}, V_{\sigma\sigma}$ der Taylorreihe $\hat{V}$

Bemerkung 4.18. *Die Routine zur Ermittlung der optimalen Wertefunktion kann erst nach erfolgreichem Ausführen der Programme* `anal_deriv.m, num_eval.m, gxhx.m, gxx_hxx.m` *und* `gss_hss.m` *aufgerufen werden, da sie jeweils auf deren Output angewiesen ist. Anhand des späteren Tests der Programmteile mit Beispielmodellen in Kapitel sechs wird die Aufrufreihenfolge klar.*

Direkte Approximation von V_∞

Für die direkte Taylor–Approximation $\hat{V}_{dir}$ der optimalen Wertefunktion existiert ebenso ein Programm, welches deren Taylor-Koeffizienten errechnet. Diese Routine wurde allerdings mit dem Computer-Algebra-System MAPLE geschrieben, welches bei einer Vielzahl an analytischen Rechnungen zu bevorzugen ist. Das Programm `Vdirect.mw` berechnet gemäß dem im Abschnitt (4.2.3) erläuterten direkten Ansatz die Koeffizienten $V_0, V_{x_1}, V_{x_2}, V_{x_1x_1}, V_{x_1x_2}, V_{x_2x_2}, V_{\sigma\sigma}$ der Taylorreihe $\hat{V}_{dir}$ und speichert diese am Ende in einer ASCII-Textdatei ab.
Nachfolgend werden die wesentlichen Schritte des Programms[7] erklärt:

`Vdirect.mw`

Funktionsweise:
Direkte Taylor–Approximation der Wertefunktion mittels Berechnung der Koeffizienten der Taylorreihe $\hat{V}_{dir}$ durch allgemeine, quadratische Entwicklung beider Seiten des Optimalitätsprinzips

Eingabeparameter:
keine (Modellparameter δ und κ werden zu Beginn per Anweisung festgesetzt)

[7] Ausführlicher Quellcode im Anhang (B.2) auf Seite 129.

Ablauf:

1. Definition der modellspezifischen Parameter, insbesondere δ und κ,
2. Deklaration der Taylorreihe $\hat{V}_{dir}$ als Polynom zweiter Ordnung mit Entwicklungspunkt $(\bar{x}, 0)$,
3. Berechnung des Gleichgewichts $(\bar{x}, \bar{u})$ des Modells per Formel,
4. Definition der Ertragsfunktion $l(x, u)$,
5. Für spezielle Konfiguration $\delta = 1, \kappa = 0$: Aufstellen der exakten, optimalen Wertefunktion V_∞ und anschließende Taylorreihenentwicklung mit dem Befehl ***mtaylor***,
6. Definition der Dynamik φ und der rechten Seite $G(x, u, \sigma)$ des Bellman-Prinzips ,
7. Quadratische Taylorentwicklung von G im Entwicklungspunkt $(\bar{x}_1, \bar{x}_2, \bar{u}, 0) \in \mathbb{R}^4$,
8. Sortierung des gebildeten Taylorpolynoms nach den Termen in u (quadratischer Ausdruck $f(u)$),
9. Ableitung des Ausdrucks f nach u mit ***diff*** und Berechnung des Maximums durch Nullsetzen der 1. Ableitung (***solve***),
10. Einsetzen des ermittelten u_{max} für u in quadratisch entwickelte rechte Seite (***subs***),
11. Entwicklung beider Seiten nach gleichen Potenzen $x_1^i \cdot x_2^j \cdot \sigma^k$ der Variablen x_1, x_2 und σ mittels ***collect*** Anweisung,
12. Abspeichern der Koeffizienten zu den geordneten Potenzen für linke und rechte Seite durch Nutzung des Befehls ***coeffs***,
13. Aufstellen eines 7×7 Gleichungssystems durch Gleichsetzen der zu gleichartigen Termen gehörigen Koeffizienten in den Unbekannten $V_0, V_{x_1}, V_{x_2}, V_{x_1x_1}, V_{x_1x_2}, V_{x_2x_2}, V_{\sigma\sigma}$,
14. Numerisches Lösen des nichtlinearen Gleichungssystems mit ***fsolve***,
15. Schreiben der ermittelten Taylor-Koeffizienten in eine ASCII-Textdatei.

Rückgabe:
ASCII- Textdatei namens `DirectV_kappa_□_delta_●.asc`
mit den gesuchten Taylor-Koeffizienten von $\hat{V}_{dir}$
(□ und ● sind Platzhalter für die Werte der Variablen κ bzw. δ)

Bemerkung 4.19 (**Notationen**). *Einige Variablen haben in der Routine eine andere Bezeichnung als im bisherigen Theorieteil. Hierzu gibt nachfolgende Tabelle einen Überblick.*

Name im Theorieteil	**Programmbezeichnung**
x_1	`x`
x_2	`y`
$\hat{V}_{dir}$	`V`
$\bar{x}_1, \bar{x}_2, \bar{u}$	`xggw, yggw, uggw`
V_∞	`V_ex`
$V_0, V_{x_1}, \ldots, V_{\sigma\sigma}$	`V0,V1...Vss`
φ	`phi`
u_{max}	`umax`
$\hat{V}_{dir}(\varphi(x,u,z))$	`Vh`

Tabelle 4.2: Variablenbezeichnungen in `Vdirect.mw`

Bemerkung 4.20. *Mit der* MAPLE*-Anweisung* `infolevel[fsolve]:=5;` *zu Beginn des Programms kann man näheren Einblick in die interne Arbeitsweise von* ***fsolve*** *gewinnen. Die Zahl* 5 *entspricht dabei der höchsten Informationsstufe.*
Somit lassen sich bei Ausführung auf dem Ausgabebildschirm die Startlösung, die Fehler sowie die Iterierten x_n des multivariaten Newton-Verfahrens nach (4.32) *ablesen.*

5 Implementierung der dynamischen Programmierung

Dieses Kapitel erläutert die Theorie der dynamischen Programmierung, die bei Kontrollproblemen gegeben in der Form (3.16) Anwendung findet, aus der Implementierungssicht. Alle Programme, die das in Abschnitt (3.2) vorgestellte Lösungskonzept verwirklichen, sind in der höheren Programmiersprache C geschrieben und dienen der Gesamtumsetzung des Algorithmus (3.31) über die Werteiteration. Nach thematischen Blöcken geordnet ist dieser Teil in drei Abschitte unterteilt, in denen die jeweiligen Routinen für den Leser erläutert werden.
Als Nachschlagewerk für die eigene Programmierung in C wurden [28] sowie [29] herangezogen.

5.1 Erzeugung normalverteilter Zufallszahlen

Beginnend wird die Frage geklärt, wie man am Rechner randomisierte Zahlen erzeugen kann, die einer Normalverteilung $N(0, \sigma^2)$ genügen sollen. Da der stochastische Input angesichts der Präsenz von z in der Dynamik φ des zeitdiskreten Kontrollsystems (3.12) einen beachtlichen Einfluss für die spätere Analyse besitzt, muss ein zuverlässiger Programmteil existieren, der entsprechend stochastische Einflüsse wiedergeben kann. Im Perturbationsmodell (4.2) wurden stochastische Schocks durch die standardnormalverteilten Zufallsvariablen ϵ_t modelliert.
Für die Simulation von Zufallszahlen am Rechner werden Zufallszahlengeneratoren benutzt. Die Zahlen werden nach einer deterministischen Vorschrift sequentiell erzeugt, weshalb man von „Pseudozufallszahlen“ spricht. Für fast alle Arten von Zufallszahlengeneratoren dienen sogenannte uniform verteilte Zufallszahlen als Ausgangspunkt. Daher führen wir nun ergänzend eine neue stetige Verteilung ein.

Definition 5.1 (Uniformverteilung). *Eine Zufallsvariable X wird auf einem Intervall $[a, b]$, $a, b \in \mathbb{R}$ als gleichverteilt bzw. uniform verteilt bezeichnet, wenn deren Verteilungsdichte p_X gegeben ist als*

$$p_X = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & sonst. \end{cases}$$

In Kurzschreibweise: $X \sim Ufo(a, b)$

Polar–Methode nach Marsaglia

Für die Zwecke des Algorithmus (3.31), in welchem (standard-)normalverteilte Zufallsvariablen als Einflussgröße der Dynamik φ Bedeutung haben, und für die späteren Simulationen der Lösungen wird als Generator die Polar–Methode von Marsaglia[1] herangezogen, die eng im Zusammenhang mit dem Verfahren von Box–Muller steht. Dank ihr lassen sich über geometrische Überlegungen Paare normalverteilter Zufallszahlen gewinnen.
Die Methode nutzt eine Transformation von Verteilungen, wie sie die der folgende Satz beschreibt.

Satz 5.2 (**Transformation von Verteilungen**). *Es sei X eine reelle Zufallsvariable mit bekannter Verteilungsdichte p_X. Eine bijektive Funktion $g : \mathbb{R} \to \mathbb{R}$ bildet X auf eine Zufallsvariable Y ab. Die Umkehrfunktion zu g werde mit $h := g^{-1}$ notiert.*
Dann gilt für die Verteilungsdichte der transformierten Zufallsvariablen $Y = g(X)$:

$$p_Y(y) = p_X(h(y)) \cdot \frac{\partial h}{\partial y}(y), \quad y \in \mathbb{R}$$

Die Transformationsregel ist auch auf mehrdimensionale Zufallsvariablen anwendbar, somit insbesondere auf Zufallsgrößen der Dimension zwei.
Hier gilt dann

$$p_Y = p_X \cdot \det(\frac{\partial h}{\partial y}),$$

wobei $\det(\frac{\partial h}{\partial y})$ die Jacobi–Determinante von h bezeichnet.

Die Polar–Methode von Marsaglia hat als Input zwei uniform verteilte Zufallszahlen
$x_1, x_2 \sim \text{Ufo}(0, 1)$.[2] Damit lassen sich v_1 und v_2 definieren:

$$\begin{aligned} v_1 &= 2 \cdot x_1 - 1 \\ v_2 &= 2 \cdot x_2 - 1 \end{aligned} \tag{5.1}$$

Das Zahlenpaar (v_1, v_2) stellt einen Punkt im Einheitsquadrat des $\mathbb{R}^2$ dar. Gilt für die Summe ihrer Quadrate $rsq := v_1^2 + v_2^2$, auch als quadrierter Abstand vom Ursprung zu deuten, die Eigenschaft $rsq < 1$, so liegt der Punkt (v_1, v_2) im Einheitskreis.
Die folgende polare Transformation g von (x_1, x_2) auf das Zahlenpaar (y_1, y_2) garantiert, dass $y_i \sim N(0, 1);\ i = 1, 2$.

$$y_1 = \sqrt{-2\ln(x1)} \cdot cos(2\pi x_2) \tag{5.2}$$

$$y_2 = \sqrt{-2\ln(x1)} \cdot sin(2\pi x_2) \tag{5.3}$$

[1] George Marsaglia ist Mathematiker und Informatiker. Er setzte sich intensiv mit der Erzeugung und dem Testen von Zufallszahlen auseinander.

[2] Gleichverteilte Zufallszahlen können nach einer Methode von Park&Miller generiert werden. Näheres in [29].

Äquivalent lässt sich $x = h(y)$ schreiben als:

$$x_1 = \exp[-\frac{1}{2}(y_1^2 + y_2^2)]$$
$$x_2 = \frac{1}{2\pi} \arctan(\frac{y_2}{y_1})$$

Wie leicht nachzurechnen ist, ergibt sich die Jacobi–Determinante $\det(\frac{\partial h}{\partial y})$ als Produkt der Standardnormalverteilungsdichten.

$$\det(\frac{\partial h}{\partial y}) = -\left[\frac{1}{\sqrt{2\pi}} e^{\frac{1}{2}\cdot y_1^2}\right] \cdot \left[\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\cdot y_2^2}\right]$$

Bei dem Verfahren vereinfachen sich die Transformationen (5.2), (5.3) dadurch, dass man ausnutzt, dass rsq ebenfalls uniform verteilt ist und anstatt x_1 eingesetzt werden kann. Als Polarkoordinate für $2\pi x_2$ dient der Winkel, den der Punkt (v_1, v_2) bezüglich der x–Achse definiert. In Abbildung (5.1) auf Seite 64 ist dieser blau eingezeichnet; zudem ist der Abstand $\sqrt{rsq}$ mit rot eingefärbt.
Somit lässt sich der Satz von Pythagoras ausnutzen und man spart bei der Implementierung der Polar–Methode den komplexen Aufruf der trigonometischen Funktionen von Sinus und Kosinus.

In der C–Datei `zufallszahlen.c`, deren ausführlicher Quellcode dem Anhang zu entnehmen ist, wurde der erläuterte Zufallsgenerator implementiert. Die Funktion „`ufo`“ erzeugt uniform verteilte Zahlen, die Routine „`gasdev`“ standardnormalverteilte nach der Vorschrift von Marsaglia.

Der Programmablauf gestaltet sich wie folgt:

`zufallszahlen.c - Funktion „double gasdev (long * )“`

Funktionsweise:
Erzeugung einer standardnormalverteilten Zufallszahl $x \sim N(0, 1)$ gemäß Polar–Methode

Eingabeparameter:
Initialisierungswert $idum$ für den Aufruf des Ufo–Zufallszahlengenerators in Form eines Zeigers vom Datentyp long

Ablauf:

1. Überprüfung, ob Initialisierungswert $idum$ negativ:
   - Falls ja: Gehe zu Punkt 2.
   - Falls nein: Gebe in vorherigem Aufruf gespeicherte Zufallszahl $y \sim N(0,1)$ zurück.
2. Generierung des Zahlenpaares (v_1, v_2) mit sequentiellem Aufruf der Routine `ufo` gemäß (5.1).
3. Überprüfe Bedingung für den quadrierten Abstand: $rsq = v_1^2 + v_2^2 \in (0,1)$:
   - Falls ja: Gehe zu Punkt 4.
   - Falls nein: Erzeuge neuen Punkt (v_1, v_2) nach Punkt 2.
4. Berechne $y = v_1 \cdot \sqrt{\frac{-2\ln(rsq))}{rsq}}$ bzw. $y = v_2 \cdot \sqrt{\frac{-2\ln(rsq))}{rsq}}$.

Rückgabe:
Rückgabe einer standardnormalverteilten Zufallszahl y

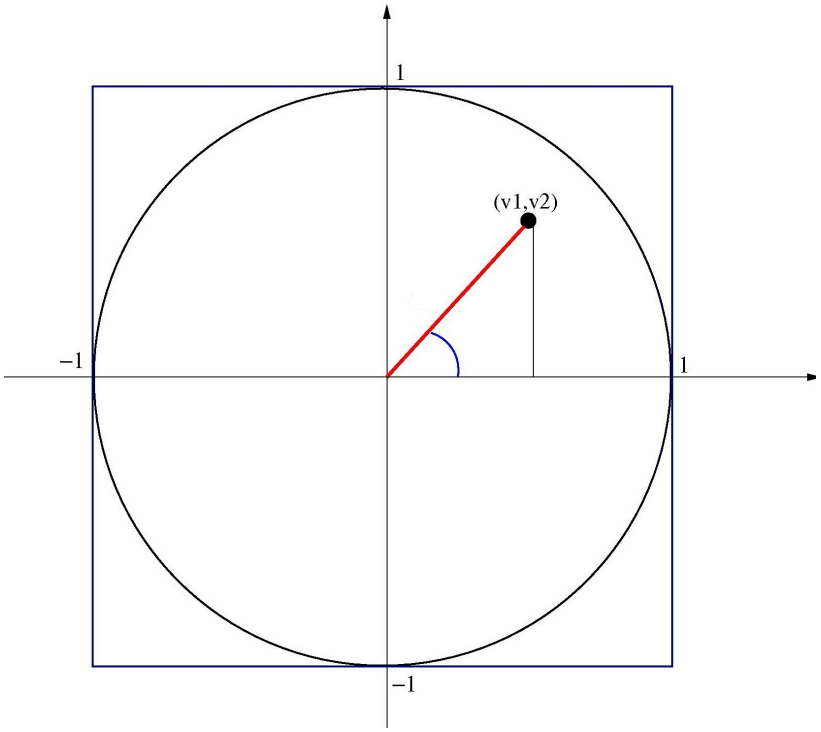

Abbildung 5.1: Geometrie bei Polar-Methode nach Marsaglia

Bemerkung 5.3. *Durch lineare Transformation lassen sich mit der Polar–Methode leicht beliebige normalverteilte Zufallszahlen generieren.*
Ist die Zufallszahl $y \sim N(0,1)$ *verteilt, so gilt für die Variable* $z := a \cdot y + b : z \sim N(b, a^2)$.

5.2 Numerische Integration des Erwartungswerts

Dieser Abschnitt beschäftigt sich mit der numerischen Berechnung des Erwartungswerts, der aufgrund seiner Existenz auf der rechten Seite der Bellman–Gleichung im Laufe des Algorithmus (3.31) sehr oft auszuwerten ist.

In der theoretischen Abhandlung des Algorithmus wurde bereits in Abschnitt (3.2.4) auf S. 32 darauf aufmerksam gemacht, dass der Erwartungswertausdruck der rechten Seite von (3.13)

$$\mathbb{E}\{l(x,\tilde{u}) + \beta\tilde{V}_k(\varphi(x,\tilde{u},z)\} = l(x,\tilde{u}) + \beta \cdot \int_z \tilde{V}_k(\varphi(x,\tilde{u},z))p_Z(z)dz$$

für festes x und $\tilde{u}$ numerisch auszuwerten ist. Die Herausforderung besteht vorwiegend darin, das Integral

$$\int_z \tilde{V}_k(\varphi(x,\tilde{u},z))p_Z(z)dz \tag{5.4}$$

mit der Integrationsvariablen z und der zum Iterationsschritt k gehörigen Wertefunktion $\tilde{V}_k$ im Integranden möglichst gut anzunähern. Da hierfür analytisch keine Stammfunktion zu bestimmen ist, nutzt man das Konzept der numerischen Integration. In der praktischen Anwendung existieren eine Vielzahl von Quadraturformeln, deren Gemeinsamkeit es ist, ein bestimmtes Integral über eine Funktion f möglichst gut zu approximieren.

$$\int_a^b f(x)dx = Q(f) + O(f)$$

Hierbei bezeichnet $Q(f)$ die Quadraturformel und $O(f)$ den Approximationsfehler.
Als $Q(f)$ wird die zusammengesetzte Sehnentrapezformel angewandt, deren Funktionsweise kurz erläutert wird.

Zur Integralnäherung wird der Integrationsbereich $[a,b]$ in mehrere Teilintervalle

$$[x_i, x_{i+1}], i = 0, \ldots, N-1$$

unterteilt, wobei $h = \frac{(b-a)}{N}$ die Intervalllänge bezeichnet und die

$$x_i = a + i \cdot h, i = 0, \ldots, N$$

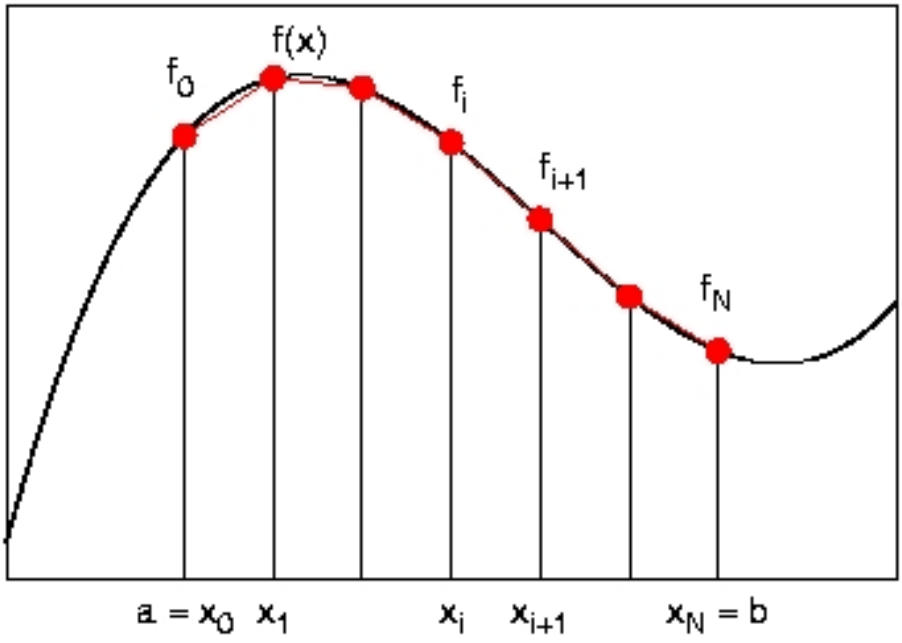

Abbildung 5.2: Zusammengesetzte Trapezregel

die Stützstellen bilden. Das Integral wird zerlegt und jedes Teilintegral $\int_{x_i}^{x_{i+1}} f(x)dx$ wird, wie die nachstehende Figur (5.2) verdeutlicht, durch die Fläche eines Trapez angenähert.
Es ergibt sich somit als zusammengesetzte Quadraturformel die **Trapezsumme**:

$$T(f) = \frac{h}{2}f(a) + h \cdot \sum_{i=1}^{N-1} f(x_i) + \frac{h}{2}f(b) \tag{5.5}$$

Bemerkung 5.4 (Fehlerabschätzung). *Es ist intuitiv klar, dass mit $h \to 0$ bzw. $n \to \infty$ die Trapezsumme gegen den exakten Integralwert konvergiert.*
Unter der Annahme, dass die zu integrierende Funktion $f \in C^2[a,b]$ ist, gilt nachweisbar die Fehlerabschätzung:

$$|\int_a^b f(x)dx - T(f)| \leq \frac{(b-a)}{12} \cdot \|f''\|_\infty \cdot h^2$$

Um mit dieser Quadraturformel das für unsere Zwecke relevante Integral (5.4) auszuwerten, ist es nötig, den Integrationsbereich näher zu betrachten. Für die Berechnung des Erwartungswerts ist grundsatzlich ganz $\mathbb{R}$ zu berücksichtigen. Jedoch ist es in Anbetracht der Tatsache, dass die Integrationsvariable z als eindimensionale Zufallsgröße des Kontrollsystems einer Normalverteilung mit Erwartungswert 0 und Varianz σ^2 genügt, möglich das Integrationsintervall auf einen endlichen Abschnitt zu verkleinern.

Der Verlauf der Dichtefunktionen p_Z (vgl. Abbildung (3.1)) zeigt, dass nur solche z aus einem symmetrischen Intervall $[-a, a], a \in \mathbb{R}$ um den Nullpunkt Wahrscheinlichkeit $p_Z(z) > 0$ tragen. Daher wird dieses Intervall $[-a, a]$ im Rahmen der Implementierung in N Teilintervalle zerlegt, so dass für die Trapezsumme über die Dichte gilt:

$$T(p_Z) \approx 1, \quad h = \frac{2 \cdot a}{N}$$

Somit erfolgt die Berechnung von (5.4) mit diskretisierter Zufallsvariable $z \in [-a, a]$ als Trapezsumme gemäß:

$$\int_{-a}^{a} \tilde{V}_k(\varphi(x, \tilde{u}, z)) p_Z(z) dz = \frac{h}{2} \cdot \left(\tilde{V}_k(\varphi(x, \tilde{u}, z_0)) p_Z(z_0) + \tilde{V}_k(\varphi(x, \tilde{u}, z_N)) p_Z(z_N) \right) + \\ + h \cdot \sum_{i=1}^{N-1} \tilde{V}_k(\varphi(x, \tilde{u}, z_i)) p_Z(z_i)$$

Die Trapezformel wurde im späteren C–Anwendungsprogramm, das für die Modelle des sechsten Kapitels geschrieben ist, als eine eigenständige Funktion implementiert.

Listing 5.1: Code zur Trapezregel

```
double trapezformel( double a, double b, unsigned int N,
double *state, double u, double * fval, qgrid * g, int * flag)
{       double sum, h;
        double x;
        int i,erg;
        calls++;
        sum=0.;
        h = ( b - a) / N;
        sum = 0.5 * (integrand(state,u,a,fval,g,flag) +
        integrand(state,u,b,fval,g,flag));
        if( N > 1)
        {
        for ( i = 1; i <= N-1; i++)
                {
                        x = a + i*h;
                        sum +=  integrand(state,u,x,fval,g,flag);
                }
        }
        return sum * h;
}
```

Bemerkung 5.5. *Der Aufruf der Funktion* `integrand(...)` *liefert jeweils den Wert des Produkts* $\tilde{V}_k(\varphi(x, \tilde{u}, z_i)) \cdot p_Z(z_i)$ *zurück. Der Parameter* `flag` *dient als Variable für den Fehlerstatus.*
Der Funktionseingabeparameter `state` *übergibt den Systemzustand,* `u` *die ausgewählte Kontrolle,* `fval` *bezeichnet den nächsten Zustand nach Auswertung von* φ *und* `g` *die Gitterstruktur, mit welcher sich der nachfolgende Abschnitt befasst. Die statische Variable* `calls` *zählt, wie oft die Trapezregel während des Algorithmus (3.31) aufgerufen wurde.*

5.3 Adaptive Gitter mit GRIDGEN

Der Algorithmus zur Berechnung der optimalen Wertefunktion basiert auf adaptiven Gittern, die eine räumliche Diskretisierung des Zustandsraums des zeitdiskreten Kontrollsystems vornehmen. Das Optimalitätsprinzip in Form des DP–Operators $\mathcal{T}$ (3.18) wird in (3.31) stets nur auf im Gitter gelegene Punkte angewandt. Wie jene Gitter in der Programmiersprache C implementiert werden, beschreibt dieser Abschnitt über den Gittergenerator GRIDGEN. Die von *Prof. Dr. Grüne* entwickelte Gitterdatenstruktur, die teils modifiziert wurde, und ihre wichtigsten Routinen werden überblicksartig vorgestellt. Als Grundlage dient die Dokumentation [12]. Die Ausführungen beziehen sich auf die Version *0.99t* des Gittergenerators in Form des C–Headerfiles `gridgen.h` und des Programms `gridgen.c`.[3]

Ein Gitter Γ wird in der Datenstruktur `qgrid` in hierarchischer Baumstruktur abgespeichert. Dabei beinhaltet `qgrid` selbst wieder mehrere C-Strukturen. Unter der Vielzahl von Datenelementen in der Struktur enthält folgender Codeauszug nur die grundlegenden Variablen sowie die Elemente, die für die Traversierung relevant sind:[4]

Listing 5.2: Datenstruktur qgrid

```
typedef struct sqgrid
  {
   /* basic grid data */
    int  dim;           /* dimension */
    real ql[MAXDIM];    /* lower corner of root element */
    real qw[MAXDIM];    /* width of root element */
    int  qn[MAXDIM];    /* elements per dimension in basic grid */
    int  max_simplex;   /* number of leaf elements in grid */
    int  bascubes;      /* number of elements in basic grid */
    int  max_vertex;    /* number of vertices in grid */
```

[3] Beide Programme finden sich auf der beigefügten CD-ROM.
[4] (...) steht für Auslassungen im Quellcode

```
    int   hanging;        /* number of hanging nodes in grid */
    int   level;          /* refinement levels for create_grid */
    cube *roots;          /* array of root elements */
    int   depth;          /* depth of grid, updated by griddepth */
        (...)

#ifdef _NODES
    /* data for traversing with next_node */
    qnode *anode;         /* actual node */
    int    ali;           /* local  index of actual node */
    int    agi;           /* global index of actual node */
    real   ax[MAXDIM];    /* coordinates  of actual node */
#endif
        (...)
 } qgrid;
```

In der Gitterstruktur `qgrid` ist das Element `anode` vom Typ `qnode` enthalten, das beim Durchlauf der Baumknoten stets den aktuellen Knoten anspricht. Diese Struktur ist wie folgt definiert:

Listing 5.3: Datenstruktur qnode

```
typedef struct sqnode
  { real value;
    real dpvalue;        /*neu hinzugefügtes Element*/
    unsigned char status;
    struct sqnode **hooks;
    int index;
  } qnode;
```

Die Datenstruktur `qnode` besitzt fünf Datenelemente - das Element `dpvalue` wurde gegenüber der originalen Version von *Prof. Dr. Grüne* hinzugefügt - mit folgender Bedeutung:

value	Wert $\tilde{V}_k(E_i)$ der approximativen, bilinearen Wertefunktion im Gitterpunkt
dpvalue	Wert des DP–Operators $\mathcal{T}(\tilde{V}_k)$, ausgewertet im Knotenpunkt
status	Ganzzahlige Statusvariable; mögliche Stati $\in \{0, 1, \ldots 9\}$; Status 6 markiert z.B. einen einzufügenden Testpunkt
hooks	Verweis auf Knoten, von denen hängende Knoten [5] abhängen
index	Knotenindex für interne Identifikation

[5] Dies sind Knoten, die nicht für alle angrenzenden Zellen Eckknoten darstellen.

Tabelle 5.1: Datenelemente der Knotenstruktur qnode

Bemerkung 5.6. *Der auftretende Datentyp* `real` *ist lediglich ein Alias für den Standarddatentyp* `double` *der Fließkommazahlen.*

Sämtliche Routinen des Gittergenerators sind im Headerfile definiert und im Programm `gridgen.c` deklariert. Die für den Algorithmus (3.31) relevantesten werden nachstehend in ihrer Funktionsweise kurz beschrieben.

,,`qgrid* create_grid(real *lo, real *hi, real *Delta, int dim, int level)`‘‘

Funktionsweise:
Erzeugung eines Gitters für ein dim– dimensionales Gebiet

Eingabeparameter:

lo	untere Ecke des zu diskretisierenden Gebiets
hi	oberer Begrenzungspunkt des Gebiets
$Delta$	Vorgabe der Zellengröße
dim	Dimension des Gebiets
$level$	Variable für Erzeugung regel- bzw. unregelmäßiger Gitter

---

Rückgabe:
Zeiger auf die neu angelegte $qgrid$ Struktur

,,`void delete_grid(qgrid *tg)`‘‘

Funktionsweise:
Löschen der Gitterstruktur tg

Eingabeparameter:
Zeiger tg auf die zu löschende Gitterstruktur

Rückgabe:
keine

,,`int first_node(qgrid *tg, real *x)` bzw. `int next_node(qgrid *tg, real *x)`‘‘

Funktionsweise:
Zugriff auf den ersten bzw. nächsten Knoten des Gitters

Eingabeparameter:

tg	Zeiger auf das Gitter
x	Array, das Koordinaten des Gitterpunkts zugewiesen bekommt

Rückgabe:
Index des ersten Knotens bzw. des nächsten Knotens innerhalb der Gitterstruktur
Rückgabe von -1, falls ein Fehler auftritt

Bemerkung 5.7. *Die beiden Funktionen dienen dem Knotenzugriff. Sie stellen eine Art Interfacefunktion dar und durchlaufen so intern die Eckpunkte des Gitters.*
Die Traversierung des Gitters erledigt im Hauptprogramm eine einfache `do-while` *Schleife.*

Listing 5.4: Knotendurchlauf

```
index = first_node(g,x)
do
{
        ...                 /*beliebige Operationen*/
        index = next_node(g,x);
}
while(index!=-1);
```

Der Durchlauf der Testpunkte, die für die Bestimmung des lokalen Fehlerschätzers in Schritt 3 von (3.31) benötigt werden, erfolgt analog.
Die Funktionen für deren Traversierung lauten:

- `int first_testpoint( qgrid * x, real * x)`
- `int next_testpoint( qgrid * x, real * x)`

Die wesentliche Operation auf den Knoten des Gitters Γ besteht darin, den Wert der bilinearen Funktion $\tilde{V}_k \in \mathcal{W}$ auszulesen bzw. zu setzen. Dafür eignen sich die folgenden beiden Routinen:

„`real current_nodevalue(qgrid *g)`“

Funktionsweise:
Abfrage des Werts der Gitterfunktion im aktuellen Knoten

Eingabeparameter:
Zeiger g auf die Gitterstruktur

Rückgabe:
Knotenwert $\tilde{V}_k(E_i)$ wird als `double`-Wert zurückgegeben.

,,`int set_current_nodevalue(qgrid *g, real v)`‘‘

Funktionsweise:
Setzen des Werts der Gitterfunktion im aktuellen Knoten

Eingabeparameter:

g	Zeiger auf das Gitter
v	der zu setzende Knotenwert

Rückgabe:
Status über Erfolg des Aufrufs: im Erfolgsfall 0, ansonsten 1.

Soll der Wert für einen beliebigen Punkt $x \in \Gamma$ abgefragt werden, so wird die rechenzeitintensivere Routine

- `real value (qgrid *g, real *x, int *flag)`

mit dem zusätzlichem Parameter $flag$, der als Fehlerstatusvariable dient, aufgerufen.
Soll hingegen der numerische Wert aus dem Optimalitätsprinzip $dpvalue$ für die Knoten gesetzt bzw. ausgelesen werden, so können die beiden in `gridgen.c` neu hinzugefügten Routinen verwendet werden:

- `real get_dpvalue(qgrid *tg)`
- `void set_dpvalue(qgrid *tg, real val)`

Ein Gitter lässt sich auch jederzeit dauerhaft abspeichern und für eine graphische Aufbereitung als ASCII–Datei nach MATLAB exportieren.

,,`int export_gridandval(qgrid *g, char * filename)`‘‘

Funktionsweise:
Exportierung nach MATLAB im ASCII-Format (`.asc`)

Eingabeparameter:

g	Zeiger auf das Gitter
$filename$	Dateiname für die Export–Datei

Rückgabe:
Status über Erfolg des Aufrufs: im Erfolgsfall 0, ansonsten 1.

Die Routine

- `export_val2d(qgrid * tg, char * filename, int res)`

speichert hingegen nur zeilenweise Koordinaten und Werte von Punkten eines zweidimensionalen Gitters in die Datei mit dem Namen $filename$ ab. Mit res übergibt man die Anzahl an Unterteilungen für beide Koordinatenrichtungen.

Abschließend wird noch darauf hingewiesen, dass für den Verfeinerungsschritt zwei Routinen von großer Bedeutung sind.
Zur Erzeugung von Testpunkten ruft man jeweils zu Beginn von Schritt 3 in (3.31)

- `void prepare_adaptation (qgrid *g)`

auf. Um Gitterpunkte in die aktuelle Gitterstruktur g einzufügen, benutzen wir

- `void insert_current_testpoint(qgrid * g)`.

Damit wird der gegenwärtig betrachtete Testpunkt als „einzufügend“ markiert.

Am Ende eines jeden Verfeinerungsschritts steht der Aufruf von

- `void complete_adaptation (qgrid *g)`.

Dabei erfolgt am Bildschirm die Ausgabe einer Statistik über die Veränderungen des Gitters g im beendeten Adaptionsschritt.

Der Gittergenerator bietet durchaus noch mehr Funktionalität. So können beispielsweise Routinen zur Zeitmessung (`tic(), toc()`) sowie zum Laden eines Gitters (`load_val()`)genutzt werden. Um Genaueres über deren Funktionsweise zu erfahren, wird auf die Kommentierung im Quellcode sowie auf [12] verwiesen.

Zur vollen Nutzung des Pakets GRIDGEN im eigenen Programm muss lediglich in der Kopfzeile mit

```
#include "gridgen.h"
```

die Headerdatei eingebunden werden.

6 Praktische Anwendung der Verfahren

Dieses Kapitel zeigt die praktische Anwendbarkeit der in den vorherigen Kapiteln erläuterten Verfahren auf. Zur Analyse der gewonnenen Approximationen aus dem Perturbationsansatz und der Näherungen, die mit Hilfe des Algorithmus der dynamischen Programmierung (3.31) bestimmt werden, betrachten wir zwei verschiedene Wachstumsmodelle der Ökonomie. Die beiden Modelle entstammen dem in Abschnitt (2.2) vorgestellten Referenzmodell von Ramsey, unterscheiden sich jedoch in der Zielfunktion. Die beim Ramsey Modell erfolgten ökonomischen Überlegungen spiegeln sich in den zwei Anwendungsobjekten wider.
Der erste Abschnitt definiert beide Modelle, insbesondere deren kennzeichnende Parameter, und interpretiert sie nach ökonomischen Gesichtspunkten. Es wird die Existenz eines Gleichgewichts hergeleitet, das sich in Abhängigkeit von den gewählten Modellparametern ergibt.
Im zweiten Abschnitt werden die numerischen Ergebnisse der mit der Perturbationsmethode berechneten indirekten Approximation $\hat{V}$ sowie die über den direkten Ansatz ermittelte Wertefunktion $\hat{V}_{dir}$ für die optimale Wertefunktion präsentiert. Ein Vergleich zwischen den Perturbationslösungen, den Lösungen der dynamischen Programmierung und der exakten analytischen Lösung, welche für Modell A existiert, wird in Abschnitt drei vorgenommen. Das Kapitel wird mit einigen numerischen Simulationen, die für beide Modelle durchgeführt werden, abgerundet.

6.1 Die ökonomischen Beispielmodelle

In diesem Abschnitt werden die beiden Modelle eingeführt und deren Bezug zum Referenzmodell nach Ramsey hergestellt. Für beide Modelle werden die Gleichgewichtspunkte, die für den Lösungsansatz über Perturbationsmethoden unverzichtbar sind, mit Hilfe einer MATLAB Routine berechnet und anschließend ökonomisch interpretiert.

6.1.1 Das Modell A

Das Modell A ist zweidimensional im Zustandsraum X und eindimensional im Raum der Kontrollen u. Es schreibt sich als zeitdiskretes, stochastisches Kontrollproblem gemäß (3.12) wie folgt:

Beispiel 1 (Modell A).

$$\max_{u_t} E\left[\sum_{t=0}^{\infty} \beta^t \log(u_t)\right]$$

mit der diskreten Dynamik $x_{t+1} = \varphi(x_t, u_t, \epsilon_t)$*, wobei* φ *gegeben ist als:*

$$\varphi(x, u, \epsilon) = \begin{pmatrix} Ae^{x_2} x_1^{\alpha} + (1-\delta) \cdot x_1 - u \\ \rho x_2 + \sigma\epsilon \end{pmatrix}$$

Hierbei ist $x_{t,1} > 0, x_{t,2} \in \mathbb{R}$ *und* $\epsilon \sim N(0,1)$.

Bedeutung der Zustandsvariablen:
Die Variable x_1 bemisst den Pro-Kopf Kapitalstock in einer geschlossenen Volkswirtschaft, der Zustand x_2 weist den technologischen Fortschritt logarithmiert aus.
Kontrolle:
Die Kontrolle u bezeichnet die Konsumgröße des Agenten des Modells.
Parameter:
Das Modell ist mit zahlreichen Parametern versehen, denen folgender Wert und Bedeutung zugewiesen sind:

Parameter	Wert	Bedeutung
β	0.95	Diskontierungsfaktor
A	5	Technologischer Wandel
α	0.34	Produktionselastizität des Faktors Kapital
δ	1	Kapitalabschreibungsrate
ρ	0.9	Faktor für x2–Dynamik
σ	0.008	Störungsparameter für stochastische Schocks

Tabelle 6.1: Parameter von Modell A

In den weiteren Ausführungen des Kapitels werden wir insbesondere den Abschreibungsparameter δ variieren.

Bemerkung 6.1.

- *Die deterministische Dynamik für* x_1 *fasst die Ramsey–Gleichungen* (2.6) *und* (2.7) *zusammen. Es gilt:*

$$K_{t+1} = (1-\delta) \cdot K_t + I_t = (1-\delta) \cdot K_t + (Y_t - C_t) \equiv (1-\delta) \cdot x_{t,1} + Ae^{x_{t,2}} x_{t,1}^{\alpha} - u_t$$

- *Die Dynamik für x_2 ist hingegen, wirtschaftliche Unsicherheit repräsentierend, stochastisch und folgt einem stationären AR(1)–Prozess ($\rho < 1$). Damit ist die Annahme eines partitionierten Zustandsraums $x_t = [x_t^1, x_t^2]$ im Perturbationsmodell erfüllt.*

- *Die betrachteten Zustandsgrößen sind als Pro-Kopf Größen aufzufassen. Der Produktionsfaktor Arbeitskräfte L tritt nicht im Modell auf.*
  Dies lässt sich ausgehend von der Cobb-Douglas Produktionsfunktion $P(K, L)$ herleiten. Dazu werden die Pro-Kopf-Größen

$$k := \frac{K}{L}; \quad c := \frac{C}{L}; \quad y := \frac{Y}{L}$$

  definiert. Somit gilt:

$$y = \frac{P(K,L)}{L} = P(\frac{K}{L}, \frac{L}{L}) = P(k, 1) = P(k)$$

- *Die zugrundeliegende Produktionsfunktion von der Klasse Cobb–Douglas kann man der Dynamik für x_1 ablesen:*

$$P(k, T) = P(x_1, e^{x_2}) = A \cdot x_1^{\alpha} \cdot e^{x_2}$$

  Der Output bestimmt sich somit durch die Kapitalintensität k ($\equiv x_1$) sowie den technologischen Fortschritt T ($\equiv e^{x_2}$). Wir bezeichnen k allerdings weiterhin als Kapitalstock.

Herleitung des Gleichgewichts

Ausgehend von der Hamilton–Funktion, die in (3.33) des Kapitels drei eingeführt wurde, wollen wir für das Modell A die Gleichgewichtsbedingungen herleiten, um damit den steady state dieser Ökonomie zu charakterisieren.
Die Hamilton–Funktion lautet demnach:

$$H^t(x_t, \lambda_t, u_t) = \mathbb{E}_t \left\{ \beta^{t+1} \cdot \log(u_t) + \lambda_{t+1,1} \cdot (Ae^{x_{t,2}} x_{t,1}^{\alpha} + (1-\delta) \cdot x_{t,1} - u) + \lambda_{t+1,2} \cdot (\rho x_{t,2} + \sigma \epsilon_t) \right\}$$

Dies führt bei Anwendung des stochastischen Maximumprinzips auf die folgenden Bedingungen:

I. **Optimalitätsbedingung für den Zustand**

$$\begin{aligned} \mathbb{E}_t[x_{t+1,1}] &= \mathbb{E}_t[Ae^{x_{t,2}} x_{t,1}^{\alpha} + (1-\delta) \cdot x_{t,1} - u] \\ \mathbb{E}_t[x_{t+1,2}] &= \mathbb{E}_t[\rho x_{t,2} + \sigma \epsilon_t] \end{aligned} \tag{6.1}$$

II. **Optimalitätsbedingung für die adjungierte Variable**

$$\begin{aligned}\lambda_{t,1} &= \mathbb{E}_t[\lambda_{t+1,1} \cdot (\alpha A e^{x_{t,2}} x_{t,1}^{\alpha-1} + (1-\delta))] \\ \lambda_{t,2} &= \mathbb{E}_t[\lambda_{t+1,2} \cdot \rho + \lambda_{t+1,1} \cdot (A e^{x_{t,2}} x_{t,1}^{\alpha})]\end{aligned} \tag{6.2}$$

III. **Extremalbedingung**

$$\mathbb{E}_t \left[\frac{\beta^{t+1}}{u_t} - \lambda_{t+1,1} \right] = 0 \tag{6.3}$$

Geschicktes Vereinfachen der Gleichungen (6.1) - (6.3) führt auf die kompakte Form der Gleichgewichtsbedingungen (vgl. (3.36)):

$$\mathbb{E}_t \left\{ f(x, x', u, u') \right\} = 0$$

mit

$$f(x, x', u, u') := \begin{bmatrix} x_1' - A e_2^x x_1^{\alpha} - (1-\delta) x_1 + u \\ x_2' - \rho x_2 \\ \frac{\beta}{u'} \cdot (\alpha A e^{x_2'} {x_1'}^{\alpha-1} + 1 - \delta) - \frac{1}{u} \end{bmatrix} \tag{6.4}$$

Im Gleichgewichtspunkt gilt:

$$\begin{aligned} x_1' &= x_1, \\ x_2' &= x_2, \\ u' &= u. \end{aligned}$$

Somit ergibt sich aus der zweiten Zeile der Gleichgewichtsbedingung (6.4):

$$x_2 - \rho x_2 = 0 \Longrightarrow x_2 = 0.$$

In die erste Zeile der Gleichgewichtsbedingungen eingesetzt, folgt:

$$\begin{aligned} & x_1 - A x_1^{\alpha} - (1-\delta) x_1 + u = 0 \\ \Longleftrightarrow & u = A x_1^{\alpha} + (1-\delta) x_1 - x_1 \\ \Longleftrightarrow & u = A x_1^{\alpha} - \delta x_1. \end{aligned}$$

Diesen Term setzt man für u und u' in die dritte Zeile ein

$$\frac{\beta}{A x_1^{\alpha} - \delta x_1} \cdot (\alpha A e^{x_2'} {x_1'}^{\alpha-1} + 1 - \delta) - \frac{1}{A x_1^{\alpha} - \delta x_1} = 0$$

und es ergibt sich nach kurzer Rechnung:

$$x_1 = \left(\frac{1 - \beta(1-\delta)}{\beta \alpha A} \right)^{\frac{1}{\alpha-1}}$$

Steady state:
Somit ist der steady state für Modell A gegeben.

$$\begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \end{pmatrix} = \begin{pmatrix} (\frac{1-\beta(1-\delta)}{\beta\alpha A})^{\frac{1}{\alpha-1}} \\ 0 \end{pmatrix} \tag{6.5}$$

und der Pro-Kopf-Konsum im Gleichgewicht $\bar{x}$ ergibt sich zu:

$$\bar{u} = Ax_1^{\alpha} - \delta x_1. \tag{6.6}$$

Um die Werte des Gleichgewichts ökonomisch interpretieren zu können, wollen wir die Gleichgewichtspunkte in Abhängigkeit vom variierenden Abschreibungsparameter δ betrachten (die restlichen Parameterwerte bleiben wie in Tabelle (6.1)).
Für den Wert $\delta = 1$, d.h. der Kapitalstock wird während pro Zeitschritt $t \rightarrow t+1$ vollständig abgeschrieben, ergibt sich der steady state zu

$$(\bar{x}_1, \bar{x}_2, \bar{u}) \approx (2.0673, 0.0, 4.3331).$$

Die nachfolgende Tabelle und Graphik zeigt die Entwicklung des ökonomischen Gleichgewichts für Modell A abhängig von der Wahl von δ. Die in MATLAB implementierte Funktion `SS_plot.m` [1] übernimmt die Berechnung des steady state bei Vorgabe der Abschreibungsrate.

δ	$\bar{\mathbf{x}}_\mathbf{1}$	$\bar{\mathbf{x}}_\mathbf{2}$	$\bar{\mathbf{u}}$
0.00	193.50	0.00	29.95
0.10	38.55	0.00	13.45
0.20	17.97	0.00	9.76
0.30	10.84	0.00	7.99
0.40	7.43	0.00	6.92
0.50	5.49	0.00	6.18
0.60	4.27	0.00	5.63
0.70	3.44	0.00	5.20
0.80	2.84	0.00	4.86
0.90	2.40	0.00	4.57
1.00	2.07	0.00	4.33

Tabelle 6.2: Gleichgewichte abhängig von δ

[1] Siehe (B.1) im Anhang

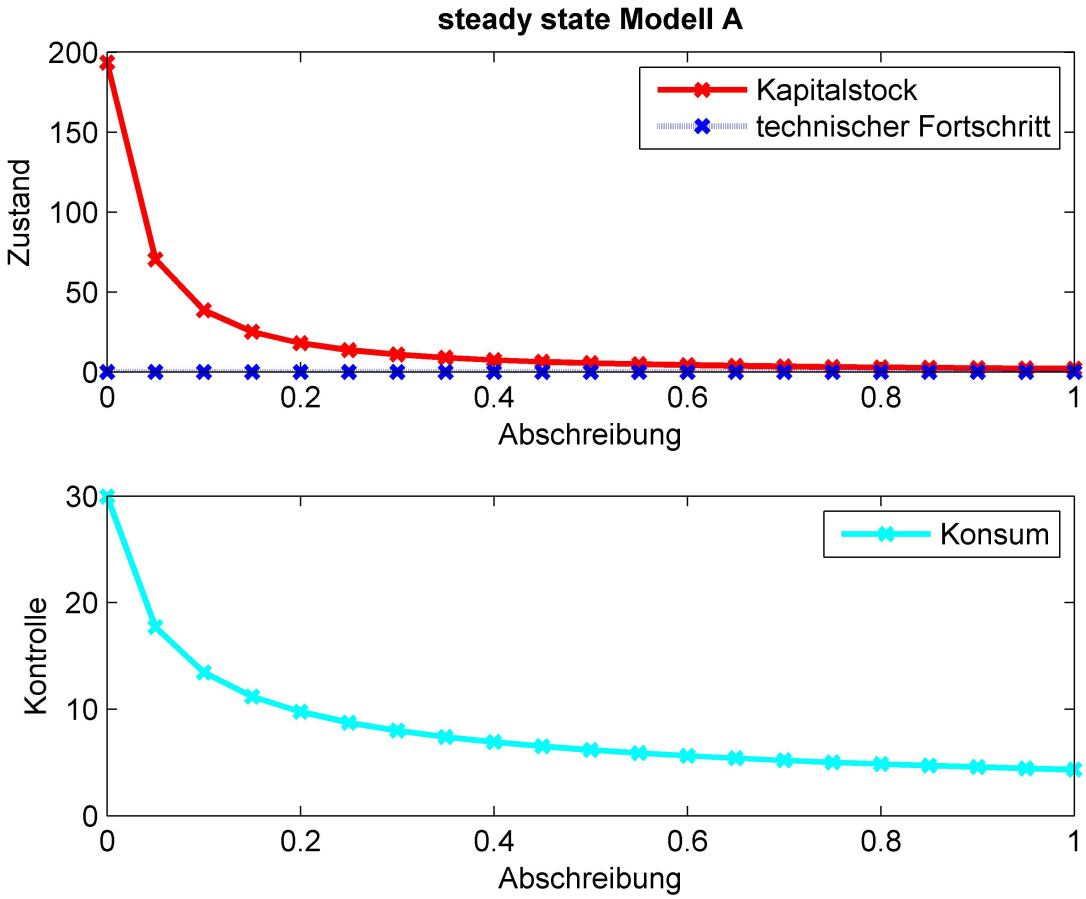

Abbildung 6.1: Gleichgewichte abhängig von der Abschreibung

Interpretation:
Die Abbildung verdeutlicht die Wirkung einer Erhöhung der Abschreibungsrate auf den steady state. Umso höher die Rate der Kapitalabschreibung im Modell gewählt wird, desto niedriger ist der gleichgewichtige Kapitalstock $\bar{x}_1$. Der zweite Effekt besteht darin, dass aufgrund des verminderten Outputs Y der Agent Haushalt weniger konsumieren kann und so sinkt der Konsum $\bar{u}$ spürbar. Stellt sich für $\delta = 0.2$ noch ein Kapitalstock von ca. 18 Einheiten ein, so beträgt bei Verdopplung von δ auf 0.4 der Kapitalstock mehr als die Hälfte weniger ($\delta \approx 7.43$) und nähert sich bei vollständiger Abschreibung dem Gleichgewichtskapitalstock $\bar{x}_1 \approx 2.07$.

Das Aufzeigen der Lage des Gleichgewichts abhängig von den Parametern liesse sich für alle Einflussgrößen anstellen, wird aber im Rahmen der Arbeit nur für die variable Abschreibungsrate δ durchgeführt.

6.1.2 Das Modell B

Beispiel 2 (Modell B).

$$\max_{u_t} E\left[\sum_{t=0}^{\infty} \beta^t \log(u_t) \cdot e^{\kappa \cdot x_2}\right]$$

mit unveränderter Dynamik φ als:

$$\varphi(x,u,\epsilon) = \begin{pmatrix} Ae^{x_2}x_1^{\alpha} + (1-\delta)\cdot x_1 - u \\ \rho x_2 + \sigma\epsilon \end{pmatrix}$$

Hierbei ist $x_{t,1} > 0, x_{t,2} \in \mathbb{R}$ und $\epsilon \sim N(0,1)$.

Beispielmodell B, bei dem gegenüber Modell A eine Änderung der Zielfunktion vorgenommen wurde, besitzt einen zusätzlichen Parameter κ. Alle anderen Parameter bleiben dem Wert nach gleich gesetzt.

Parameter	Wert	Bedeutung
κ	2	Verstärkungsfaktor für technologischen Einfluss

Tabelle 6.3: Parameter von Modell B

Bemerkung 6.2. *Wählt man $\kappa = 0$, so ergibt sich als Spezialfall das Modell A. Unsicherheit in der wirtschaftlichen Entwicklung drückt das Modell mit größerem Wert für κ aus.*

Gleichgewicht des Modells B

Trotz der veränderten Nutzenfunktion $l(x,u) = \log(u) \cdot e^{2x_2}$ ergeben sich für das steady state der Modellökonomie keine Veränderungen.
Die Rechnung zur Herleitung der Gleichgewichtsbedingungen erfolgt analog zu Modell A. Diese führt auf folgende Gleichgewichtsbedingungen:

$$E_t\left(f(x,x',u,u')\right) = 0 \quad mit\ f(x,x',u,u') := \begin{pmatrix} x_1' - Ae_2^x x_1^{\alpha} - (1-\delta)x_1 + u \\ x_2' - \rho x_2 \\ \frac{\beta}{u'} \cdot e^{\kappa x_2'} \cdot (\alpha A e^{x_2'} {x_1'}^{\alpha-1} + 1 - \delta) - \frac{1}{u} \cdot e^{\kappa x_2} \end{pmatrix}. \tag{6.7}$$

Die dritte Zeile von f, aus der Maximumsgleichung stammend, hat sich verändert.
Der Gleichgewichtspunkt bleibt jedoch (6.5) entsprechend unverändert.

$$\begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \end{pmatrix} = \begin{pmatrix} (\frac{1-\beta(1-\delta)}{\beta\alpha A})^{\frac{1}{\alpha-1}} \\ 0 \end{pmatrix} \tag{6.8}$$

Der Grund hierfür ist, dass Zeile 2 von (6.7) $\bar{x}_2 = 0$ impliziert. So verschwindet der zusätzliche Term $e^{\kappa \cdot x_2} = 1$ in den Bedingungen. Für die Konsumvariable im Gleichgewicht gilt wie in (6.6):

$$\bar{u} = Ax_1^{\alpha} - \delta x_1. \tag{6.9}$$

Die ökonomischen Interpretationen im Hinblick auf die Abhängigkeit des steady state von den Modellparametern bleiben somit auch für Modell B gültig.

6.2 Numerische Ergebnisse der Perturbationsmethoden

6.2.1 Modell A

In diesem Abschnitt werden unter Einsatz der Perturbationsroutinen die Approximationen für das Modell A berechnet und die numerischen Ergebnisse untersucht. Insbesondere ist es zum Zwecke einer Genauigkeitsanalyse von Vorteil, dass eine analytische Lösung für das Beispiel mit den gewählten Parametern gemäß Tabelle (6.1) existiert.

Geschlossene Lösungen sowohl für die optimale Wertefunktion V als auch für die optimale Kontrolle u sind gegeben:

Optimale Wertefunktion - exakt:

$$V(x) = B + C \cdot \ln(x_1) + Dx_2 \tag{6.10}$$

mit

$$B = \frac{\ln((1-\beta\alpha)A) + \frac{\beta\alpha}{1-\beta\alpha}\ln(\beta\alpha A)}{(1-\beta)}$$

$$C = \frac{\alpha}{(1-\alpha\beta)}$$

$$D = \frac{1}{(1-\alpha\beta)(1-\rho\beta)}$$

Die nachfolgende Abbildung zeigt deren Verlauf. Die optimale Wertefunktion ist in x_1-Richtung

gekrümmt, in x_2-Richtung linear ansteigend. Das Krümmungsverhalten nimmt bezüglich der Kapitalstock–Achse mit $x_1 \to 0$ zu. Wie zu erwarten ist, steigt der Gesamtnutzen in der Modellwirtschaft mit zunehmendem Kapitalstock und mit erhöhtem technologischen Fortschritt. So stellt sich z.B. für $x = (10, 0.32)$ ein maximaler, optimaler Wert von 33.37 ein.

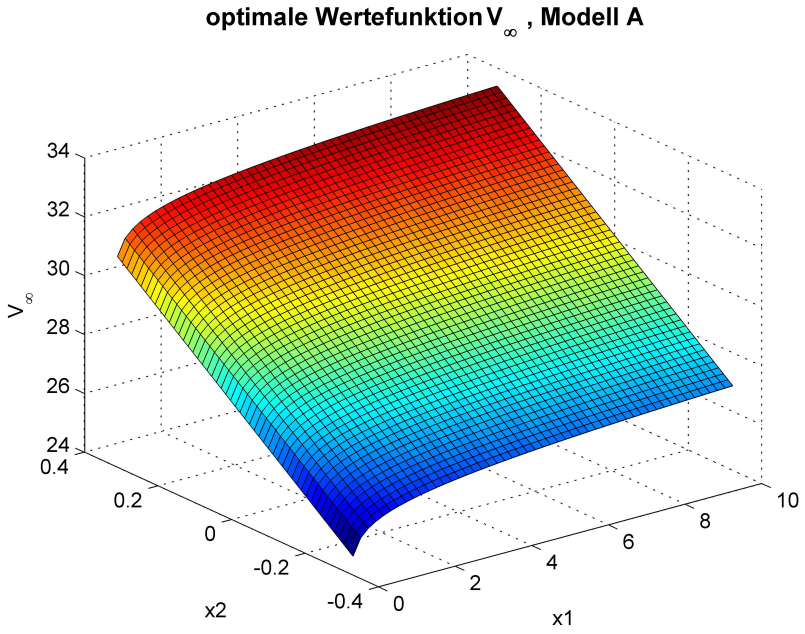

Abbildung 6.2: Exakte opt. Wertefunktion Modell A

Optimale Steuerung - exakt:

$$u(x) = (1 - \alpha\beta) A e^{x_2} x_1^{\alpha} \tag{6.11}$$

Anhand von Abbildung (6.3) erkennt man, dass der den Konsum repräsentierende Kontrollwert mit höherem Kapitalstock anwächst. Eine Steigerung des Konsums durch positive technologische Schocks macht sich erst für größere x_1-Werte bemerkbar. So ist für $x_1 \to 10$ eine Krümmung in x_2-Richtung aufgrund des positiven Einflusses der Technologieentwicklung deutlich erkennbar.

Nun wollen wir die Lösungen der Perturbationsmethoden, die mit Hilfe der Perturbationsroutinen in MATLAB gewonnen werden, angeben. Die MATLAB Routine `ModellA_run.m` [2] berechnet die folgenden Koeffizienten für die Kontrollfunktion g und die Dynamik h aus dem Perturbationsmodell.

[2] Siehe ausführlich kommentierter Quellcode im Anhang (B.1)

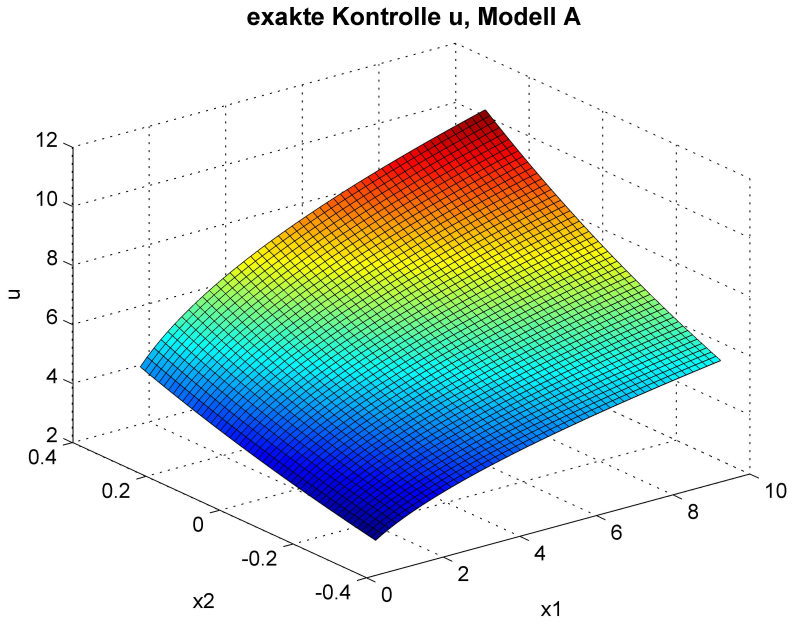

Abbildung 6.3: Exakte Kontrolle Modell A

Koeffizienten[3] der linearen Terme:

$$g_{x_1} = 0.7126, \quad g_{x_2} = 4.3331$$

$$h_{x_1} = \begin{pmatrix} 0.3400 \\ 0 \end{pmatrix}, \quad h_{x_2} = \begin{pmatrix} 2.0673 \\ 0.9000 \end{pmatrix}$$

Koeffizienten der quadratischen Terme für x:

$$g_{x_1x_2} = -0.2275, \quad g_{x_1x_2} = g_{x_2x_1} = 0.7126, \quad g_{x_2x_2} = 4.3331$$

$$h_{x_1x_1} = \begin{pmatrix} -0.1085 \\ 0 \end{pmatrix}, \quad h_{x_1x_2} = h_{x_2x_1} = \begin{pmatrix} 0.3400 \\ 0 \end{pmatrix} \quad h_{x_2x_2} = \begin{pmatrix} 2.0673 \\ 0 \end{pmatrix}$$

Koeffizienten der quadratischen Terme für σ:

$$g_{\sigma\sigma} = 2.9836 \cdot 10^{-15}, \quad h_{\sigma\sigma} = \begin{pmatrix} -0.2984 \cdot 10^{-14} \\ 0 \end{pmatrix}$$

Dass die letzten beiden Koeffizienten nahezu 0 sind, zeigt auf, dass der stochastische Anteil in der Approximation eine vernachlässigbare Rolle spielt. Dies sieht man auch daran, dass weder die geschlossene Lösung V (6.10) noch u (6.11) vom Perturbationsparameter σ abhängt.

[3] Auf vier Nachkommastellen genau angegeben.

Setzt man obige Werte in die Taylorreihe ein, so ergeben sich die approximierte Kontrolle $\hat{u} = g(x, \sigma)$ und die Dynamik $\hat{x}' = h(x, \sigma)$.

$$\begin{aligned}
\hat{u} &= 4.3331 + 0.7126 \cdot (x_1 - 2.067344815) + 4.3331 \cdot x_2 \\
&+\frac{1}{2} \cdot (-0.2275) \cdot (x_1 - 2.067344815) \cdot (x_1 - 2.067344815) \\
&+0.7126 \cdot x_2 \cdot (x_1 - 2.067344815) \\
&+\frac{1}{2} \cdot 4.3331 \cdot x_2^2
\end{aligned}$$

$$\begin{aligned}
\hat{x_1}' &= 2.0673 + 0.34 \cdot (x_1 - 2.067344815) + 2.0673 \cdot x_2 \\
&+\frac{1}{2} \cdot (-0.1085) \cdot (x_1 - 2.067344815) \cdot (x_1 - 2.067344815) \\
&+0.34 \cdot x_2 \cdot (x_1 - 2.067344815) \\
&+\frac{1}{2} \cdot 2.0673 \cdot x_2^2 \\
\hat{x_2}' &= 0.9 \cdot x_2 + \sigma \cdot \epsilon'
\end{aligned}$$

Approximationen der optimalen Wertefunktion

Zur Berechnung der indirekten bzw. direkten Taylor–Approximationen der optimalen Wertefunktion $\widehat{V}$ bzw. $\hat{V}_{dir}$ kommen die beiden Perturbationsroutinen `Vcoeff.m` bzw. `Vdirect.mw`, die in Abschnitt (4.3) vorgestellt wurden, zum Einsatz.
Deren Verwendung liefert die folgenden Taylor-Koeffizienten für die optimalen Wertefunktion $\widehat{V}$ und $\hat{V}_{dir}$, auf der unser Hauptinteresse liegt.

	$\widehat{V}$	$\hat{V}_{dir}$	$\lvert\hat{V}_{dir} - \hat{V}\rvert$
V_0	29.32568071	29.32568074	$0.2879 \cdot 10^{-7}$
V_{x1}	0.24292786	0.24292786	$0.0003 \cdot 10^{-7}$
V_{x2}	10.18693017	10.18693019	$0.2141 \cdot 10^{-7}$
$V_{x_1x_1}$	-0.11750718	$-.1175071817$	$0.007 \cdot 10^{-7}$
$V_{x_1x_2}$	0.00000000	-0.00000000	$0.0082 \cdot 10^{-7}$
$V_{x_2x_2}$	0.00000000	0.00000000	$0.0902 \cdot 10^{-7}$
$V_{\sigma\sigma}$	0.00000000	0.00000000	0.0

Tabelle 6.4: Unterschied der Koeffizienten von $\widehat{V}$ und $\hat{V}_{dir}$ bei Modell A

Die Abweichung der Taylor-Koeffizienten von $\hat{V}_{dir}$ zu denen von $\widehat{V}$ ist sehr gering, wie die Tabelle (6.4) zeigt, so dass gilt:

$$\hat{V}_{dir}(x) \approx \widehat{V}(x) \quad \forall x.$$

Somit liefern beide Routinen zur direkten bzw. indirekten Approximation der optimalen Wertefunktion trotz verschiedener Berechnungsarten approximativ dasselbe Taylorpolynom für die optimale Wertefunktion. Dieses lautet:

$$\begin{aligned} \widehat{V}(x,\sigma) = {} & 29.3257 + 0.2429(x_1 - 2.0673) + 10.1869x_2 + \\ & + \frac{1}{2} \cdot (-0.1175)(x_1 - 2.0673)x_2 \end{aligned} \tag{6.12}$$

Bemerkung 6.3. *Wie schon bei der Approximation der Kontrolle g fällt auf, dass der stochastische Parameter σ bei beiden Taylorreihen nicht auftritt. Die Wertefunktion zeigt sich von den Störungen bezüglich des technologischen Fortschritts x_2 unabhängig.*

Variation der Abschreibung δ

Die Antwort auf die Frage, wie sich die Koeffizienten der Taylor-Approximationen für $\widehat{V}$ bzw. $\hat{V}_{dir}$ bei Variation des Parameters δ ändern, wird nun gegeben. Mehrmaliges Aufrufen der MATLAB Routine `ModellA_run.m`, welcher einzig der Wert für δ übergeben wird, liefert als Rückgabe die folgenden Koeffizienten der Taylorreihe $\hat{V}_{dir}$. Diese entstammen der jeweiligen ASCII-Textdatei, die durch die MAPLE Routine `Vdirect.mw` von Seite 57 erzeugt wurde.

δ	V_0	V_{x_1}	V_{x_2}	$V_{x_1x_1}$	$V_{x_1x_2}$	$V_{x_2x_2}$	$V_{\sigma\sigma}$
0.0	67.9925	0.0351	6.8966	−0.0001	−0.0086	2.3783	0.0000
0.2	45.5596	0.1079	9.4366	−0.0041	−0.0409	1.1551	0.0000
0.4	38.6760	0.1522	9.8588	−0.0153	−0.0509	0.5949	0.0000
0.6	34.5564	0.1870	10.0325	−0.0362	−0.0454	0.3043	0.0000
0.8	31.6142	0.2167	10.1273	−0.0693	−0.0279	0.1242	0.0000
1.0	29.3257	0.2429	10.1869	−0.1175	−0.0000	−0.0000	0.0000

Tabelle 6.5: Taylor-Koeffizienten von $\hat{V}_{dir}$ für verschiedene δ bei Modell A

Interpretation:
Man erkennt, dass mit steigender Abschreibung der konstante Koeffizient V_0 der Taylorreihe, welcher dem optimalen Wert im Gleichgewicht entspricht, massiv abnimmt. Dies bedeutet, dass der Gesamtnutzen bzw. die Wohlfahrt niedriger ausfällt bei steigendem Kapitalverbrauch mittels Abschreibung. Dies ist intuitiv klar, denn dieses Phänomen trat bereits beim steady state in Bezug auf den Kapitalstock auf. Der durch das verringerte Einkommen resultierende Konsumrückgang führt

unmittelbar zu einem geringeren Nutzen für den Agenten Haushalt in allen Zeitpunkten und so stellt sich auf unendlichem Horizont betrachtet ein verminderter Zielfunktionswert ein. Diejenigen Koeffizienten wie $V_{x1}, V_{x_1x_1}$, welche nur x_1-Ableitungen enthalten, nehmen zudem betragsmäßig mit höherer Abschreibungsrate zu.
Die Abbildung (6.4) verdeutlicht das Anwachsen des Gesamtnutzens in der Volkswirtschaft, falls grundsätzlich weniger Kapital abgeschrieben wird. Die Graphen der optimalen Wertefunktionen liegen quasi gestapelt aufeinander und entfernen sich zunehmend von der (x_1, x_2)-Ebene.

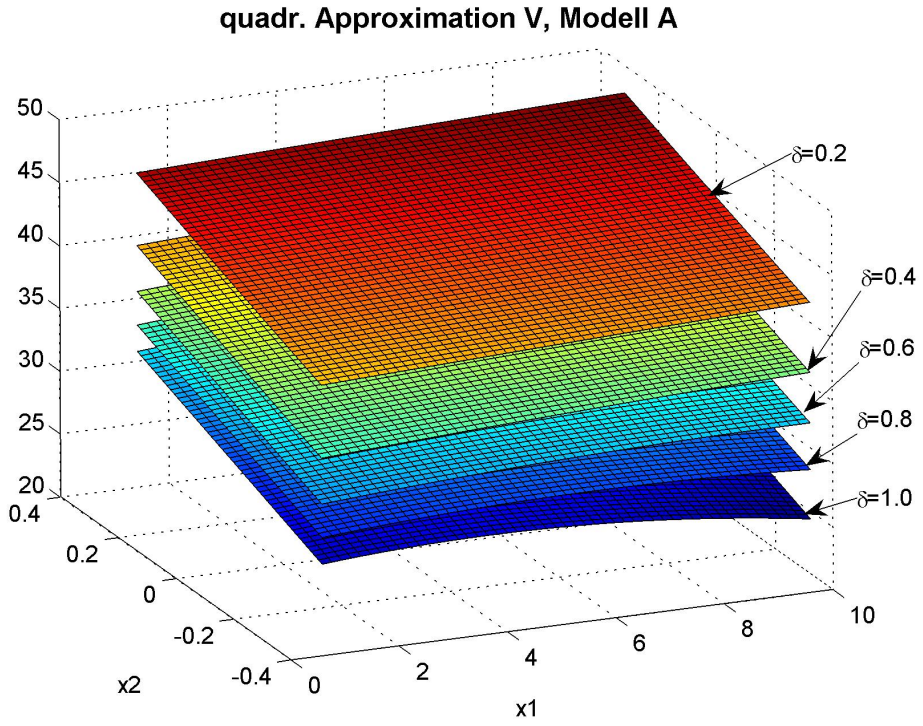

Abbildung 6.4: Direkte optimale Wertefunktionen $\hat{V}_{dir}$ für verschiedene δ bei Modell A

6.2.2 Modell B

Mit Hilfe der Perturbationsroutinen, die für Modell B im MATLAB Programm `ModellB_run.m`[4] sukzessiv aufgerufen werden, erhält man die folgenden Koeffizienten für die Approximationen der Dynamik h und der Kontrolle g:

Koeffizienten der linearen Terme:

$$g_{x_1} = 0.7126, \quad g_{x_2} = 4.7277$$

[4] Auf der CD-ROM enthalten.

$$h_{x_1} = \begin{pmatrix} 0.3400 \\ 0 \end{pmatrix}, \quad h_{x_2} = \begin{pmatrix} 1.6727 \\ 0.9000 \end{pmatrix}$$

Koeffizienten der quadratischen Terme für x:

$$g_{x_1x_2} = -0.2275, \quad g_{x_1x_2} = g_{x_2x_1} = 0.7775, \quad g_{x_2x_2} = 5.0563$$

$$h_{x_1x_1} = \begin{pmatrix} -0.1085 \\ 0 \end{pmatrix}, \quad h_{x_1x_2} = h_{x_2x_1} = \begin{pmatrix} 0.2751 \\ 0 \end{pmatrix} \quad h_{x_2x_2} = \begin{pmatrix} 1.3441 \\ 0 \end{pmatrix}$$

Koeffizienten der quadratischen Terme für σ:

$$g_{\sigma\sigma} = -7.5821, \quad h_{\sigma\sigma} = \begin{pmatrix} 7.5821 \\ 0 \end{pmatrix}$$

Die Terme $g_{\sigma\sigma}$ sowie $h_{\sigma\sigma}$ haben beträchtlich zugenommen. Dies belegt, dass der stochastische Einfluss, der über x_2 nun auch in die Zielfunktion eingeht, bei Modell B wesentlich größer ist.

Approximationen der optimalen Wertefunktion

Wir sind aber inbesondere wieder an den durch die Perturbationsmethode gewonnenen Approximationen der optimalen Wertefunktion interessiert.
Tabelle (6.6) listet die Koeffizienten von $\widehat{V}$ bzw. $\hat{V}_{dir}$ auf und zeigt, dass sich die Taylor-Koeffizienten

	$\widehat{V}$	$\hat{V}_{dir}$	$\lvert\hat{V}_{dir} - \hat{V}\rvert$
V_0	29.32568071	29.32568075	$0.3879 \cdot 10^{-7}$
V_{x1}	0.24292786	0.24292786	$0.0003 \cdot 10^{-7}$
V_{x2}	30.41153756	30.41153759	$0.3437 \cdot 10^{-7}$
$V_{x_1x_1}$	-0.11750718	-0.11750718	$0.0007 \cdot 10^{-7}$
$V_{x_1x_2}$	0.46373090	0.46373090	$0.0134 \cdot 10^{-7}$
$V_{x_2x_2}$	50.02247003	50.02247005	$0.1902 \cdot 10^{-7}$
$V_{\sigma\sigma}$	0.00000000	0.00000000	0.0

Tabelle 6.6: Unterschied der Koeffizienten von $\widehat{V}$ und $\hat{V}_{dir}$ bei Modell B

der direkten und indirekten Berechnung erneut nahezu gleichen.

Bemerkung 6.4. *Gegenüber Modell A hat sich die Hinzunahme von e^{x_2} als Term in die Zielfunktion spürbar auf die Koeffizienten V_{x2} sowie $V_{x_1x_2}, V_{x_2x_2}$ ausgewirkt, die wertmäßig stark zugenommen haben. Dies kann man damit deuten, dass für die Wohlfahrt bzw. den Gesamtnutzen der Volkswirtschaft im Falle von Modell B der technologische Fortschritt einen*

wesentlich größeren Einfluss besitzt und somit die entsprechenden Koeffizienten in $\widehat{V}$ *bzw.* $\hat{V}_{dir}$ *neu justiert werden.*

An dieser Stelle wird angemerkt, dass bei der Abhandlung dieses Modells in der Arbeit von [19], die bereits zitiert wurde, vom Wert abweichende Taylor–Koeffizienten errechnet wurden (vgl. [19] S. 73). Dort wurden für Modell B bei gleicher Parametersetzung mittels eines MAPLE Programms [5] die folgenden Koeffizienten ermittelt:

$$\begin{array}{lll} V_0 = 29.32566, & V_{x_1} = 0.24291, & V_{x_2} = 28.38888 \\ V_{x_1x_1} = -0.11749, & V_{x_1x_2} = V_{x_2x_1} = 0.41735 & \\ V_{x_2x_2} = 42.71896, & V_{\sigma\sigma} = -0.15508 \cdot 10^{-2} & \end{array}$$

Die unterschiedlichen Werte resultieren aus einem Tippfehler im Programm selbst. Dort wurde als Argument der Kostenfunktion `r` innerhalb der Bellman–Gleichung nicht der aktuelle Zustand `(x[1],x[2])`, sondern der nachfolgende Zustand `(h[1],h[2])` der Dynamik $h(x, \sigma)$ eingesetzt. Die fehlerbehaftete Zeile in der Routine lautet wie folgt:

Listing 6.1: Auszug aus der Maple Datei `compute_Vcoeff.mw`

```
(...)
Ayy := coeftayl(r(g,h[1],h[2])+expand(beta*Vh)-expand(V),
[x[1],x[2],sigma] = [0,0,0],[i1,i2,i3]):
(...)
```

Bei Modell A blieb dieser Fehler ohne bemerkbare Auswirkungen, da die Zielfunktion von x unabhängig gegeben ist als $l(x, u) := \log(u)$.

Variation der Abschreibung δ

Auch für Modell B, in welchem der stochastische Einfluss mehr gewichtet ist und somit die wirtschaftliche Unsicherheit steigt, wurden für verschiedene Abschreibungsraten die Approximationen $\hat{V}_{dir}$ berrechnet. Wird δ aus der Menge $\{0.2, 0.4, 0.6, 0.8, 1\}$ möglicher Abschreibungsraten gewählt, so ergibt der wiederholte Aufruf von `ModellB_run.m` die Koeffizienten von Tabelle (6.7).

Allen Graphen der optimalen Wertefunktion in Figur (6.5) ist die stärkere Krümmung in $x2$-Richtung gemeinsam, wie ein Vergleich mit Abbildung (6.4) zeigt. Ein positiver technologischer Fortschritt wirkt sich günstig auf die ökonomische Lage aus und führt zu einem Anwachsen der Wohlfahrt. Erneut gilt der Grundsatz, dass mit fallender Abschreibungsrate δ ein höheres Einkommen erzielt wird und somit eine höhere Wohlfahrt, ausgedrückt durch den optimalen Wert von $\hat{V}_{dir}$, erreicht wird.

[5] Das Programm `compute_Vcoeff.mw` ist deswegen auf der CD-ROM enthalten.

δ	V_0	V_{x_1}	V_{x_2}	$V_{x_1x_1}$	$V_{x_1x_2}$	$V_{x_2x_2}$	$V_{\sigma\sigma}$
0.0	67.9925	0.0351	53.7879	−0.0001	0.0275	76.6914	0.0000
0.2	45.5596	0.1079	40.8570	−0.0041	0.1291	61.7166	0.0000
0.4	38.6760	0.1522	36.5319	−0.0153	0.2153	56.9651	0.0000
0.6	34.5564	0.1870	33.8645	−0.0362	0.2972	53.9844	0.0000
0.8	31.6142	0.2167	31.9302	−0.0693	0.3792	51.7806	0.0000
1.0	29.3257	0.2429	30.4115	−0.1175	0.4637	50.0225	0.0000

Tabelle 6.7: Taylor-Koeffizienten von $\hat{V}_{dir}$ für verschiedene δ bei Modell B

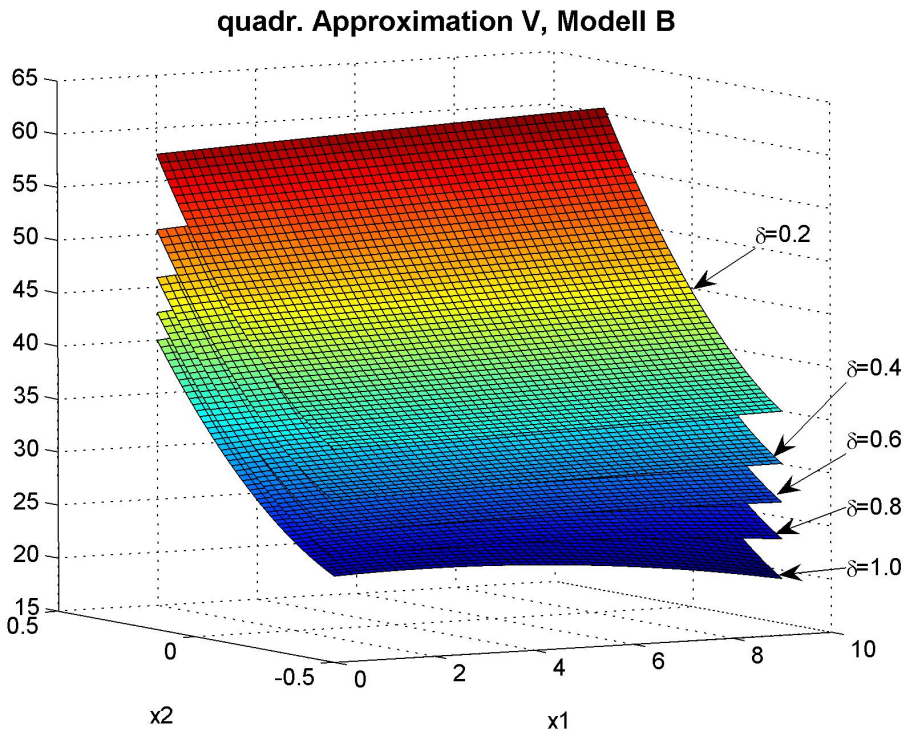

Abbildung 6.5: Direkte optimale Wertefunktionen $\hat{V}_{dir}$ für verschiedene δ bei Modell B

6.3 Auswertung und Analyse der Modelle

Ein Vergleich der im vorherigen Abschnitt ermittelten Perturbationslösungen $(\hat{V}_{dir}, \widehat{V})$ mit den Lösungen der dynamischen Programmierung, welche der Algorithmus (3.31) liefert, wird in diesem Abschnitt vorgenommen. Grundlage der nachfolgenden Abbildungen und sämtlicher Zahlenvergleiche bildet die MATLAB Routine `evaluation.m`. [6]
Der Aufruf der Routine lautet: `evaluation(model,domain)` wobei die Parameter nachstehende Bedeutung besitzen.

model	Modellwahl: **1**: Modell A gemäß (6.1) , **2**: Modell B gem. (6.3)
domain	Größe des Auswertungsbereichs Γ: **1**: $[1, 10] \times [-0.32, 0.32]$, **2**: $[1, 4] \times [-0.32, 0.32]$

Innerhalb der Auswerteroutine `evaluation` werden die Werte der optimalen Wertefunktion $\tilde{V}$ automatisch von ASCII-Dateien eingelesen. Diese lauten je nach Modell und Wahl des Rechengebiets wie folgt:

model	**domain**	**ASCII-Datei**
1	1	`Values_delta_1.00_kappa_0.asc`
1	2	`Values_delta_1.00_kappa_0_small.asc`
2	1	`Values_delta_1.00_kappa_2.asc`
2	2	`Values_delta_1.00_kappa_2_small.asc`

Tabelle 6.8: Dateinamen der gespeicherten Werte von $\tilde{V}$ (DP)

Diese vier Dateien müssen sich beim Aufrufen von `evaluation.m` im selben Ordner befinden wie das MATLAB Programm selbst.

Implementierung der DP Methode in `DPalg.c`

Das C Programm `DPalg.c` implementiert unter Berücksichtigung der in Kapitel fünf erwähnten Aspekte den Algorithmus (3.31) zur Berechnung der optimalen Wertefunktion. Wie das Programm, welches auf die beiden C Dateien `zufallszahlen.c` sowie auf die Gitterstruktur `gridgen.c` angewiesen ist, zu kompilieren bzw. zu linken ist, kann der Readme-Datei `readme.txt`auf der beigelegten CD-ROM entnommen werden.

Für die Ausführung der DP Methode (3.31) sind einige Parameter bzw. Werte festzulegen.
Dies erfolgt im Quellcode an den folgenden, kommentierten Stellen.

[6] Routine befindet sich im Anhang (B.1), siehe S. 119.

Listing 6.2: Parameter des DP Algorithmus in `DPalg.c`

```
18 (...)
19
20 /*Sämtliche Modellparameter werden in einer globalen
21 Strukturvariable p zusammengefasst*/
22 struct parameters
23         {
24         double beta; /*Diskontierungsfaktor */
25         double delta; /*Abschreibung*/
26    (...)
27         }p ={0.95, 1.0, 5., 0.34, 0.9, 0.008, 0};
28
29 /*Parameter fuer adaptive Gitter*/
30
31 //#define Refine  /*Schalter zur Adaption (entkommentieren)*/
32 #define DIFF 0.00001  /*Differenz Knotenwerte bei regelmaessigem Gitter*/
33 #define RTOL  0.001      /* Verfeinerungstoleranz */
34 #define RHO  0.9  /* Sicherheitsfaktor fuer Verfeinerung! */
35
36 #define U_SECTIONS 160;  /*Diskretisierung der Kontrollmenge*/
37 const unsigned int MAX_ITER = 150;  /*Maximalanzahl Iterationen*/
38
39 (...)
```

Wird das erstellte Programm `DPalg.exe` erfolgreich ausgeführt, so werden mehrere Dateien, teils zu statistischen Zwecken, andererseits für eine Weiterbearbeitung in MATLAB erstellt.
Dies sind folgende Dateien, welche sich im selben Ordner wie die `.exe` Datei befinden.

- `Values_delta_□_kappa_□.asc`
  (dreispaltige Datei mit den gespeicherten Knotenwerten der Funktion $\tilde{V}$)

- `Grid_delta_□_kappa_□_RTOL_□_RHO_□.asc`
  (abgespeichertes Gitter im Standardformat)

- `Stat_delta_□_kappa_□_RTOL_□_RHO_□.dat`
  (Statistikdatei mit Angaben zu Gittergröße, Knotenanzahl, Rechenzeit usw.)

(□ dient als Platzhalter für den jeweiligen Wert des Parameters)

Für die Zwecke der Auswertung mit `evaluation.m` sind, wie eingangs erwähnt, die ASCII-Dateien mit den abgespeicherten Knotenwerte $\tilde{V}(E_i)$ relevant.

6.3.1 Modell A

Für das Modell A existiert bekanntlich eine analytische Lösung (vgl. (6.10)), daher erfolgt eine Fehlerermittlung für die numerische Approximation $\hat{V}_{dir}$ der direkten Perturbationsmethode und für $\tilde{V}$, die optimale Wertefunktion der dynamischen Werteiteration. Nachfolgende Ergebnisse bzw. Abbildungen sind Folge des Aufrufs von `evaluation(1,1)` bzw. `evaluation(1,2)`.

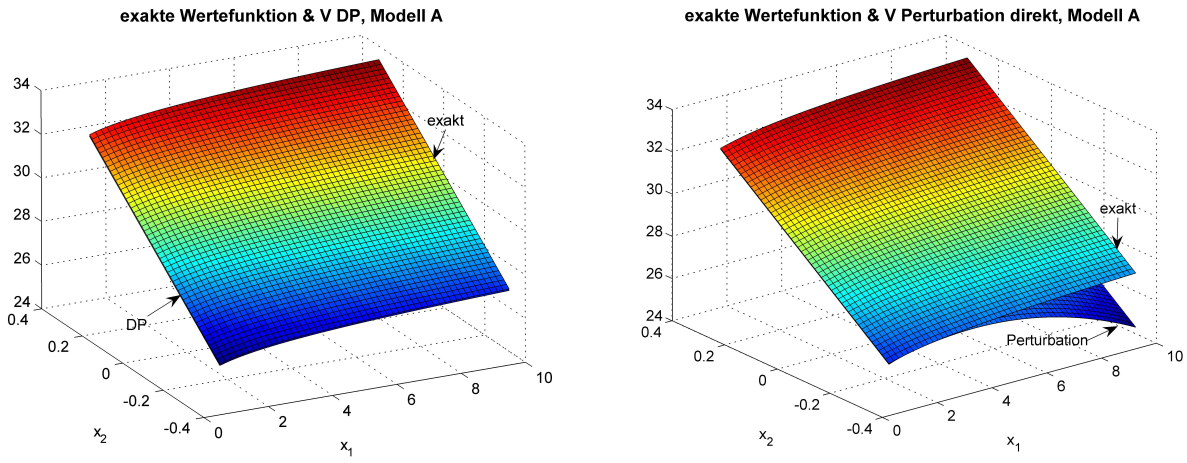

Abbildung 6.6: Exakte Lösung V_∞ und DP Lösung $\tilde{V}$ (links), V_∞ und $\hat{V}_{dir}$ (rechts)

Die DP Methode $\tilde{V}$ kommt der exakten Wertefunktion V_∞ auf dem gesamten, großen Zustandsbereich (`domain=1`) sehr nahe. Der Graph der exakten Lösung liegt nur minimal über dem der affin bilinearen Approximation $\tilde{V}$. Das Krümmungsverhalten beider Funktionen ist identisch.

Hingegen erkennt man in der rechten Figur von (6.6) deutlich die Unterschiede im Verlauf beider Graphen. Die Perturbationslösung $\hat{V}_{dir}$ verläuft auf ganz Γ unterhalb der exakten Wertefunktion, jedoch weichen mit größer werdendem Abstand des Zustands x vom Gleichgewicht $\bar{x} \approx (2.0673, 0)$ die Werte $V_\infty(x)$ und $\hat{V}_{dir}(x)$ erheblich voneinander ab.
Die Näherungslösung $\hat{V}_{dir}$ ist im Bereich $x_1 \in [6; 10]$ stärker gekrümmt als die exakte Lösung.

Die nachfolgende Abbildung (6.7) mit den drei Figuren belegt diese Aussagen. Während der absolute Fehler $|\tilde{V} - V_\infty|$ der dynamischen Programmierung selbst auf dem großen Gebiet für kein x den Wert 0.07 übersteigt, wächst, wie der rechten Figur gut zu entnehmen ist, der Fehler $|\hat{V}_{dir} - V_\infty|$ mit zunehmendem Kapitalstock x_1 stark an.
Für Zustände mit $x_1 = 10$ am Rande des Bereichs beträgt dieser mehr als 2.5.

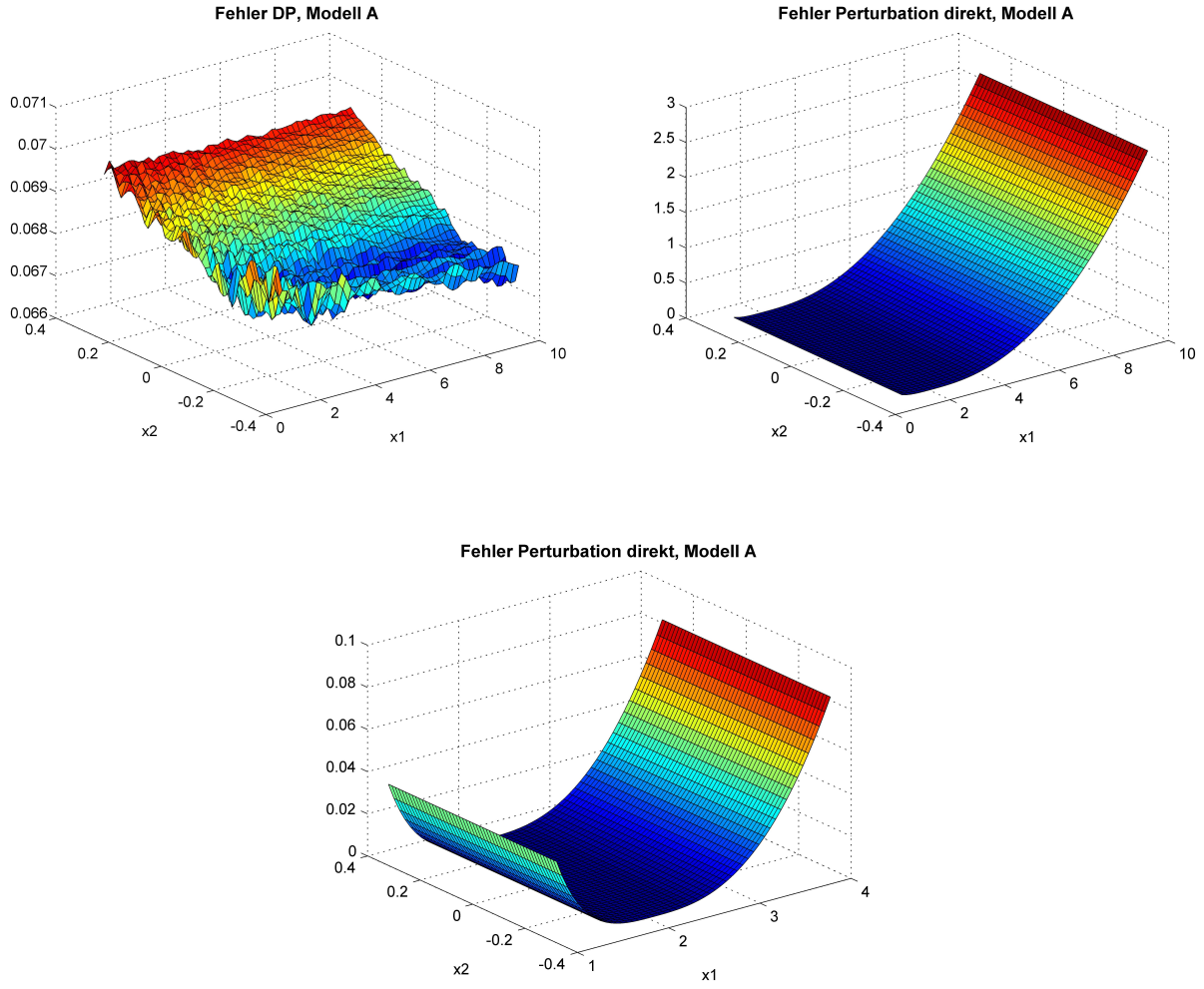

Abbildung 6.7: Absoluter Fehler $|\tilde{V} - V_\infty|$ (oben links) sowie $|\hat{V}_{dir} - V_\infty|$ (oben rechts) auf großem Bereich, absoluter Fehler $|\hat{V}_{dir} - V_\infty|$ auf kleinerem Bereich (unten)

Daran erkennt man anschaulich die wesentlich geringere Genauigkeit der Perturbationslösung $\hat{V}_{dir}$, wenn man am optimalen Wert für Zustände x, die weiter entfernt von $\bar{x}$ liegen, interessiert ist. Selbst auf dem kleineren Zustandsbereich $[1, 4] \times [-0.32, 0.32]$ (Aufruf entsprechend `evaluation(1,2)`) nimmt der Fehler zu, sobald man sich vom steady state entfernt (vgl. untere Figur (6.7)).
Selbst auf diesem Gitter treten Fehler vom Betrag größer als 0.07 auf. Dieses Fehlerausmaß wurde bei der DP Methode $\tilde{V}$ selbst auf dem großen Gitter nicht überschritten.

Die DP Approximation $\tilde{V}$ wird noch exakter, wenn man die Werteiteration (3.31) auf adaptiven

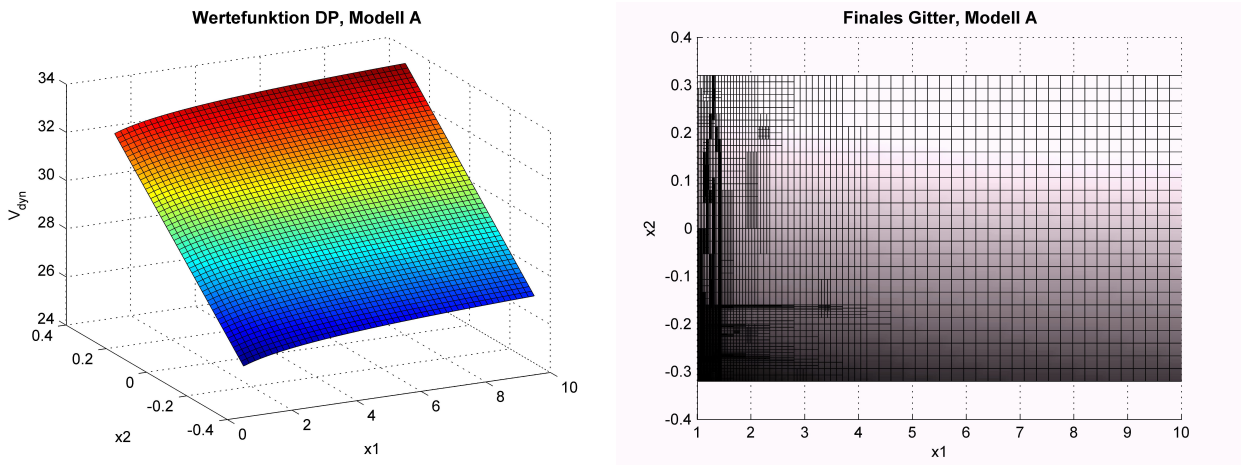

Abbildung 6.8: Optimale Wertefunktion $\tilde{V}$ und finales adaptives Gitter Γ_{98}

Gittern durchführt. Inbesondere sind diese bei gekrümmter Wertefunktion vorteilhaft. Die Abbildung (6.8) zeigt die auf dem großen Gitter $[1, 10] \times [-0.32, 0.32]$ angenäherte Wertefunktion, deren Verlauf sich von dem Graphen der linken Figur in (6.6) kaum unterscheidet, und das finale adaptive Gitter Γ_{98} nach 98 Iterationen.
Dabei wurden die Parameter wie folgt gesetzt:

$rtol : 0.001$	$\rho : 0.8$

Das Startgitter wurde mit 273 Knoten besetzt, das Endgitter enthält dann nach Durchlauf der Verfeinerungsschritte am Ende 8081 Eckpunkte.
Wie die Graphik zeigt, wurden besonders diejenigen Rechtecke nahe der $x_1 = 1$ - Achse verfeinert, da dort die Funktion am stärksten gekrümmt ist.

Für eine detailliertere Analyse des Modells, beispielsweise des Verhaltens der Lösungen (mit Ausnahme von $\hat{V}_{dir}$) auf mehreren, unterschiedlichen Gittergrößen, kann auf [3] verwiesen werden.

6.3.2 Modell B

Für Modell B werden nun die direkte Perturbationslösung $\hat{V}_{dir}$ und die Wertefunktion der DP Methode $\tilde{V}$ miteinander verglichen. Hierfür wurde exemplarisch `evaluation(2,1)` aufgerufen. Da eine analytische Lösung fehlt, lassen sich nur die beiden Approximationen auf dem gewählten Bereich miteinander vergleichen.

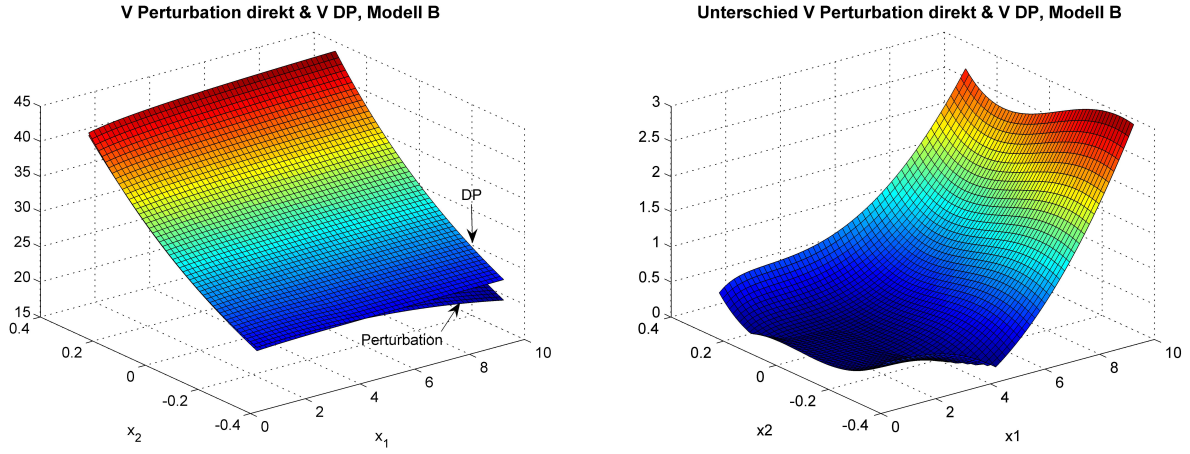

Abbildung 6.9: Wertefunktion $\tilde{V}$ und $\hat{V}_{dir}$(links), Unterschied $|\tilde{V} - \hat{V}_{dir}|$ (rechts)

Der Vergleich der optimalen Wertefunktion der direkten Perturbationsmethode mit der DP Methode zeigt qualitativ die selben Ergebnisse wie beim Modell A.
Die Differenz zwischen beiden Approximationen nimmt auf dem ausgewerteten, größeren Bereich $[1, 10] \times [-0.32, 0.32]$ mit zunehmendem Abstand vom Gleichgewicht $\bar{x}$ zu, was durch den Genauigkeitsverlust bei der Taylor-Approximation $\hat{V}_{dir}$, die nur nahe des Gleichgewichts akkurat ist, zu erklären ist.
Die Abbildung (6.9) belegt dieses Verhalten anschaulich. Am rechten Rand des Gitters, für $x_1 \to 10$, weichen beide Näherungslösungen für V_∞ sogar um fast 3 Einheiten voneinander ab.

6.4 Numerische Simulationen

Eine interessante numerische Anwendung ist die Simulation der Modelle. Numerische Simulation ist eine gebräuchliche Vorgehensweise zur Analyse von dynamischen Systemen, denen die hier betrachteten zeitdiskreten Kontrollsysteme zuzurechnen sind.

Mit Hilfe der in C implementierten Funktion `simulation()`, die sich im Hauptprogramm `Dpalg.c` befindet und als sogenanter Simulator dient, lässt sich ausgehend von einem übergebenen Anfangswert $x_0 \in \mathbb{R}^2$ die Lösung X_t als auch die Kontrolle u_t simulieren.
Um die Kontrolle u zu simulieren, kann im Fall der beiden Modelle A und B entweder die bekannte Taylorapproximation $g(x, \sigma)$ der Perturbationsmethode oder das online gemäß (3.32) ermittelte $\tilde{u}$ vom Optimalitätsprinzip verwendet werden.
Die in der Dynamik eingehenden stochastischen Prozesse in Form der normalverteilten Zufallsvariablen z_t werden mithilfe des in Kapitel fünf vorgestellten Zufallszahlengenerators nach Marsaglia (siehe S. 63) nachgebildet. Mit einer entsprechend grafischen Aufbereitung der erhaltenen Ergebnisse lassen sich durchaus Rückschlüsse auf die dynamische Entwicklung der ökonomischen Größen beider Modelle ziehen.
Die Simulationsroutine ist wie folgt definiert:

```
double simulation(double *x0, qgrid *g,long * idum, char * file, int method)
```

Funktionsweise:
Simulation der Trajektorie und der Kontrolle sowie Berechnung des Zielfunktionswerts $J(x_0)$.
Export der numerischen Werte in eine ASCII-Datei (Endung `.asc`).
Textdatei wird in vier Spalten (x_1-Wert, x_2-Wert, Störung z, Kontrolle u) gegliedert für eine spätere Aufbereitung in MATLAB.

Eingabeparameter:

$x0$	gewünschter Anfangswert des Systems
g	Zeiger auf das Gitter
$idum$	Initialisierungswert für den Zufallszahlengenerator
$file$	Zeichenkette mit dem vorgesehenen Dateinamen der ASCII-Datei
$method$	Parameter für Auswahl der Approximation für u(1: DP, 2: Perturbationsmethode)

Rückgabe:
Wert des Zielfunktionals $J(x_0, u)$ nach „hinreichend“ vielen Zeitschritten

Die in der ASCII-Textdatei gespeicherten Werte der jeweils ersten Simulation werden in MATLAB entsprechend eingelesen und anschließend in einer graphischen Animation umgesetzt.
Das Einlesen der Daten und Erstellen der nachfolgenden Abbildungen erledigt die MATLAB Routine `simplot2d.m`, deren Code auf Seite 126 im Anhang zu finden ist.

6.4.1 Modell A

Die numerischen Simulationen werden für verschiedene Anfangswerte, die bei der Ausführung des Programms von `DPalg.c` im Anschluss an die Berechnung der optimalen Wertefunktion eingegeben werden können, durchgeführt. Die Kontrolle u wird in jedem Zeitschritt t als optimale approximative Kontrolle wie in (3.32) gewählt. Die resultierenden Werte für den Zustand x sowie für die Steuerung u werden in ASCII-Textdateien mit dem Dateinamen

`Simu_delta_□_kappa_□_•.asc`

gespeichert, wobei □ als Platzhalter für die festgesetzten Werte steht.
Für • steht die entsprechende arabische Nummer i für die i.te Simulation. Die dynamische Untersuchung der Zustands- und Kontrollgrößen endet, wenn sich der durch Summation ergebende Zielfunktionswert $J(x_0) = \sum_{t=0}^{\infty} \beta^t \log(u_t)$ in einem Zeitschritt nicht mehr wesentlich ändert. Der letzte Simulationszeitpunkt t_{end} wurde im C-Programm gewählt als $t_{end} := \operatorname{argmin}\{\beta^t | \beta^t \geq 10^{-3}\}$.

Für die Durchführung der Simulation wurde die Werteiteration der dynamischen Programmierung auf dem Bereich $\Gamma = [1; 10] \times [-0.32; 0.32]$ ohne Gitteradaption ausgeführt. Als $RTOL$ wurde 10^{-3} festgesetzt und als Zellenanzahl wurde 2500 gewählt, d.h. es ergeben sich 2601 Knotenpunkte, für welche iterativ der DP Operator gemäß (3.24) ausgewertet wird. Der Algorithmus benötigte 144 Iterationen, um mit der gewünschten Fixpunktgenauigkeit die numerische, optimale Wertefunktion $\tilde{V}$ zu berechnen. Die Simulationen erfolgen dann auf dem Gitter g mit den abgespeicherten Knotenwerten der bilinearen Approximation der optimalen Wertefunktion. Als Anfangswert x_0 können selbstverständlich nur Werte innerhalb des Gitters Γ gewählt werden. Der simulierte Wert $\bar{J}(x_0)$ entspricht dem Mittelwert einer Serie von 1000 stochastischen Simulationen. Die konstante Anzahl an Auswertungen, von welcher auch die Rechenzeit abhängt, kann nach Wunsch in folgender Quellcodezeile des Hauptprogramms `DPalg.c` abgeändert werden:

Listing 6.3: Anzahl Simulationen

```
41 const unsigned int anz_sim = 1000; /*Anzahl an Simulationen*/
```

Die exportierten ASCII-Dateien, welche durch die Simulationsroutine `simulation` erzeugt werden, lauten bei dieser Wahl der Parameter:

`Simu_delta_1.00_kappa_0_•.asc,` $\bullet \in \{1, \ldots, 5\}$

Bei Ausführung des Programms ergeben sich die Werte von Tabelle (6.9).

$\mathbf{x_0}$	Sim. $\bar{J}(x_0)$ (u DP)	Sim. $\bar{J}(x_0)$ (u Pert.)	$V_\infty(x_0)$	DP $\tilde{V}(x_0)$	Pert. $\hat{V}(x_0)$
$(1.0, 0.2)$	30.9716	30.9609	30.9983	30.9291	31.0368
$(2.1, 0.0)$	29.3130	29.3178	29.3336	29.2652	29.3336
$(4.5, 0.3)$	32.7350	32.7492	32.7724	32.7029	32.6250
$(7.1, -0.1)$	28.8898	28.7873	28.9266	28.8592	28.0415
$(9.0, -0.25)$	27.5000	26.5685	27.5177	27.4499	25.6393

Tabelle 6.9: Vergleich der simulierten Werte $\bar{J}(x_0)$ mit exakter, dynamischer und Perturbationslösung

Man erkennt, dass für alle Startzustände der simulierte Wert $\bar{J}(x_0)$ mit dem u von der dynamischen Programmierung und der approximierte Gitterwert $\tilde{V}(x_0)$ eng beieinander liegen und deren maximale Differenz ca. $5.01 \cdot 10^{-2}$ beträgt. Beide Werte weichen selbst auf dem gleichmäßigen Gitter unabhängig von der Position von x_0 nur mit einer konstanten Größenordnung von 10^{-2} von der exakten Lösung $V_\infty(x_0)$ ab. Dagegen unterscheidet sich der Wert $\hat{V}(x_0)$, der mit der Perturbationsmethode gewonnen wird, für alle Anfangswerte x_0, die nicht in der Nähe des Gleichgewichts $(\bar{x}_1, \bar{x}_2, \bar{u}) = (2.0673, 0.0, 4.3331)$ liegen, signifikant von der exakten Lösung. Für die fünfte Simulation mit $x_0 = (9, -0.25)$ beträgt der Fehler nahezu 1.9 . Nur für $x_0 = (2.1, 0.0)$ ist der Wert des Taylorpolynoms $\hat{V}(x_0)$ sehr genau.
Auffällig ist weiterhin, dass bei Wahl von $u_t = g(x_t, \sigma)$ mit g aus der Perturbationsmethode der simulierte Wert $\bar{J}(x_0)$ für alle Anfangswerte mit Ausnahme des zweiten näher am exakten optimalen Wert liegt als $\hat{V}$ selbst. Die Ursache hierfür ist, dass sich das System bereits nach wenigen Zeitschritten nahe des Gleichgewichtspunkts befindet und für diese Zustände die Approximationslösung g verlässliche Kontrollwerte liefert. Die Funktion g des Perturbationsmodells wurde ja gerade im steady state nach Taylor entwickelt.

Nachfolgend sind einige typische Trajektorienverläufe grafisch veranschaulicht. In Abbildung (6.10) wurde ein Startwert gewählt, der sich dem steady state $(\bar{x}_1, \bar{x}_2, \bar{u})$ bezüglich x_1 von unten, bezüglich dem Zustand x_2 von oben nähert.
Abbildung (6.11) zeigt den Verlauf der ökonomischen Größen, wenn $x_0 = (2.1, 0)$, also nahe des Gleichgewichts gewählt wird. Hier oszilliert die Lösung den gesamten Zeithorizont über um den Gleichgewichtszustand, ausgelöst durch die stochastischen Schocks, die eine kleine, aber spürbare Veränderung der Variable x_2 nach sich ziehen. Die letzte Figur (6.12) resultiert aus der Wahl eines hohen Anfangskapitals von 9.0 und eines negativen technologischen Fortschritts in $t = 0$ von -0.25. Diese Abbildung zeigt am besten, wie sich die Lösung rasch unmittelbar nahe des Gleichgewichts einpendelt. Allen Trajektorien ist gemeinsam, dass dass sich die Größen nach endlicher Zeit in der Umgebung des Gleichgewichtspunkts aufhalten.

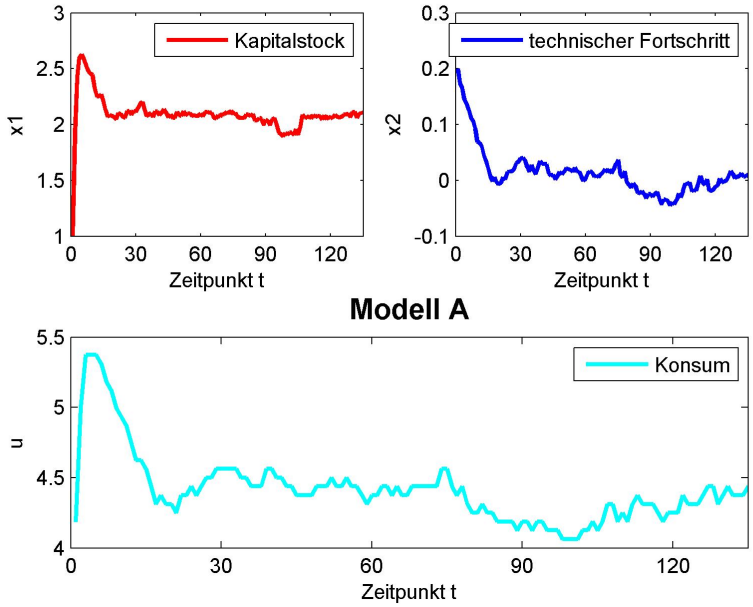

Abbildung 6.10: Simulation für Anfangswert $x_0 = (1.0, 0.2)$ - Modell A

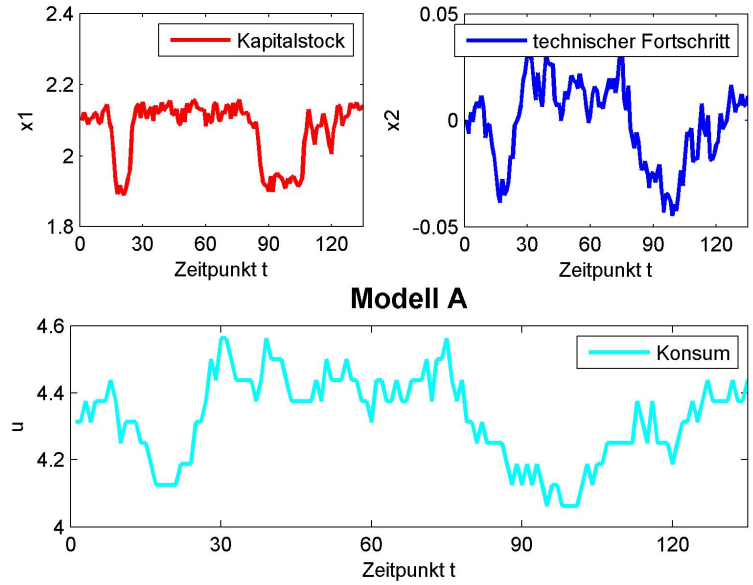

Abbildung 6.11: Simulation für Anfangswert $x_0 = (2.1, 0.0)$ - Modell A

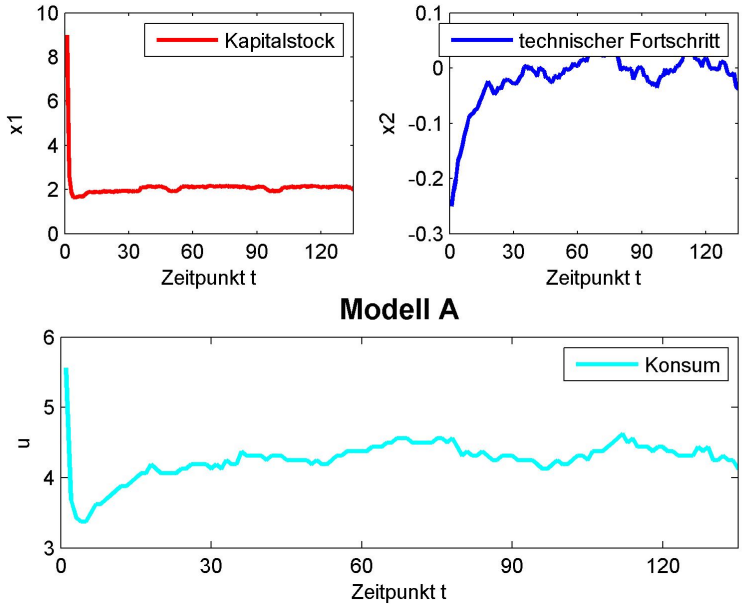

Abbildung 6.12: Simulation für Anfangswert $x_0 = (9.0, -0.25)$ - Modell A

6.4.2 Modell B

Auch für das Modell B, für dessen optimale Wertefunktion V_∞ keine analytische Lösung bekannt ist, wollen wir eine stochastische Computersimulation als nützliches, alternatives Analyse-Tool nutzen.
Wegen dem erwähnten Fehlen einer analytischen Lösung werden diesmal der simulierte Wert für beide Approximationen von u, der Wert aus der indirekten Perturbationsmethode sowie der optimale Wert, der sich mittels der Werteiteration der dynamischen Programmierung ergibt, für mehrere, ausgewählte Startzustände x_0 miteinander verglichen.
Wie in Modell A wurde der simulierte Wert $\bar{J}(x_0)$ als arithmetisches Mittel von 1000 Simulationen gebildet. Das Gitter Γ diskretisiert den Zustandsbereich $[1; 10] \times [-0.32; 0.32]$ mit 2601 Knotenpunkten. Im Anschluss an die 144 Iterationen zur Bestimmung von $\tilde{V}$ wurden im Programm die in (6.10) aufgelisteten Anfangswerte eingegeben und die Simulationsroutine erstellte die folgenden Dateien:

`Simu_delta_1.00_kappa_2_•.asc,` $\quad \bullet \in \{1, \ldots, 8\}$

Die folgende Tabelle (6.10) zeigt die numerischen Ergebnisse bei Wahl verschiedener Anfangswerte $x_0 \in [1; 10] \times [-0.32; 0.32]$.

$\mathbf{x_0}$	Sim. $\bar{J}(x_0)$ (u DP)	Sim. $\bar{J}(x_0)$ (u Pert.)	DP $\tilde{V}(x_0)$	Pert. $\hat{V}(x_0)$
$(1.0, 0.2)$	36.0223	35.9863	35.9856	35.9832
$(2.1, 0.0)$	29.3629	29.3159	29.3081	29.3336
$(4.5, 0.3)$	41.8507	41.9275	41.8502	41.2818
$(5.4, 0.15)$	35.1314	35.1702	35.1247	34.8390
$(7.1, -0.1)$	27.0668	26.9520	27.0076	26.0357
$(8.25, 0.0)$	30.0436	29.7908	29.9955	28.5818
$(9.0, -0.25)$	23.5478	22.9490	23.5077	21.3426
$(10.0, 0.3)$	42.5607	41.9819	42.5625	40.0336

Tabelle 6.10: Vergleich der simulierten Werte $\bar{J}(x_0)$ mit dynamischer und Perturbationslösung

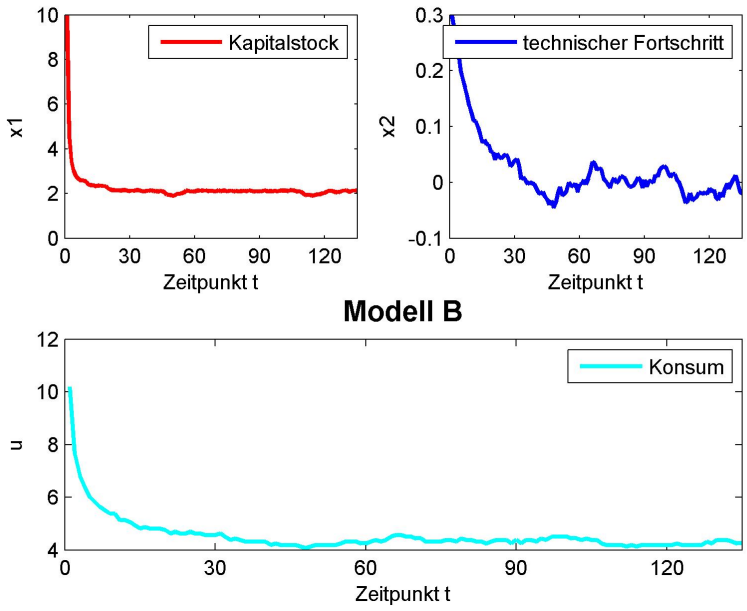

Abbildung 6.13: Simulation für Anfangswert $x_0 = (10.0, 0.3)$ - Modell B

Da sich die bei den Abbildungen von Modell A beobachteten Verläufe auch bei Modell B ergeben, wird nur eine Abbildung, nämlich die zum Anfangswert $(10.0, 0.3)$ am Rande von Γ dargestellt. Der Verlauf der Trajektorie, welche sich dem volkswirtschaftlichen Gleichgewicht nähert, ändert sich im Langzeitverhalten nicht. Die Abweichung des Werts $\hat{V}(x_0)$ aus der Perturbationsmethode gegenüber dem simulierten Wert ist umso größer, je weiter x_0 vom steady state entfernt gewählt wurde. Dies lässt auf zunehmende Ungenauigkeit der Perturbationslösung mit Entfernung vom Gleichgewichtspunkt schließen und bekräftigt die Aussagen von Abschnitt (6.3).
Zudem liegt der simulierte Wert $\bar{J}(x_0)$, der sich ergibt, wenn u durch g errechnet wurde, stets näher am Wert der bilinearen Funktion $\tilde{V}$ als die Perturbationslösung $\hat{V}$. Der Grund hierfür ist derselbe wie bei den Simulationen von Modell A.

7 Fazit und Ausblick

Die Analyse der beiden Wachstumsmodelle A und B, die an Ramseys Referenzmodell angelehnt sind, hat im Rahmen dieser Arbeit Vor- und Nachteile der verwendeten Lösungsmethoden aufgezeigt. Die genutzten Perturbationsmethoden lieferten sowohl für die Steuerung u des Kontrollsystems als auch die optimale Wertefunktion V_∞ Approximationen in Form von Taylorpolynomen. Diese erzielten in der Nähe des Gleichgewichts sehr genaue und verlässliche Ergebnisse. Mit zunehmender Entfernung vom steady state nahmen die Fehler signifikant zu.
Kritisch bei dieser Lösungstechnik ist, dass deren Anwendung an die Existenz eines Gleichgewichts gebunden ist, welches in unseren Anwendungsbeispielen über die Gleichgewichtsbedingungen, die sich aus dem stochastischen Maximumprinzip herleiteten, berechenbar ist. Der steady state, der Entwicklungspunkt der Taylor-Approximationen ist, kann in anderen Modellen nur schwierig oder gar nicht berechnet werden. Die aufgrund der Taylor-Entwicklungen auf eine Vielzahl an Ableitungen erster und zweiter Ordnung angewiesene Vorgehensweise macht symbolische Berechnungen nötig, die möglichst exakt ausgeführt werden müssen. Mit Programmpaketen wie MAPLE oder MATLAB's Symbolic Math Toolbox ist diese Aufgabe auf Rechnern aber relativ leicht zu bewältigen. Die Herleitung der Funktion f der Gleichgewichtsbedingungen erfordert jedoch eine Rechnung per Hand, da hierfür eine Implementierung als Quellcode noch nicht realisiert ist.

Dagegen kennzeichnet den Algorithmus der dynamischen Programmierung, der eine numerische Approximation der Wertefunktion und der Steuerung in Form einer affin bilinearen Gitterfunktion liefert, eine wesentlich höhere Rechenkomplexität. Für Modelle niedriger Dimension ($n \leq 2$), so wie es die beiden Anwendungsbeispiele A und B sind, ist der Anstieg der Rechenzeit noch akzeptabel. Diese lag für das Programm `DPalg.c`, welches den Algorithmus implementiert, im Bereich einiger Minuten. Für höherdimensionale Modelle mit mehreren Zustandsvariablen allerdings ist die Werteiteration aufgrund der immens großen Anzahl an Gitterpunkten sehr schwer durchzuführen, hier stoßen selbst die adaptiven Gitter an ihre Grenzen („curse of dimensionality“).
Ein großer Vorteil der Werteiteration am Optimalitätsprinzip wurde bei der Auswertung ersichtlich. Die Methode der dynamischen Programmierung lieferte für die optimale Wertefunktion auf den betrachteten Gebieten wesentlich niedrigere Approximationsfehler von einer konstanten Größenordnung. Beim Modell A konnte dies dank der Existenz einer analytischen Lösung für V_∞

explizit festgestellt werden, bei Modell B durch einen Vergleich der simulierten Werte mit den Werten der Perturbationsmethoden und der DP Methode gefolgert werden. Nur unmittelbar nahe des Gleichgewichts konnte die Perturbationsmethode eine höhere Genauigkeit erzielen, was in der Natur der endlichen Taylorreihe, deren Entwicklungspunkt gerade der steady state ist, liegt.
Zu erwähnen ist, dass die Genauigkeit der Approximationen bei der Werteiteration noch weiter erhöht werden kann, wenn beispielsweise eine Vergröberung in den Algorithmus aufgenommen wird, alternativ ein kontrolliertes Gauss-Seidel-Verfahren statt der Fixpunkt-Iteration durchgeführt wird und ergänzend die Verfeinerungsstrategie, z.B. in Form einer Fehlerschätzung je Koordinatenrichtung, abgeändert wird. Als Referenzmaßstab ist hier beispielsweise der Algorithmus in [14] zu nennen, der für das Wachstumsmodell A einen maximalen Approximationsfehler der Größenordnung $9.6 \cdot 10^{-3}$ bei etwa 900 Knotenpunkten aufweist. Zudem ist der Algorithmus der dynamischen Programmierung unabhängig von der Existenz eines Gleichgewichtpunkts und dessen Lage.

Da die stochastisch, dynamischen Gleichgewichtsmodelle in der Ökonomie eine sehr starke Verwendung finden, haben sich nicht zuletzt deswegen für deren Analyse eine Reihe weiterer Lösungsmethoden etabliert. In dem Buch von [9] werden neben Perturbationsmethoden und der dynamischen Programmierung die Fair-Taylor-Methode, die log-lineare Approximation sowie die linear-quadratische Approximation als Lösungsmöglichkeiten genannt. Allen drei Lösungsmethoden ist gemeinsam, dass ihre Durchführung auf gewisse Bedingungen, die sogenannten „First-Order-Conditions“ angewiesen ist. Damit wird unter anderem die den Namen „Euler-Gleichung“ tragende Gleichung der Form

$$\frac{\partial l(x,u)}{\partial u} + \beta \mathbb{E}\left[\frac{\partial \varphi}{\partial u}(x,u,z) \cdot \frac{\partial l}{\partial x'}(x',u')\right] = 0$$

gemeint. Diese Bedingungen sind zu den im Rahmen der Arbeit bekannten Bedingungen des stochastischen Maximumprinzips äquivalent.
Die in den letzten Jahrzehnten sehr gebräuchliche und weit verbreitete Methode der log-linearen Approximation, die eine Gemeinsamkeit zu der Perturbationsmethode aufweist, beruht auf einer Transformation der Zustands- und Kontrollvariablen. Als neue Variable x_t des Systems wird die logarithmierte Abweichung des Zustands X_t vom steady state $\bar{X}$ gemäß

$$x_t := \ln(\frac{X_t}{\bar{X}}) = \ln(X_t) - \ln(\bar{X})$$

eingeführt. Alle notwendigen Gleichungen, unter anderem die Abbildungsvorschrift der Dynamik φ sowie die „First-Order-Conditions“ , werden in der Form der logarithmierten Abweichung linearisiert. So wird die Originalvariable X_t in den Gleichungen zunächst durch $\bar{X}e^{x_t}$ und anschließend zum Zwecke der Linearisierung durch ihre Approximation $\bar{X} \cdot (1+x_t)$ ersetzt. Zuletzt wird das log-linearisierte System mit der sogenannten Methode der unbestimmten Koeffizienten gelöst. Damit ist die rekursive Bewegungsgleichung für das Gleichgewicht ermittelt und die Lösungspfade

des Originalsystems in den Zuständen X_t lassen sich durch entsprechende Rücktransformation berechnen. Die Gemeinsamkeit zu den in der Arbeit behandelten Perturbationsmethoden besteht darin, dass die Berechnung des Gleichgewichts absolute Voraussetzung für die Durchführung ist.

Die Frage nach der passenden Lösungstechnik ist im allgemeinen schwer zu beantworten (vergleiche [25]), es sind mehrere Aspekte für die Wahl zu berücksichtigen. Ist die Genauigkeit der numerischen Lösung der entscheidende Maßstab, dann ist eine Entscheidung für die Gittermethode der dynamischen Programmierung zu präferieren. Jede der getesteten bzw. erwähnten Lösungsvarianten hat seine Vor- und Nachteile, die beim Auswahlprozess maßgebend sind. Die Perturbationsmethode, insbesondere die direkte Form zur Approximation der optimalen Wertefunktion des Modells, liefert in kurzer Rechenzeit eine lokal äußerst genaue Approximation, mit welcher sich das System in einer Art Vorab-Analyse gut untersuchen lässt.

Anhang A

Notationen

An dieser Stelle des Anhangs werden die wichtigsten Variablenbezeichnungen und Notationen übersichtsartig aufgeführt. Die Auflistung erfolgt dabei nach der Reihenfolge des Auftretens im Text mit Seitenverweis.

K	Kapitalstock	siehe S. 6
L	Produktionsfaktor Arbeit	siehe S. 7
P	aggregierte Produktionsfunktion	siehe S. 7
T	technologischer Fortschritt	siehe S. 7
N	Nutzenfunktion	siehe S. 8
C	Konsum	siehe S. 8
δ	Abschreibungsrate	siehe S. 9
(Ω, Σ, P)	Wahrscheinlichkeitsraum	siehe S. 14
F_X	Verteilungsfunktion einer Zufallsvariablen X	siehe S. 14
p_X	Dichtefunktion einer Zufallsvariablen X	siehe S. 15
$N(0, \sigma^2)$	Normalverteilung mit Erwartungswert 0 und Varianz σ^2	siehe S. 16
T	diskrete Zeitachse	siehe S. 17
φ	Dynamik des Kontrollsystems	siehe S. 18
u_t	Kontrolle in diskreter Zeit	siehe S. 18
X_t	Zustand in diskreter Zeit	siehe S. 18
x_0	Anfangswert	siehe S. 18
z	stochastische Störung	siehe S. 18
X	kompakter Zustandsraum	siehe S. 18

β	Diskontrate	siehe S. 19
J_∞	Zielfunktional des optimalen Kontrollproblems	siehe S. 19
U_{x_0}	zulässige Kontrollmenge zum Anfangswert x_0	siehe S. 19
V_∞	optimale Wertefunktion	siehe S. 20
F_∞	Feedbackfunktion	siehe S. 23
Π	Projektionsoperator, der Funktion auf ihre Approximation abbildet	siehe S. 24
$\mathcal{T}$	Dynamische Programmierung–Operator	siehe S. 24
$\tilde{V}_k$	Iterierte der Werteiteration	siehe S. 24
$\tilde{V}_\infty$	Approximation der optimalen Wertefunktion mittels Werteiteration	siehe S. 24
Γ	Rechteckgitter	siehe S. 25
C_i	Rechteck eines Rechteckgitters	siehe S. 25
E_i	Knoten eines Rechteckgitters	siehe S. 25
$\mathcal{W}$	Raum der affin bilinearen Funktionen	siehe S. 26
η	lokaler Fehlerschätzer	siehe S. 28
$\tilde{u}$	Approximation der Kontrolle u	siehe S. 32
H^t	Hamilton–Funktion	siehe S. 32
λ	Kozustand bzw. adjungierte Variable	siehe S. 33
$(\bar{x}, \bar{u})$	Gleichgewicht bzw. steady state	siehe S. 34
x_t^1	endogene Zustandsvariablen des Perturbationsmodells	siehe S. 38
x_t^2	exogene Zustandsvariablen des Perturbationsmodells	siehe S. 38
ϵ_t	Stochastischer Prozess Weißes Rauschen	siehe S. 38
g	optimale Kontrolle des Perturbationsmodells	siehe S. 38
h	optimale Dynamik des Perturbationsmodels	siehe S. 38
σ	Perturbationsparameter bzw. Störung	siehe S. 38
F	Nullfunktion, eingeführte Hilfsfunktion im Perturbationsmodell	siehe S. 38
$\widehat{V}$	indirekte Taylor–Approximation der optimalen Wertefunktion	siehe S. 46
$\hat{x}_1$	Dynamik für 1. Zustandsvariable des Perturbationsmodells	siehe S. 48
$\hat{x}_2$	Dynamik für 2. Zustandsvariable des Perturbationsmodells	siehe S. 48
$\hat{V}_{dir}$	direkte Taylor–Approximation der optimalen Wertefunktion	siehe S. 49
G	Bezeichnung für rechte Seite der Bellman-Gleichung bei direkter Approximation	siehe S. 50
y	4-dim. Variable, die Zustand, Steuerung und Störung zusammenfasst	siehe S. 50
N_e	Multivariates Newton-Verfahren	siehe S. 52
Ufo(a,b)	Uniformverteilung auf dem Intervall $[a, b]$	siehe S. 61
$T(f)$	Trapezsumme zur Integration einer Funktion f	siehe S. 66

Anhang B

Quellcode der Programme

An dieser Stelle erfolgt eine Auflistung der für die numerische Berechnung benutzten, eigens erstellten Programme in MATLAB, MAPLE sowie in C. Im ersten Abschnitt erfolgt die Auflistung der MATLAB Routinen, in Abschnitt zwei der MAPLE und in Abschnitt drei der C Quellcode.

Um die Implementierung für den Leser verständlicher zu machen, sind die Quelltexte an einigen Stellen mit Kommentaren versehen.
Umlaute sind in der Form 'ae' usw. und 'ß' als 'ss' geschrieben.

B.1 MATLAB

Quelltext V_coeff.m

```
1  function [Vcoeff] = V_coeff(beta,d,gx,hx,gxx,hxx,gss,hss,approx,model)
2  % Uebergabeparameter sind:
3  % beta = Diskontfaktor
4  % d= Abschreibungsrate
5  % gx,hx ...hss = Ableitungen 1./2. Ordnung
6  % approx= Approximationsgrad
7  % model = Modellparameter (1=Modell A 2= Modell B)
8
9  % Funktion zur Berechnung der Taylorkoeffizienten für die
10 % optimale Wertefunktion V
11 % ausgehend von der Bellman-Gleichung
12 %V(x) = sup_u (l(x,u) + V(h(x)))
13
14 %symbolische Variablen werden definiert
15 syms x1 x2 s u
16
17 %Steady state verschoben in die 0
```

```
[x1ggw,x2ggw,uggw]= SS_delta(d);
x1ggw=0;
x2ggw=0;
sggw=0;
%d.h. keine Stoerung --> deterministisches Modell

%Definition der bekannten Approximationen aus dem
%Perturbationsmodell
%g (Kontrolle) und h (Dynamik), mit
%in Ursprung verschobenen Entwicklungspunkt (x1,x2,sigma)=(0,0,0)

%Definition von u_dach als Kontrollfunktion
if approx == 2
    u_dach =  uggw + gx(1)*x1 + gx(2)*x2+ 0.5*gxx(:,1,1)*x1^2+gxx(:,2,1)...
        *x2*x1 +0.5*gxx(:,2,2)*x2^2+ 0.5*gss*s^2;
else
    u_dach =  uggw + gx(1)*x1 + gx(2)*x2;
end

u_dach_abl_x1=diff(u_dach,x1);
u_dach_abl_x2= diff(u_dach,x2);
u_dach_abl_x1x1 = diff(u_dach_abl_x1,x1);
u_dach_abl_x1x2 = diff(u_dach_abl_x1,x2);
u_dach_abl_x2x2 = diff(u_dach_abl_x2,x2);
u_dach_abl_sigma = diff(u_dach, s);
u_dach_abl_sigmasigma = diff(u_dach_abl_sigma,s);

nu_dach_abl_x1= subs(u_dach_abl_x1,{x1,x2},{x1ggw,x2ggw});
nu_dach_abl_x2= subs(u_dach_abl_x2,{x1,x2},{x1ggw,x2ggw});
nu_dach_abl_x1x1 = subs(u_dach_abl_x1x1,{x1,x2},{x1ggw,x2ggw});
nu_dach_abl_x1x2 = subs(u_dach_abl_x1x2,{x1,x2},{x1ggw,x2ggw});
nu_dach_abl_x2x2 = subs(u_dach_abl_x2x2,{x1,x2},{x1ggw,x2ggw});
nu_dach_abl_sigma = subs(u_dach_abl_sigma,{x1,x2,s},...
{x1ggw,x2ggw,sggw});
nu_dach_abl_sigmasigma = subs(u_dach_abl_sigmasigma,{x1,x2,s},...
    {x1ggw,x2ggw,sggw});

%Definition von x1_dach als Dynamik in erster
%Komponente
if approx==2
    x1_dach= x1ggw+ hx(1,1)*x1+hx(1,2)*x2+0.5*hxx(1,1,1)*x1^2 + ...
        hxx(1,2,1)*x2*x1 +0.5*hxx(1,2,2)*x2^2+ 0.5*hss(1)*s^2;
else
    x1_dach= x1ggw+ hx(1,1)*x1+hx(1,2)*x2;
end

x1_dach_abl_x1=diff(x1_dach,x1);
```

```
65 x1_dach_abl_x2 = diff(x1_dach,x2);
66 x1_dach_abl_x1x1 = diff(x1_dach_abl_x1,x1);
67 x1_dach_abl_x1x2 = diff(x1_dach_abl_x1,x2);
68 x1_dach_abl_x2x2 = diff(x1_dach_abl_x2,x2);
69 x1_dach_abl_sigma = diff(x1_dach,s);
70 x1_dach_abl_sigmasigma = diff(x1_dach_abl_sigma,s);
71
72 nx1_dach_abl_x1= subs(x1_dach_abl_x1,{x1,x2},{x1ggw,x2ggw});
73 nx1_dach_abl_x2= subs(x1_dach_abl_x2,{x1,x2},{x1ggw,x2ggw});
74 nx1_dach_abl_x1x1= subs(x1_dach_abl_x1x1,{x1,x2},{x1ggw,x2ggw});
75 nx1_dach_abl_x1x2= subs(x1_dach_abl_x1x2,{x1,x2},{x1ggw,x2ggw});
76 nx1_dach_abl_x2x2= subs(x1_dach_abl_x2x2,{x1,x2},{x1ggw,x2ggw});
77 nx1_dach_abl_sigma= subs(x1_dach_abl_sigma,{x1,x2,s},...
78 {x1ggw,x2ggw,sggw});
79 nx1_dach_abl_sigmasigma= subs(x1_dach_abl_sigmasigma,{x1,x2,s},...
80     {x1ggw,x2ggw,sggw});
81
82 %Definition von x2_dach als Dynamik in zweiter
83 %Komponente
84 if approx==2
85     x2_dach= x2ggw+ hx(2,1)*x1+hx(2,2)*x2+0.5*hxx(2,1,1)*x1^2 +...
86         0.5*hxx(2,2,1)*x2*x1+0.5*hxx(2,1,2)*x1*x2+0.5*hxx(2,2,2)*x2^2+...
87         0.5*hss(2)*s^2;
88 else
89     x2_dach = x2ggw+ hx(2,1)*x1+hx(2,2)*x2;
90 end
91
92 x2_dach_abl_x1 = diff(x2_dach,x1);
93 x2_dach_abl_x2=diff(x2_dach,x2);
94 x2_dach_abl_x1x1= diff(x2_dach_abl_x1,x1);
95 x2_dach_abl_x1x2=diff(x2_dach_abl_x1,x2);
96 x2_dach_abl_x2x2= diff(x2_dach_abl_x2,x2);
97 x2_dach_abl_sigma = diff( x2_dach, s);
98 x2_dach_abl_sigmasigma = diff(x2_dach_abl_sigma,s);
99
100 nx2_dach_abl_x1= subs(x2_dach_abl_x1,{x1,x2},{x1ggw,x2ggw});
101 nx2_dach_abl_x2= subs(x2_dach_abl_x2,{x1,x2},{x1ggw,x2ggw});
102 nx2_dach_abl_x1x1= subs(x2_dach_abl_x1x1,{x1,x2},{x1ggw,x2ggw});
103 nx2_dach_abl_x1x2= subs(x2_dach_abl_x1x2,{x1,x2},{x1ggw,x2ggw});
104 nx2_dach_abl_x2x2= subs(x2_dach_abl_x2x2,{x1,x2},{x1ggw,x2ggw});
105 nx2_dach_abl_sigma= subs(x2_dach_abl_sigma,{x1,x2,s},...
106 {x1ggw,x2ggw,sggw});
107 nx2_dach_abl_sigmasigma= subs(x2_dach_abl_sigmasigma,{x1,x2,s},...
108 {x1ggw,x2ggw,sggw});
109
110 %Definition der modellspezifischen Kostenfunktion
111 if model ==1
```

```
112     l = log(u);
113 else
114     l = log(u)*exp(2*x2);
115
116 end
117 %Einsetzen der Approximation u_dach als Steuerung in l(x,u)
118 l=subs(l,u,u_dach);
119
120 %Berechnen der Ableitungen von l
121 l_abl_u_x1 = diff(l,x1);
122 l_abl_u_x2 = diff(l,x2);
123 l_abl_u_sigma = diff(l,s);
124
125 l_abl_u_x1x1 = diff(l_abl_u_x1,x1);
126 l_abl_u_x1x2 = diff(l_abl_u_x1,x2);
127 l_abl_u_x2x2 = diff(l_abl_u_x2,x2);
128 l_abl_u_sigmasigma= diff(l_abl_u_sigma,s);
129
130 %Auswertung der Ableitungen im steady state
131 nl = subs(l,{x1 x2 s}, {x1ggw x2ggw sggw});
132 nl_abl_u_x1 = subs(l_abl_u_x1,{x1 x2 s}, {x1ggw x2ggw sggw});
133 nl_abl_u_x2 = subs(l_abl_u_x2,{x1 x2 s}, {x1ggw x2ggw sggw});
134 nl_abl_u_sigma = subs (l_abl_u_sigma,{x1 x2 s}, {x1ggw x2ggw sggw});
135
136 nl_abl_u_x1x1 = subs(l_abl_u_x1x1,{x1 x2 s}, {x1ggw x2ggw sggw});
137 nl_abl_u_x1x2 = subs(l_abl_u_x1x2,{x1 x2 s}, {x1ggw x2ggw sggw});
138 nl_abl_u_x2x2 = subs(l_abl_u_x2x2,{x1 x2 s}, {x1ggw x2ggw sggw});
139 nl_abl_u_sigmasigma= subs(l_abl_u_sigmasigma,{x1 x2 s},...
140  {x1ggw x2ggw sggw});
141
142 if approx== 1
143
144 M=[1-beta,0,0; 0, 1-beta*nx1_dach_abl_x1, -beta*nx2_dach_abl_x1; 0,...
145 -beta*nx1_dach_abl_x2, 1 - beta* nx2_dach_abl_x2];
146
147 b=[nl; nl_abl_u_x1; nl_abl_u_x2];
148
149 x=double(M)\double(b);
150
151 V0=x(1);V1=x(2);V2=x(3);
152 Vcoeff = [V0 V1 V2]';
153
154
155 else
156
157     if approx==2
158
```

```
M(1,:)= [1-beta,  0, 0,  0,   0, 0,  0];
M(2,:)=[ 0, 1-beta* nx1_dach_abl_x1,  -beta*nx2_dach_abl_x1,0 ,0 ,0 ,0];
M(3,:)=[ 0, -beta*nx1_dach_abl_x2, 1 - beta* nx2_dach_abl_x2, 0, 0, 0, 0];
M(4,:)= [  0,-beta*nx1_dach_abl_x1x1, - beta*nx2_dach_abl_x1x1, ...
1-beta*(nx1_dach_abl_x1)^2, -2*beta*nx1_dach_abl_x1*nx2_dach_abl_x1,...
-beta*(nx2_dach_abl_x1)^2 ,0];
M(5,:)= [  0,-beta*nx1_dach_abl_x1x2, -beta*nx2_dach_abl_x1x2, ...
-beta*nx1_dach_abl_x1* nx1_dach_abl_x2, 1-beta*(nx1_dach_abl_x1* ...
nx2_dach_abl_x2+nx2_dach_abl_x1*nx1_dach_abl_x2), -beta* ...
nx2_dach_abl_x1*nx2_dach_abl_x2 ,0];

M(6,:)=[  0, -beta* nx1_dach_abl_x2x2, -beta* nx2_dach_abl_x2x2, ...
-beta *(nx1_dach_abl_x2)^2, -2*beta* nx1_dach_abl_x2* ...
nx2_dach_abl_x2 , 1- beta*(nx2_dach_abl_x2)^2,0];
M(7,:)=[  0, -beta*nx1_dach_abl_sigmasigma,  ...
-beta*nx2_dach_abl_sigmasigma, -beta* (nx1_dach_abl_sigma)^2, ...
-2*beta*nx1_dach_abl_sigma * nx2_dach_abl_sigma,  ...
-beta*( nx2_dach_abl_sigma)^2, 1];

b= [nl;nl_abl_u_x1; nl_abl_u_x2; nl_abl_u_x1x1; nl_abl_u_x1x2; ...
nl_abl_u_x2x2; nl_abl_u_sigmasigma];

x=double(M)\double(b);

V0=x(1);V1=x(2);V2=x(3); V11=x(4); V12=x(5); V22=x(6); Vss = x(7);
Vcoeff = [V0 V1 V2 V11 V12 V22 Vss]';


    else
        disp('Higher order of approximation not implemented!');
    end

end
```

Quelltext SS_plot.m

```
function SS_plot

%Vorgabe von Abschreibungswerten
delta = 0:0.05:1
for i=1:length(delta)
[x(i),y(i),u(i)] = SS_delta(delta(i));
q(i)=x(i)/u(i);
end;

```

```
% Plotten des Kapitalstocks, Technischen Fortschritts bzw.
% des Konsums gegen die Abschreibung
subplot(2,1,1)
plot(delta,x,'-r','LineWidth',2,'Marker','x');
hold on
plot(delta,y,':b','LineWidth',2,'Marker','x');
xlabel('Abschreibung');
ylabel('Zustand');
legend('Kapitalstock','technischer Fortschritt');
title('steady state Modell A','FontWeight','bold');
subplot(2,1,2)
plot(delta,u,'-c','LineWidth',2,'Marker','x');
xlabel('Abschreibung');
ylabel('Kontrolle');
legend('Konsum');


  function [x1ggw, x2ggw ,uggw ] = SS_delta(d)
    % Rueckgabe des Gleichgewichtspunkts bei
   % uebergebenem Abschreibungsparameter
    %Input: d = Abschreibungsparameter

    %Definition der Modellparameter
    beta=0.95;  alpha=0.34;
    rho=0.9;    A=5;   sigma=0.008;

    % Gleichgewichtspunkt berechnet ueber
    % stochastisches Maximumprinzip
    x1ggw =  ((1-beta+beta*d)/(beta*alpha*A))^(1/(alpha-1));
    x2ggw =  0.;
    uggw  = A*x1ggw^alpha- d*x1ggw;
  end
end
```

Quelltext ModellA_run.m

```
function [Vcoeffdir] =ModellA_run(delta)
% Berechnung der Koeffizienten der
% Taylorapproximationen für die Kontrolle g, die
% Dynamik h und die optimale Wertefunktion des
% Perturbationsmodells fuer das Modell A
% Uebergabeparameter ist:
% delta = Abschreibungsrate im Modell
% Rueckgabe:
% Vcoeffdir= direkte Taylorkoeffizienten
```

```
%*************************************************
format('short');

%Einstellen des Modells
model=1;  %1: Modell A; 2: Modell B

%Definition der symbolischen Variablen für
%Parameter sowie Kontrolle und Steuerung
syms    sig ALFA Beta RHO A eta d

syms x1 x1p x2 x2p u up
%Notation:
%Suffix -p bezeichnet naechsten Zeitpunkt (z.B
%up = u_{t+1}

f1 = x1p - A*exp(x2) * x1^(ALFA) -(1 - d)*x1 + u ;
f2 = Beta/up * (ALFA * A *exp(x2p)*  x1p^(ALFA -1) + 1 - d)- 1/u;
f3 = x2p - RHO*x2 ;
%Die Gleichgewichtsfunktion f wird zusammengesetzt
f = [f1;f2;f3];

x = [x1 x2];  %Zustand
y = u;          %Kontrolle
xp = [x1p x2p];
yp = up;

[fx,fxp,fy,fyp,fypyp,fypy,fypxp,fypx,fyyp,fyy,fyxp,fyx,fxpyp,fxpy, ...
  fxpxp,fxpx,fxyp,fxy,fxxp,fxx]=anal_deriv(f,x,y,xp,yp);

%Setzen der Modellparameter gemaeß Tabelle (6.1)
A=5;
ALFA=0.34;
Beta=0.95;
RHO=0.9;
sig=0.008;
d=delta;

eta=[0 1]';

%Gleichgewichtspunkt bestimmen
[x1,x2,u] = SS_delta(d);
x2p = x2;
x1p = x1;
up = u;

%Gewuenschte Approximationsordnung einstellen
approx = 2;
```

```
57
58 num_eval
59
60 %lineare Approximation
61 [gx,hx] = gx_hx(nfy,nfx,nfyp,nfxp)
62
63 %quadratische Approximation
64 [gxx,hxx] = gxx_hxx(nfx,nfxp,nfy,nfyp,nfypyp,nfypy,nfypxp, ...
65   nfypx,nfyyp,nfyy,nfyxp,nfyx,nfxpyp,nfxpy,nfxpxp,nfxpx, ...
66   nfxyp,nfxy,nfxxp,nfxx,hx,gx)
67
68 [gss,hss] = gss_hss(nfx,nfxp,nfy,nfyp,nfypyp,nfypy,nfypxp, ...
69   nfypx,nfyyp,nfyy,nfyxp,nfyx,nfxpyp,nfxpy,nfxpxp,nfxpx, ...
70   nfxyp,nfxy,nfxxp,nfxx,hx,gx,gxx,eta)
71
72 %Approximation der optimalen Wertefunktion
73 %indirekte Taylor- Approximation von V
74 Vcoeff = V_coeff(Beta,d,gx,hx,gxx,hxx,gss,hss,approx,model)
75
76 %direkte Taylor-Approximation von V
77 %Import der Maple Koeffizienten
78 filename=strcat('Daten/','DirectV_kappa_0.00_delta_',...
79   num2str(delta, '%.2f'),'.asc');
80 Vcoeffdir = zeros(7,1);
81
82 [fid,message] = fopen(filename,'r');
83 if fid==-1
84   fprintf('ASCII-Datei mit direkten Koeffizienten nicht vorhanden!\n')
85   fprintf('Direkte Koeffizienten gleich 0 gesetzt!');
86
87 else
88 %Einlesen des ASCII Files
89 [data,value]=textread(filename,'%s = %f');
90
91 % Namen der unbek. Taylorkoeffizienten
92 coeffs= ['V0 ';'V1 ';'V2 ';'V11';'V12';'V22';'Vss'];
93 %coeffnames als String Array
94 coeffnames = cellstr(coeffs);
95
96 for i=1 : length(data)
97     Vcoeffdir(strmatch(data(i), coeffnames,'exact'))=value(i);
98 end
99
100 end
101 %Abspeichern der Workspace Variablen
102 save('ModellA');
```

Quelltext ModellB_run.m

Aufgrund der sehr ähnlichen Struktur des Programms im Vergleich zu `ModellA_run.m` ist der Quellcode lediglich mit auf der beigelegten CD-ROM enthalten.

Quelltext evaluation.m

```
function evaluation(model,domain)
% Auswertungsroutine für die Modelle A und B
% zum Vergleich der Approximationen der
% opt. Wertefunktion
% Uebergabeparameter sind:
% model = Modellwahl (1: A, 2: B)
% domain = Wahl des Rechengebiets (Zustandsraum)
% 1:großes Gebiet, 2: kleines Gebiet

%Aufruf der Perturbationsroutinen & Laden der Koeffizienten bzw. Parameter
%(inkl. Koeffizienten Vcoeff_dir fuer Wertefunktion) aus Modell?_run.m

delta=1;
if(model==1)
  ModellA_run(delta);
  kappa=0;
  var= load('ModellA');

else
  if(model==2)
    ModellB_run(delta);
    kappa=2;
    var= load('ModellB');
  else
    fprintf('Unpassender Übergabeparameter model!\n');
    quit cancel;
  end
end;


%Ermitteln des steady state
[x1ggw,x2ggw,uggw]=SS_delta(delta);
fprintf('Gleichgewicht lautet: x= (%f,%f),\t', x1ggw,x2ggw);
fprintf('u = %f\n',uggw);



x1sections=50;
x2sections = 50;
```

```
40
41 if domain ==1
42   x1a=1;
43   x1b=10.0;
44 end
45
46 if domain==2
47   x1a=1.0;
48   x1b= 4.0;
49 end
50
51 x1step = (x1b-x1a)/x1sections;
52
53 % x2-Bereich symmetrisch um die 0 (=Ruhelage)
54 x2a= -0.32;
55 x2b=  +0.32;
56 x2step = (x2b-x2a)/x2sections;
57
58 %Setzung Perturbationsparameter bzw. Std.abweichung z
59 sig=0.008;
60
61 % Gitter definieren zum gewaehlten Bereich I;II
62 [X1,X2] = meshgrid(x1a:x1step:x1b,x2a:x2step:x2b);
63 x1 = X1(1,:);
64 x2 = X2(:,1);
65 lenx1 = length(x1);
66 lenx2 = length(x2);
67
68 % Berechnung der exakten Loesungen für Modell A
69 if (model==1)
70   for i=1:lenx1
71     for j= 1:lenx2
72       [V_exact(j,i),U_exact(j,i)]= exactSolution(x1(i),x2(j));
73     end
74   end
75 end
76
77 % Wertefunktion V mit Perturbationsmethoden (direkte Taylorapproximation)
78 for i=1:lenx1
79   for j=1:lenx2
80     V_pertdir(j,i) = var.Vcoeffdir(1) + var.Vcoeffdir(2)*(x1(i)-x1ggw)  ...
81       + var.Vcoeffdir(3)*(x2(j)-x2ggw) ...
82       + 0.5*var.Vcoeffdir(4)*(x1(i)-x1ggw)*(x1(i)-x1ggw)  ...
83       + var.Vcoeffdir(5)*(x1(i)-x1ggw)*(x2(j)-x2ggw)  ...
84       + 0.5* var.Vcoeffdir(6)*(x2(j)-x2ggw)*(x2(j)-x2ggw) ...
85       + 0.5*var.Vcoeffdir(7)*sig*sig;
86   end
```

```
87 end
88
89 % Wertefunktion V mit Perturbationsmethoden (indirekte Taylorapproximation)
90 for i=1:lenx1
91   for j=1:lenx2
92     V_pert(j,i) = var.Vcoeff(1) + var.Vcoeff(2)*(x1(i)-x1ggw)  ...
93       + var.Vcoeff(3)*(x2(j)-x2ggw) ...
94       + 0.5*var.Vcoeff(4)*(x1(i)-x1ggw)*(x1(i)-x1ggw)  ...
95       + var.Vcoeff(5)*(x1(i)-x1ggw)*(x2(j)-x2ggw)  ...
96       + 0.5* var.Vcoeff(6)*(x2(j)-x2ggw)*(x2(j)-x2ggw) ...
97       + 0.5*var.Vcoeff(7)*sig*sig;
98   end
99 end
100
101 % Generierung der Wertefunktion Werte aus der .asc
102 % Datei der dynamischen Programmierung
103 % Laden aus den ASCII Files
104 if domain ==2
105   file= strcat('Values_delta_',num2str(delta,'%.2f'),'_kappa_',...
106     num2str(kappa,'%1d'),'_small.asc');
107 else
108   file= strcat('Values_delta_',num2str(delta,'%.2f'),'_kappa_',...
109     num2str(kappa,'%1d'),'.asc');
110 end
111
112 Werte  = load(file);
113
114 sb = size(Werte);
115 if(sb(1) == (size(V_pertdir,1)*size(V_pertdir,2)))
116   fprintf('Einlesen erfolgreich!\n');
117 else
118   fprintf('Nicht kompatible ASCII Wertedatei!!\n');
119   quit cancel;
120 end
121
122 for i=1:lenx1
123   for j=1:lenx2
124     % 3. Spalte enthaelt affin bilineare
125     % Approximation V(E_i)
126     V_dyn(i,j) = Werte((j-1)*lenx1 +i,3);
127   end
128 end
129
130 %Plots der Funktionen
131 if model==1
132   % Exakte Wertefunktion
133   figure
```

```
134   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
135     V_exact(1:lenx2,1:lenx1))
136   title('\bf{optimale Wertefunktion {V_\infty} , Modell A}',...
137     'FontSize',12,'FontWeight','bold')
138   grid on;
139   xlabel('x1');
140   ylabel('x2');
141   zlabel('{V_\infty}')
142 end
143
144 % Wertefunktion - Perturbation -direkt
145 figure
146 surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
147   V_pertdir(1:lenx2,1:lenx1))
148 grid on;
149 xlabel('x1');
150 ylabel('x2');
151 zlabel('V_{dir}')
152 if (model==1)
153   title('\bf{Wertefunktion Perturbation direkt, Modell A}',...
154     'FontSize',12,'FontWeight','bold')
155 else
156   title('\bf{Wertefunktion Perturbation direkt, Modell B}',...
157     'FontSize',12,'FontWeight','bold')
158 end;
159
160 % Wertefunktion - Perturbation
161 figure
162 surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
163   V_pert(1:lenx2,1:lenx1))
164 grid on;
165 xlabel('x1');
166 ylabel('x2');
167 zlabel('V_{pert}')
168 if (model==1)
169   title('\bf{Wertefunktion Perturbation, Modell A}',...
170     'FontSize',12,'FontWeight','bold')
171 else
172   title('\bf{Wertefunktion Perturbation, Modell B}',...
173     'FontSize',12,'FontWeight','bold')
174 end;
175
176 % Wertefunktion - DP
177 figure
178 surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
179   V_dyn(1:lenx2,1:lenx1))
180 grid on;
```

```
181 xlabel('x1');
182 ylabel('x2');
183 zlabel('V_{dyn}')
184 if (model==1)
185   title('\bf{Wertefunktion DP, Modell A}',...
186     'FontSize',12,'FontWeight','bold')
187 else
188   title('\bf{Wertefunktion DP, Modell B}',...
189     'FontSize',12,'FontWeight','bold')
190 end;
191
192 %2-elementige Plots
193 if (model==1)
194   % exakt und Perturbation direkt
195   figure
196   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
197     V_exact(1:lenx2,1:lenx1))
198   hold on
199   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
200     V_pertdir(1:lenx2,1:lenx1))
201   xlabel('x_1');
202   ylabel('x_2');
203   title('\bf{exakte Wertefunktion & V Perturbation direkt, Modell A}',...
204     'FontSize',12,'FontWeight','bold')
205
206   % exakt und Perturbation indirekt
207   figure
208   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
209     V_exact(1:lenx2,1:lenx1))
210   hold on
211   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
212     V_pert(1:lenx2,1:lenx1))
213   xlabel('x_1');
214   ylabel('x_2');
215   title('\bf{exakte Wertefunktion & V Perturbation, Modell A}',...
216     'FontSize',12,'FontWeight','bold')
217
218   %exakt und DP
219   figure
220   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
221     V_exact(1:lenx2,1:lenx1))
222   hold on
223   surf(x1(x1a):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
224     V_dyn(1:lenx2,1:lenx1))
225   xlabel('x_1');
226   ylabel('x_2');
227   title('\bf{exakte Wertefunktion & V DP, Modell A}',...
```

```
228     'FontSize',12,'FontWeight','bold')
229 end
230
231 %DP und Perturbation direkt
232 figure
233 surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
234   V_pertdir(1:lenx2,1:lenx1))
235 hold on
236 surf(x1(x1a):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
237   V_dyn(1:lenx2,1:lenx1))
238 xlabel('x_1');
239 ylabel('x_2');
240 if (model==1)
241   title('\bf{V Perturbation direkt & V DP, Modell A}',...
242     'FontSize',12,'FontWeight','bold')
243 else
244   title('\bf{V Perturbation direkt & V DP, Modell B}',...
245     'FontSize',12,'FontWeight','bold')
246 end;
247
248 %DP und Perturbation indirekt
249 figure
250 surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
251   V_pert(1:lenx2,1:lenx1))
252 hold on
253 surf(x1(x1a):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
254   V_dyn(1:lenx2,1:lenx1))
255 xlabel('x_1');
256 ylabel('x_2');
257 if (model==1)
258   title('\bf{V Perturbation & V DP, Modell A}',...
259     'FontSize',12,'FontWeight','bold')
260 else
261   title('\bf{V Perturbation & V DP, Modell B}',...
262     'FontSize',12,'FontWeight','bold')
263 end;
264
265 %Fehlerberechnung und anschl. Plot der Fehler
266 if (model==1)
267
268   %Fehler DP
269   for i=1:lenx1
270     for j=1:lenx2
271       Verr_ex_dyn(j,i) = abs(V_exact(j,i)- V_dyn(j,i));
272     end
273   end
274
```

```
275   figure
276   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
277     Verr_ex_dyn(1:lenx2,1:lenx1))
278   grid on;
279   xlabel('x1');
280   ylabel('x2');
281   title('\bf{Fehler DP, Modell A}','FontSize'...
282     ,12,'FontWeight','bold')
283
284   %Fehler direkte Perturbationsmethode
285   for i=1:lenx1
286     for j=1:lenx2
287       Verr_ex_pert(j,i) = abs(V_exact(j,i)- V_pertdir(j,i));
288     end
289   end
290
291   figure
292   surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
293     Verr_ex_pert(1:lenx2,1:lenx1))
294   grid on;
295   xlabel('x1');
296   ylabel('x2');
297   title('\bf{Fehler Perturbation direkt, Modell A}',...
298     'FontSize',12,'FontWeight','bold')
299 end
300
301 %Abweichung direkte - indirekte Perturbationsmethode
302 for i=1:lenx1
303   for j=1:lenx2
304     Vdiff_dir_pert(j,i) = abs(V_pert(j,i)- V_pertdir(j,i));
305   end
306 end
307
308 figure
309 surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
310   Vdiff_dir_pert(1:lenx2,1:lenx1))
311 grid on;
312 xlabel('x1');
313 ylabel('x2');
314 if (model==1)
315   title('\bf{Differenz V Perturbation direkt & indirekt, Modell A}',...
316     'FontSize',12,'FontWeight','bold')
317 else
318   title('\bf{Differenz V Perturbation direkt & indirekt, Modell B}',...
319     'FontSize',12,'FontWeight','bold')
320 end;
321
```

```
322 %Abweichung DP Loesung- Perturbation
323 for i=1:lenx1
324   for j=1:lenx2
325     Vdiff_dyn_pert(j,i) = abs(V_dyn(j,i)- V_pertdir(j,i));
326   end
327 end
328
329 figure
330 surf(x1(1):x1step:x1(lenx1),x2(1):x2step:x2(lenx2), ...
331   Vdiff_dyn_pert(1:lenx2,1:lenx1))
332 grid on;
333 xlabel('x1');
334 ylabel('x2');
335 if (model==1)
336   title('\bf{Differenz V Perturbation direkt & V DP, Modell A}',...
337     'FontSize',12,'FontWeight','bold')
338 else
339   title('\bf{Differenz V Perturbation direkt & V DP, Modell B}',...
340     'FontSize',12,'FontWeight','bold')
341 end;
342
343 %Hilfsfunktion, die exakte Kontrolle & Wertefunktion liefert
344   function [V, U] =  exactSolution (x1, x2)
345     %Definition Modellparameter
346     beta = 0.95; alpha=0.34;   rho=0.9;   A=5;
347
348     % Komponenten der geschlossenen Loesung V(x)=B+C*ln(x1)+D*x2
349     B= (log((1-beta*alpha)*A)+((beta*alpha)/(1-beta*alpha))*...
350       log(beta*alpha*A))/(1-beta);
351     C=alpha/(1-alpha*beta);
352     D=1/((1-alpha*beta)*(1-rho*beta));
353
354     % Berechnung des exakten Werts fuer uebergegeben
355     % Zustand x=[x1,x2]
356     V=B +C*log(x1) +D*x2;
357     U=(1-alpha*beta)*A*exp(x2)*x1^alpha;
358   end
359
360 end
```

Quelltext simplot2d.m

```
1 function simplot2d(simfile)
2 % grafische Darstellung der Loesung sowie
3 % der zugehoerigen Kontrolle
```

```
% Input: mit C erzeugtes ASCII Simulationsfile

%Auslesen der Parameter der Simulation
pos= findstr(simfile,'_');
pos=pos+1;
delta = str2num(simfile(pos(2):(pos(3)-2)));
kappa= str2num(simfile(pos(4):(pos(5)-2)));

b  = load(simfile);
sb = size(b);
t= 1:sb(1);

for i=1:sb(1)
    x(i)=b(i,1);
    y(i)=b(i,2);
    z(i)=b(i,3);
    u(i)=b(i,4);
 end;

figure;
subplot(2,2,1)
for indexnumber=1:length(t)
plot(t(1:indexnumber), x(1:indexnumber),'-r','LineWidth',2)
set(gca,'XLim',[0 t(sb(1))]);
set(gca,'xtick',[0:30:135])
drawnow;
end
xlabel('Zeitpunkt t');
ylabel('x1');
legend('Kapitalstock');

subplot(2,2,2)
for indexnumber=1:length(t)
plot(t(1:indexnumber), y(1:indexnumber),'-b','LineWidth',2)
set(gca,'XLim',[0 t(sb(1))]);
set(gca,'xtick',[0:30:135])
drawnow;
end

xlabel('Zeitpunkt t');
ylabel('x2');
legend('technischer Fortschritt');

if delta==1

    if kappa==0
        text='Modell A';
```

```
51      end
52      if kappa==2
53          text='Modell B';
54      end
55
56  else
57      text= strcat('Modell mit delta=',num2str(delta),...
58        ', kappa=',num2str(kappa));
59
60  end
61
62  subplot(2,2,3:4)
63  for indexnumber=1:length(t)
64  plot(t(1:indexnumber), u(1:indexnumber),'-c','LineWidth',2)
65  set(gca,'XLim',[0 t(sb(1))]);
66  set(gca,'xtick',[0:30:135])
67  drawnow;
68  end
69  xlabel('Zeitpunkt t');
70  ylabel('u');
71  legend('Konsum');
72  title(text,'FontSize',14,'FontWeight','bold')
73
74  %Abspeichern der Grafik im Ordner Simulationen
75  picname= strcat('Simulationen',filesep,simfile(1:length(simfile)-4),'.fig');
76  saveas(gcf,picname);
```

B.2 Maple

Quelltext Vdirect.mw

```
# Routine zur direkten Approximation der optimalen Wertefunktion anhand Bellman-Gleichung
# für Wachstumsmodelle A und B
#
> restart; readlib(mtaylor); infolevel[fsolve] := 5
> kappa := 0; delta := 1.0
> V := V0+V1*(x-xggw)+V2*(y-yggw)+1/2*V11*(x-xggw)^2+V12*(x-xggw)*(y-yggw)+1/2*V22*(y-yggw)^2+1/2*Vss*sigma^2
> #Definition der übrigen Modellparameter (Diskontrate beta etc.)
>    beta := 0.95; alpha:=0.34; A:=5; rho:=0.9;

> #Definiton des steady state (xggw,yggw,uggw)per hergeleiteter Formel
> xggw:=((1-beta*(1-delta))/(beta*alpha*A))^(1/(alpha-1));
> yggw:=0.;
> uggw:=A*(xggw)^alpha-delta*(xggw);
> #Definition der Ertragsfunktion
> l := (x,y,u)->ln(u)*exp(kappa*y);
> if(kappa=0 and delta=1)then
> #Definiton der bek., exakten optimale Wertefunktion mit Komp. B,C,d
> B:= (log((1-beta*alpha)*A)+((beta*alpha)/(1-beta*alpha))*log(beta*alpha*A))/(1-beta);
> C:= alpha /(1-alpha*beta);
> d:= 1./((1-alpha*beta)*(1-rho*beta));
> V_ex:=(x,y)->B+C*ln(x)+d*y;
> # zum Vergleich: die Taylorreihe von V_ex
> Vtaylor:=mtaylor(V_ex(x,y),[x=xggw,y=yggw],3);
> end if;
> # unbekannte Taylorkoeff.von V zweite Ordnung
>   Vcoeff := [V0, V1, V2, V11, V12, V22, Vss];
> #Definition der 2-dim.Dynamik
> phi := Vector([A*exp(y)*x^alpha +(1-delta)*x -u, rho *y]);
> Vh := subs({x=phi[1],y=phi[2]},V):
> Vh;
> #Definition von G (rechte Seite Bellman Gleichung)
> G:=l(x,y,u)+expand(beta*Vh);
> #Taylorentwicklung der rechten Seite
> #3. Argument von mtaylor bestimmt totale Ordnung bei der abgebrochen werden soll
> taylorpol:=mtaylor(G,[x=xggw,y=yggw,u=uggw,sigma=0],3);
> convert(taylorpol,polynom):type(taylorpol,polynom);
> RS := sort(expand(taylorpol), u)
> # Ermittlung der Koeffizienten des Ausdrucks in u: "a2*u^2 + a1*u +a0"
```

```
> for i from 0 to 2 do
> a[i]:= coeff (RS, u, i);
> od;
> #Bestimmung des maximalen u (analytisch ableiten nach u);
> f:= a[0]+a[1]*u +a[2]*u^2;
> df := diff(f, u)
> # Bestimmung des Extremums umax aus notwend. Bedingung
> umax:=solve(df=0,u);
> df2:=diff(f,u,u);#(V11 negativ)
> #u ist abhängig von x,y und Vcoeff
> #Einsetzen von Symbol umax in die rechte Seite
> right:=subs(u=maxu,taylorpol);
> left:=V;right:=subs(maxu=umax,right):right:=expand(right):
> left := collect(left, [x, y, sigma], distributed);
  right2:=collect(right,[x,y,sigma],distributed);
> Kright := coeffs(right2, [x, y, sigma], 'Potenzenr'); Kleft := coeffs(
    left, [x, y, sigma], 'Potenzenl')
> for i to 7 do eq[i] := Kleft[i] = Kright[i]; print(eq[i]); printf(''\n'
    ') end do


> equations := {eq[1], eq[2], eq[3], eq[4], eq[5], eq[6], eq[7]}
> lös := fsolve(equations); taylorcoeff := lös;
> filename := sprintf("DirectV_kappa_%.2f_delta_%.2f.asc", kappa, delta)
> fp := fopen(filename, WRITE);
> for k from 1 to 7 do
> fprintf(fp, "%q\n",taylorcoeff[k]);
> od:
> fclose(fp);
```

B.3 C

Quelltext zufallszahlen.c

```
1  /*Einbindung des Headerfiles*/
2  #include "zufallszahlen.h"
3
4  /*Erzeugung uniform verteilter Zufallszahlen
5    nach Methode von Park & Miller.*/
6  /*Rückgabe einer Ufo(0,1) Zufallszahl*/
7
8  double ufo(long *idum)
9  {       long k;
10     double ans;
11         *idum ^= MASK;
12         k=*idum/IQ;
13         *idum = IA*(*idum-k*IQ)-IR*k;
14         if(*idum <0)
15         *idum +=IM;
16         ans = AM*(*idum);
17         *idum ^= MASK;
18         return ans;
19 }
20
21
22
23 /*Polare Form der Box-Muller Transformation nach Marsaglia*/
24 double gasdev (long * idum)
25 {
26         static double y;
27         static int iset;
28         double fac,rsq,v1,v2;
29         if(*idum <0) iset=0;
30     if(iset==0){
31         do
32                 {
33                         /*(v1,v2) als Punkt im Einheitskreis*/
34
35                         v1 =2.0 * ufo(idum)-1.0;
36                         v2 =2.0 * ufo(idum)-1.0;
37
38         /*Bilden des quadrierten Abstands*/
39                         rsq = v1*v1 + v2*v2;
40                 }
41         while ( rsq >=1.0 || rsq==0.0);
42
```

```
43          fac = sqrt((-2.0*log(rsq))/rsq);
44      y = v1*fac;
45      iset=1;              /*Flagge setzen*/
46      return v2*fac;
47      }
48      else                 /*Flagge auf Null rücksetzen*/
49      {  iset =0;
50         return y;
51      }
52
53
54
55  }
```

Der Quelltext des einzubindenden C–Headerfiles findet sich auf der CD–ROM, enthält jedoch typischerweise nur die Funktionsprototypen und die Definition einiger Konstanten.

Quelltext DPalg.c

```
1  /*Diplomarbeit*/
2  /*Dynamische Programmierung*/
3  /*Algorithmus (3.29) zur Berechnung der opt. Wertefunktion      */
4  /*Christian Spieler*/
5
6  /* Einbinden der Header-Datei von GRIDGEN */
7  #include "gridgen.h"
8
9  /* fuer Zufallszahlengenerierung*/
10 #include "zufallszahlen.h"
11 /*sonstige benoetigte Headerfiles*/
12 #include <math.h>
13 #include <float.h>
14 #include <stdlib.h>
15 #include <stdio.h>
16 #include <string.h>
17
18 /*Modellparameter in globaler Strukturvariable p*/
19 struct parameters
20 {
21 double beta;    /*Diskontierungsfaktor */
22 double delta;           /*Abschreibungsrate*/
23 double A;                       /*Technologieparamater*/
24 double alpha;           /*Substitutionsparameter Kapitalstock*/
25 double rho;                     /* Technologischer Faktor*/
26 double sigma;           /*Stoerungsparameter*/
27 int kappa;                      /*Zielfunktionparameter*/
```

```
28 }p ={0.95, 1.0, 5., 0.34, 0.9, 0.008, 0};
29
30 /*Parameter fuer adaptive Gitter*/
31
32 #define Refine                          /*Schalter zur Adaptation (
       entkommentieren)*/
33 #define DIFF 0.00001    /*Differenz Knotenwerte bei regelmaessigem Gitter
       */
34 #define RTOL  0.001     /* Verfeinerungstoleranz */
35 #define RHO  0.8                        /* Sicherheitsfaktor fuer
       Verfeinerung! */
36
37 #define U_SECTIONS 160;         /*Diskretisierungsanzahl der Kontrollmenge
       */
38 const unsigned int MAX_ITER = 150;              /*Maximalanzahl
       Iterationen*/
39
40 static unsigned int aufrufe=0;          /*Zaehler fuer Aufrufe der
       Trapezregel*/
41 const unsigned int anz_sim = 1000; /*Anzahl an Simulationen*/
42
43
44 /*Dynamik des Modells A & B*/
45 void dynamik (double *x, double u, double z, double *erg)
46 {
47 erg[0] = p.A* exp(x[1])* pow(x[0],p.alpha) +(1-p.delta)*x[0] - u;
48 /*z stochastischer Einfluss*/
49 erg[1] = p.rho * x[1] + z;
50 }
51
52 /*Definition der exakten Wertefunktion*/
53 double V_exact(double *x)
54 {
55 double erg;
56 double B,C,D;
57 B=(log((1-p.beta*p.alpha)*p.A)+((p.beta*p.alpha)/(1-p.beta*p.alpha))*log(p
       .beta*p.alpha*p.A))/(1-p.beta);
58 C= p.alpha /(1-p.alpha*p.beta);
59 D= 1./((1-p.alpha*p.beta)*(1-p.rho*p.beta));
60 erg =B;
61 erg+= C* log(x[0]);
62 erg+= D* x[1];
63 return erg;
64 }
65
66 /*Definition der Nutzenfunktion*/
67 double l(double *x, double u)
```

```
68 {
69 return  log(u)*exp(p.kappa*x[1]);
70 }
71
72 /*Dichte der Gaussverteilung N(0,sigma)*/
73 double gaussian (double z)
74 {
75 double density;
76 density = exp(-0.5 * pow(z/p.sigma,2));
77 density/= (p.sigma*sqrt(2*PI));
78 return density;
79 }
80
81 void parameter_output()
82 {
83 printf("Parametersetzungen:\n");
84 printf("_____________________________\n");
85 printf("Delta.....................%f\n",p.delta);
86 printf("Kappa.....................%d\n",p.kappa);
87 printf("Beta......................%f\n",p.beta);
88 printf("A.........................%f\n",p.A);
89 printf("Alpha.....................%f\n",p.alpha);
90 printf("Rho.......................%f\n",p.rho);
91 printf("Sigma.....................%f\n",p.sigma);
92 printf("Bitte beliebige Taste druecken \n");
93 printf("zur Ausfuehrung des Algorithmus!\n");
94 }
95
96 void steady_state (double *xggw, double * uggw)
97 {
98 xggw[0] =  (1-p.beta+p.beta*p.delta)/(p.beta*p.alpha*p.A);
99 xggw[0] =       pow(xggw[0],(1/(p.alpha-1)));
100 xggw[1] =  0.;
101 *uggw  = p.A*pow(xggw[0],p.alpha)-p.delta*xggw[0];
102
103 }
104 void belege_gamma(double * lo, double * hi, double * delta)
105 {
106 /*Rechengebiet Gamma (raeumliche Diskretisierung) ist:
107 [lo[0];hi[0]]x [lo[1];hi[1]]*/
108 if(p.delta >=0.4)
109 {
110 lo[0]=1.0;
111 hi[0]=10.0;              /*alternativ: 10.0*/
112 }
113 else
114 /*Gebiet Gamma in Abh. vom Gleichgewichtspunkt*/
```

```
115 { double ss[2],u;
116 steady_state(ss,&u);
117 lo[0]=ss[0]-6;
118 if (lo[0]<=0)
119 lo[0] =1.0;
120 hi[0] =floor(ss[1]+6);
121 }
122 /*2. Dimension fest, da ss[1] stets 0*/
123 lo[1]=  -0.32;/*alternativ:-0.32;*/
124 hi[1]=  0.32;/*alternativ: 0.32;*/
125
126 /*Gitterschrittweite delta*/
127 delta[0]=(hi[0]-lo[0])/20;
128 delta[1]=(hi[1]-lo[1])/10;
129 }
130
131
132 /*Routine zur Auswertung des Integranden des Erwartungswert*/
133 double integrand (double *x, double u, double z, double * fval,qgrid * g,
       int *flag)
134 {
135 double temp=0;
136 /*fval enthaelt naechsten Zustand*/
137 dynamik(x, u, z, fval);
138
139 int currentflag=0;
140 /*Abfragen des Werts der Wertefunktion an diesem Punkt*/
141 temp = value(g, fval, &currentflag);
142 /*fval muss stets im Gebiet bleiben, d.h. flag immer 0 bleiben*/
143 *flag= *flag || currentflag;
144
145 return (temp * gaussian(z));
146
147 }
148 /*Trapezregel zur numerischen Integration*/
149 double trapezformel( double a, double b, unsigned int sections, double *
       state, double u, double * fval, qgrid * g, int * flag)
150
151 { double sum, h;
152 aufrufe++;
153 int i,erg;
154 sum=0.;
155 h = ( b - a) / sections;
156 /*Falls fval nicht im Gebiet liegt, wird 0 zurueckgegeben*/
157 sum = 0.5 * (integrand(state,u,a,fval,g,flag) + integrand(state,u,b,fval,g
       ,flag));
158 /*mehr als 1 Intervall , d.h. zusammengesetzte Trapezregel*/
```

```
159 if( sections > 1)
160 {
161 double x;
162 for ( i = 1; i <= sections-1; i++)
163 {
164 x = a + i*h;
165 sum +=  integrand(state,u,x,fval,g,flag);
166 }
167 }
168 return sum * h;
169 }
170
171
172 double optprinzip(double *x, qgrid *g, double *u)
173 {
174 /* Wahl U:=[0.5;10.5] */
175
176 double ub_u = 10.5;
177 double lb_u = 0.5;
178 double current_u;
179 int u_cnt, flag, firstval;
180 double bounddiff = ub_u - lb_u;
181 /*Diskretisierung der Kontrollmenge*/
182 double delta_u = bounddiff/U_SECTIONS;
183 double u_max = lb_u;
184 double temp = 0;
185 double max_val;
186 max_val = -DBL_MAX;
187
188 double fval[2];
189 /*Diskretisierung des normalverteilten stoch. Einflusses
190 11 gleichverteilte Werte im Intervall [-0.032;0.032]*/
191 double z_range[2];
192 z_range[0]= -4*p.sigma;
193 z_range[1] = 4*p.sigma;
194
195 unsigned int sections = 10;
196 double expectation = 0.;
197 /* berechne den Maximalwert aus dem Optimalitaetsprinzip */
198 u_cnt = 0;
199 firstval = 1;
200 do
201 {
202 current_u = lb_u + u_cnt*delta_u;
203 flag=0;
204 expectation = trapezformel(z_range[0],z_range[1],sections,x,current_u,fval
    ,g,&flag);
```

```
205 temp = l(x,current_u) + p.beta * expectation;
206 if ((firstval==1) || (temp > max_val))
207 {
208 /*Ueberpruefung, ob x ausserhalb des Gitters lag bei gewaehltem current_u
    */
209 if (flag==0)
210 {
211 u_max = current_u;
212 max_val = temp;
213 firstval=0;
214 }
215 }
216
217 u_cnt++;
218 } while (u_cnt*delta_u <= bounddiff);
219
220 if (u!=NULL) *u = u_max;
221 return max_val;
222 }
223
224
225 /*Funktion liefert optimale approximative Kontrolle zurueck*/
226 double feedback(double *x, qgrid *g)
227 {       double u_max;
228 double u=0;
229 optprinzip(x,g,&u);
230 u_max = u;
231 return u_max;
232 }
233
234
235 /*Abspeichern der opt. Wertefunktion in ASCII Datei*/
236 void  savevalues(qgrid * g, int anz, char* filename)
237 {
238 export_val2d(g,filename,anz);
239 }
240
241 double u_pert( double * x)
242 {
243 double g0,g1,g2,g11,g12,g22,gss;
244 double ss[2];
245 double uggw;
246 /*Ermittlung Steady State*/
247 steady_state(ss,&uggw);
248 double control=0;
249 if(p.delta==1 && p.kappa==2)
250 {       /*Modell B:g Koeffizienten*/
```

```
g0   =  4.3331;
g1   =  0.7126; g2   =  4.7277;
g11  = -0.2275; g12  =  0.7775;
g22  =  5.0563; gss  = -7.5821;
}
else
{
if(p.delta==1 && p.kappa==0)
{
/*Modell A: g Koeffizienten*/
g0 = 4.3331;
g1 = 0.7126; g2 = 4.3331;
g11 = -0.2275; g12 = 0.7126;
g22 = 4.3331; gss = 0.0;
}
else
return -123456789.;
}

control= g0 + g1*(x[0]-ss[0]) + g2*x[1] +0.5* g11*pow((x[0]-ss[0]),2) +
   g12*(x[0]-ss[0])*x[1] + 0.5*g22*pow(x[1],2) + 0.5* gss*p.sigma*p.sigma;
return control;
}


double simulation(double *x0, qgrid *g,long * idum, char * file, int
   method)
{
/*Einmalige Speicherung der Werte in einer .asc Datei */
/*der Form x1 Wert | x2 Wert | Stoerung z | Kontrolle u */

FILE * simfile;
simfile=fopen(file,"r");

int issaved;
if(simfile == NULL)
{               simfile = fopen(file,"wt");
issaved=0;
}
else
{
fclose(simfile);
issaved=1;
}

double x[2], x_neu[2];
double z, u;
```

```
296 int t=0;
297 double J = 0.;
298 x[0] = x0[0];
299 x[1] = x0[1];
300 while(pow(p.beta,t)>=0.001)
301 {
302 /* normalverteilte N(0,sigma) Zufallszahl z ziehen*/
303 z = p.sigma * gasdev(idum) ;
304
305 if(method==1)
306 /* Feedback ermitteln fuer Zustand x ueber DP*/
307 u=feedback(x,g);
308 else
309 {/* g aus Perturbationsmethode auswerten fuer Zustand x,*/
310 if(method==2)
311 u = u_pert(x);
312 else
313 {printf("Falscher Uebergabeparameter method!\n");break;}
314 }
315
316 if(!issaved )
317 {
318 fprintf(simfile,"%lf\t%lf\t",x[0],x[1]);
319 fprintf(simfile,"%lf\t",z);
320 fprintf(simfile,"%lf\n",u);
321 }
322 /* einen Zeitschritt ausfuehren*/
323 dynamik(x,u,z,x_neu);
324 J = J + pow(p.beta,t) * l(x,u);
325
326 x[0] = x_neu[0];
327 x[1] = x_neu[1];
328 t++;
329 }
330
331 if(!issaved)
332 fclose(simfile);
333 /* Zielfunktionswert zurueckgeben*/
334 return J;
335 }
336
337
338 int main(int argc, char *argv [])
339 {
340 /*Modellparameter fuer delta und kappa koennen ueber
341 die Kommandozeile (argv) uebergeben werden*/
342 char pause;
```

```
343 printf("\n------------------Dynamische Programmierung---------------\n");
344 printf("|Algorithmus (3.29) zur Berechnung der opt. Wertefunktion|\n");
345 printf("-----------------------------------------------------------\n");
346 if(argc==1)
347 {       printf("Keine Parameter uebergeben\n");
348 parameter_output();
349 }
350 else
351 {
352 if(argc==2)
353 {       p.delta= atof(argv[1]);
354 printf("Delta wurde uebergeben!\n");
355 }
356
357 if(argc==3)
358 {       p.delta = (double) atof(argv[1]);
359 p.kappa = (int) atoi(argv[2]);
360 printf("Delta wurde uebergeben!\n");
361 printf("Kappa wurde uebergeben!\n");
362 }
363 parameter_output();
364 }
365 pause= getchar();/*Pause*/
366
367 /*Bestimmung des steady states des Modell*/
368 double xggw[2];
369 double uggw;
370 steady_state(xggw, &uggw);
371 printf("Gleichgewicht des Modells lautet:\n");
372 printf("_____________________________\n");
373 printf("x1.......................%f\n",xggw[0]);
374 printf("x2.......................%f\n",xggw[1]);
375 printf("u.......................%f\n\n",uggw);
376
377 /*Start des Algorithmus*/
378 int i;
379 int k=0; /*Iterationsvariable k*/
380
381 double max_error, max_err, zeit;
382
383 /* Deklaration der Variablen (2d) */
384 double lo[2], hi[2], Delta[2], x[2], u;
385
386 /*Definition einer Gittervariablen g (fuer Wertefunktion)
387 sowie u (fuer Kontrolle)*/
388 qgrid * g;
389 FILE * stat_output;
```

```
390
391 char datei[100];
392 char statistik[100];
393 sprintf(statistik,"Stat_delta_%.2lf_kappa_%1d_RTOL_%.3f_RHO_%.2f.dat",p.
       delta,p.kappa,RTOL,RHO);
394 stat_output=fopen(statistik,"r");
395
396
397 if(stat_output == NULL)
398 stat_output = fopen(statistik,"wt");
399 else
400 {       fclose(stat_output);
401 stat_output = fopen(statistik,"a");
402 }
403 int index, flag;
404 int flag2;
405
406 /*Schritt 1 des Algorithmus*/
407 /* Festlegen der Gittergeometrie und Wahl des Startgitters */
408 belege_gamma(lo,hi,Delta);
409 g= create_grid(lo,hi,Delta,2,2);
410
411 /*Start der Zeitmessung*/
412 tic(g);
413 #ifdef Refine
414 fprintf(stat_output,"Ausgabe Fehlerschaetzer Eta_max pro Iteration:\n");
415 #else
416 fprintf(stat_output,"Ausgabe max. Knotenwertabweichung pro Iteration:\n");
417 #endif
418 printf("Bitte beliebige Taste druecken \n");
419 printf("zum Start des Algorithmus!\n");
420 pause=getchar();
421
422 unsigned int nodes;
423
424 do{
425 /*Schritt 2 des Algorithmus*/
426 /*Schleife uber alle Eckpunkte, Auswertung des DP Operators*/
427 double old=0;
428 double max_diff=0;;
429 double diff;
430 index = first_node(g, x);
431 do
432 {
433 /* Ausgabe der Koordinaten */
434
435 #ifndef Refine
```

```
436 old = current_nodevalue(g);
437 #endif
438
439 if (k==0)
440 {       set_current_nodevalue(g, 0.);
441 max_diff=1;
442 }
443 else
444 {
445 set_current_nodevalue(g,optprinzip(x,g,NULL));
446 }
447
448
449 #ifndef Refine
450 diff= fabs(old-current_nodevalue(g));
451 max_diff = MAX(diff,max_diff);
452
453 #endif
454 index = next_node(g, x);
455 }
456 while (index!=-1);
457 #ifndef Refine
458 fprintf(stat_output,"%f\n",max_diff);
459 if(max_diff < DIFF) break;
460 #endif
461
462 #ifdef Refine
463
464 /* Schritt 3 des Algorithmus*/
465 /* Schleife ueber Testpunkte zur Bestimmung des maximalen Fehlers eta_max
       */
466 prepare_adaptation(g);
467 if(k==0)
468 max_err = 2*RTOL;
469 else
470 max_err = 0.0;
471
472 index = first_testpoint(g,x);
473 do
474 { /*Fehler wird nur in den Kantentestpunkten gemessen*/
475 if (current_testpointtype(g)==1)
476 {
477 double val1, val2, err;
478
479 /* gewuenschter Wert */
480 if (k==0)
481         val1 = 0.;
```

```
482 else
483         val1 = optprinzip(x,g,NULL);
484
485 /* interpolierter Wert im Testpunkt wird abgefragt */
486 val2 = current_nodevalue(g);
487
488 /* Fehler err wird berechnet  */
489 err = fabs(val1-val2);
490 /*Knotenvariable dpvalue wird gesetzt fuer den aktuellen Testpunkt
491 um erneuten optprinzip Aufruf zu sparen*/
492 set_dpvalue(g,val1);
493
494 /*Maximumsbestimmung*/
495 max_err = MAX(max_err, err);
496
497 }
498 index = next_testpoint(g, x);
499 }
500 while (index!=-1);
501
502 /* Ausgabe des maximalen Fehlers*/
503 fprintf(stat_output,"Eta_max: %lf\n",max_err);
504 /*Beendigung Algorithmus bei sehr kleinem Fehlerschaetzer*/
505 if(max_err < RTOL)
506 break;
507
508 /*Verfeinerung der Zellen mit Testpunktfehler groesser als RHO * max_err*/
509 index=first_testpoint(g,x);
510 double val1, val2, err;
511 do
512 {
513 if (current_testpointtype(g)==1)
514         {
515         if (k==0)
516                 val1 = 0.;
517         else
518                 val1 = get_dpvalue(g);
519
520         /* Abfrage interpolierter Wert im Testpunkt */
521         val2 = current_nodevalue(g);
522         /* Fehler eta_l wird berechnet*/
523         err = fabs(val1-val2);
524         if(err >= RHO*max_err)
525                 {
526                 set_current_nodevalue(g,val1);
527                 insert_current_testpoint(g);
528                 }
```

```
529         }
530 index = next_testpoint(g, x);
531 }
532 while (index!=-1);
533
534 printf("Iteration step #%d finished...\n",k);
535 complete_adaptation(g);
536 #endif
537 k++;/*Iterationszaehler hochzaehlen*/
538 }
539 while( k <=MAX_ITER);
540
541 /*Stop der Zeitmessung*/
542 zeit = toc(g);
543 k-=1;
544
545 /*finales Gitter abspeichern, Export nach MATLAB*/
546 /*als .asc Datei*/
547 #ifdef Refine
548 sprintf(datei,"Grid_delta_%.2lf_kappa_%1d_RTOL_%.3f_RHO_%.2f.asc",p.delta,
      p.kappa,RTOL,RHO);
549 #else
550 sprintf(datei,"Grid_delta_%.2lf_kappa_%1d.asc",p.delta,p.kappa);
551 #endif
552 export_gridandval(g, datei);
553
554 /*Sicherung der Knotenwerte*/
555 char  vfile [100];
556 sprintf(vfile,"Values_delta_%.2lf_kappa_%1d.asc",p.delta,p.kappa);
557 savevalues(g,51,vfile ) ;
558
559 fprintf(stat_output,"\n******ZUSAMMENFASSUNG DP ALGORITHMUS:******\n");
560 fprintf(stat_output,"_____________________________\n");
561 fprintf(stat_output,"Ausgefuehrt am %s\n",__DATE__);
562 fprintf(stat_output, "auf dem Gitter [%f; %f] x  [%f; %f]\n\n",lo[0],hi
      [0],lo[1],hi[1]);
563 #ifdef Refine
564 fprintf(stat_output,"Algorithmus mit Verfeinerung: Ja.\n");
565 fprintf(stat_output,"Verfeinerungstoleranz: %f\n",RTOL);
566 fprintf(stat_output,"Sicherheitsfaktor Rho: %f\n",RHO);
567 #else
568 fprintf(stat_output,"Algorithmus mit Verfeinerung: Nein.\n");
569 #endif
570
571 fprintf(stat_output,"Anzahl der Iterationen : %d\n",k);
572 fprintf(stat_output,"Anzahl der Knoten im Gitter:%d\n",g->max_vertex);
573 double restsek;
```

```
574 restsek= fmod(zeit,60);
575 fprintf(stat_output,"Anzahl Aufrufe der Trapezformel:%d\n",aufrufe);
576 fprintf(stat_output,"Rechenzeit in Minuten und Sek. : %d Min und %2.2fs\n"
        ,(int)(zeit/60),restsek);
577
578 /*Statistikdatei schliessen*/
579 fclose(stat_output);
580
581
582 /*Teil 2: Numerische Simulation des Modells*/
583 printf("Bitte beliebige Taste druecken\n");
584 printf("zur Ausfuehrung der numerischen Simulaton!\n");
585 pause=getchar();
586 printf("\n------------------Numerische Simulationen-------------------\n")
        ;
587 printf("|.................bei Eingabe eines Anfangswerts x0........|\n");
588 printf("-------------------------------------------------------------\n");
589
590 char simfile [100];
591 char delta [100];
592 char kappa [100];
593 sprintf(delta,"delta_%.2lf",p.delta);
594 double anfangswert[2];
595 double sim_val, exact_val,num_val;
596
597
598 /*Zufallszahlengenerator von Marsaglia initialisieren*/
599 long idum=(-13);
600
601 double input;
602 double temp =0.;
603 int answer=1;
604 int counter=1;
605 unsigned int method=1;
606
607 while(answer==1)
608 {
609 strcpy(simfile,"Simu_");
610 strcat(simfile,delta);
611 sprintf(kappa,"_kappa_%1d_%1d",p.kappa,counter);
612 strcat(simfile,kappa);
613 /*Anhaengen der .asc Endung*/
614 strcat(simfile,".asc");
615
616 /*Anfangswert per Eingabe festlegen*/
617 printf("Eingabe Startkapitalstock x1\n");
618 scanf("%lf",&input);
```

```
619
620 anfangswert[0]= input;
621 printf("Eingabe Startwert technischer Fortschritt x2\n");
622 scanf("%lf",&input);
623 anfangswert[1]= input;
624 if(p.delta==1 &&(p.kappa==2|| p.kappa==0))
625 {printf("Simulation der Kontrolle? (1: nach DP, 2: nach
       Perturbationsmethode)\n");
626 scanf("%d",&method);}
627
628 /*Durchschnittswert initialisieren*/
629 sim_val =0.;
630 printf("%d Simulationen werden nun ausgefuehrt:\n", anz_sim);
631 for(i=1;i<=anz_sim;i++)
632 {
633 printf(".");
634 if(fmod(i,100)==0)
635 printf("\n");
636
637 temp = simulation(anfangswert,g,&idum,simfile,method);
638 sim_val+= temp;
639 }
640 /*Mittelwertbildung*/
641 sim_val= sim_val/anz_sim;
642
643 /*Gitterwert abfragen*/
644 num_val = value(g,anfangswert,&flag);
645 printf("\n\nSimulationsergebnisse:\n");
646 printf("_____________________________\n");
647
648 if(p.delta==1.0 && p.kappa==0)
649 {
650 exact_val= V_exact(anfangswert);
651 printf("Exakter Wert:..............%.4f\n",exact_val);
652 }
653 printf("Simulierter Wert:...........%.4f\n",sim_val);
654 printf("Numerischer Wert:...........%.4f\n\n",num_val);
655 if(p.delta==1.0 && p.kappa==0)
656 printf("Unterschied ex./num. Wert:...%.4f\n",fabs(exact_val-num_val));
657
658 printf("Unterschied sim./num. Wert:...%.4f\n",fabs(num_val-sim_val));
659 printf("Werte der Trajektorie & Kontrolle in '%s' gespeichert!\n\n",
       simfile);
660
661 printf("\nweitere Simulation ausfuehren ( 1: JA, 0: NEIN) ?");
662 scanf("%d",&answer);
663 counter++;
```

```
664 }
665
666 /* Loeschen des Gitters*/
667 delete_grid(g);
668 return 0;
669
670 }
```

Anhang C

Material auf der beiliegenden CD–ROM

Die der Arbeit beiliegende CD-ROM enthält alle benutzten und angesprochenen Routinen im Verzeichnis **Programme**, alle in der Arbeit verwendeten Abbildungen im Verzeichnis **Bilder**, die komplette Arbeit im pdf-Format im Verzeichnis **Diplomarbeit** sowie die Internetreferenzen des Literaturverzeichnisses im Ordner **Internetquellen**.
Im Ordner **Anwendung** sind die Programme so abgelegt, dass ihre gegenseitigen Abhängigkeiten (z.B. Output der C Routine ist Input für MATLAB Programm) berücksichtigt sind.

Literaturverzeichnis

[1] BAUER, Heinz: *Maß- und Integrationstheorie*. De Gruyter Lehrbuch, 1992. – 2. Ausgabe

[2] BECKER, Stephanie: *Numerische Lösung dynamischer Gleichgewichtsmodelle*, Universität Bayreuth, Diplomarbeit, 2005. – Diplomarbeit

[3] BECKER, Stephanie ; GRÜNE, Lars ; SEMMLER, Willi: Comparing accuracy of second-order approximation. In: *Computational economics* 30 (2007), S. 65–91

[4] BLANCHARD, Olivier ; ILLING, Gerhard: *Makroökonomie*. Person Studium, 2006. – 4. Ausgabe

[5] COLLARD, Fabrice ; JUILLARD, Michel: Accuracy of stochastic perturbation methods: The case of asset pricing models. In: *Journal of Economic Dynamics and Control* 25 (2001), S. 979–999

[6] ENCYCLOPEDIA, Wikipedia the f.: *Frank P. Ramsey*. `http://en.wikipedia.org/wiki/Frank_P._Ramsey`, Abruf: 20.03.2008

[7] FEICHTINGER, Gustav ; HARTL, Richard: *Optimale Kontrolle ökonomischer Prozesse*. de Gruyter, 1986

[8] FRIEDRICH, H. ; LANGE, C.: *Stochastische Prozesse in Natur und Technik*. Verlag Harri Deutsch, 1999

[9] GONG, Gang ; SEMMLER, Willi: *Stochastic Dynamic Macroeconomics: Theory, Numerics and Empirical Evidence*. 2004

[10] GRÜNE, Lars: An adaptive grid scheme for the discrete Hamilton–Jacobi–Bellmann equation. In: *Numerische Mathematik* 75 (1997), S. 319–337

[11] GRÜNE, Lars: *Numerische Dynamik von Kontrollsystemen*. SS 2004. – Vorlesungsskript

[12] GRÜNE, Lars: *Der Gittergenerator* GRIDGEN. SS 2007. – Anhang zum Vorlesungsskript

[13] GRÜNE, Lars: *Stochastische Dynamische Optimierung*. SS 2007. – Vorlesungsskript

[14] GRÜNE, Lars ; SEMMLER, Willi: Using dynamic programming with adaptive grid scheme for optimal control problems in economics. In: *Journal of Economic Dynamics & Control* 28 (2004), S. 2427–2456

[15] JUDD, Kenneth: Approximation, perturbation and projection methods in economic analysis. In: *Handbook of Computational Economics* Bd. 1. Elsevier Science B.V., 1996, Kapitel 12

[16] KARVELAGEN, Erwin: *An elementary Ramsey growth model*. http://www.gams.com/~erwin/micro/ramsey.pdf

[17] KAUFMANN, Dr.rer.pol. R.: *Einfach lernen ! Makroökonomie*. http://studentensupport.de/store/product_103_download.aspx

[18] KLEIN, Paul: *Homepage Paul Klein*. http://www.ssc.uwo.ca/economics/faculty/klein/personal/codes.htm, Abruf: 05.05.2008

[19] ÖHRLEIN, Stefanie: *Numerische Berechnung der optimalen Wertefunktion mit Approximation zweiter Ordnung*, Universität Bayreuth, Diplomarbeit, 2007. – Diplomarbeit

[20] PITTSBURGH, Universität: *The Frank Ramsey Collection*. http://www.library.pitt.edu/libraries/special/asp/ramsey.html

[21] RAMSEY, Frank: A mathematical theory of saving. In: *Economic Journal* 38 (1928), Nr. 152, S. 543–559

[22] SCHMITT-GROHÉ, Stephanie: *Perturbation Methods for the Numerical Analysis of DSGE Models*. Februar 2005. – Vorlesungsskript

[23] SCHMITT-GROHÉ, Stephanie ; URIBE, Martin: *Matlab Codes for Second Order Approximation*. http://www.econ.duke.edu/~uribe/2nd_order.htm. Version: 2004

[24] SCHMITT-GROHÉ, Stephanie ; URIBE, Martin: Solving dynamic generel equilibrium models using a second-order approximation to the policy function. In: *Journal of Economic Dynamics & Control* 28 (2004), S. 755–775

[25] UHLIG, Harald ; TAYLOR, John: Solving Nonlinear Stochastic Growth Models. In: *NBER Working Paper Series* (1989)

[26] VERMEYLEN, Koen: *The Neoclassical Growth Model and Richardian Equivalence*. http://studentensupport.de/store/product_180.aspx

[27] VERMEYLEN, Koen: *The Stochastic Growth Model*. http://studentensupport.de/store/product_182.aspx

[28] VETTERLING, W. T. ; TEUKOLSKY, S. A. ; PRESS, W. H. ; FLANNERY, B. P.: *Numerical Recipes Example Book (C)*. Cambridge University Press, 1994. – 2. Ausgabe

[29] VETTERLING, W. T. ; TEUKOLSKY, S. A. ; PRESS, W. H. ; FLANNERY, B. P.: *Numerical Recipes in C - The Art of Scientific Computing*. Cambridge University Press, 2002. – 2. Ausgabe

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.
Wörtliche und sinngemäße Zitate habe ich als solche gekennzeichnet.
Diese Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Bayreuth, den 29. Juli 2008

Christian Spieler