

Fakultät für Informatik
der Technischen Universität München

Masterarbeit in Finance & Information Management

Integration von Adaptivität in einen Dünngitterlöser für Basket-Optionen

Integration of Adaptivity into a Sparse Grid Solver for Basket Options

Autor:	Alexander Friedrich Heinecke
Prüfer:	Prof. Dr. Hans Joachim Bungartz Prof. Dr. Helmut Krcmar
Betreuer:	Dr. Dirk Pflüger Dipl.-Tech. Math. Stefanie Schraufstetter
Abgabedatum:	15.02.2011

Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this master thesis only supported by declared resources.

München, den 15.02.2011

Alexander Heinecke

Zusammenfassung

Die Bestimmung des Preises einer Basket Option (einer Option auf mehrere Underlyings) ist eine numerisch anspruchsvolle Aufgabe, da keine geschlossene Lösung der dazugehörigen Black-Scholes Gleichung existiert. Sie kann zum Einen über Monte Carlo Simulationen erfolgen, was aber Nachteile in der Konvergenzgeschwindigkeit und der Bestimmung von Risikomaßen mit sich bringen kann. Zum Anderen kann die multi-dimensionale Black-Scholes Gleichung, welche eine parabolische partielle Differentialgleichung ist, auch direkt gelöst werden. Da die Anzahl der Dimensionen in den interessanten Fällen größer 2-3 ist, führt eine direkte Diskretisierung mit klassischen regulären Gitteransätzen zu Lösungsvorgehen, die nicht in realistischer Zeit und mit angemessenem Speicherverbrauch ausgeführt werden können. Aus diesem Grund wird in der vorliegenden Arbeit die Methode der Dünne Gitter angewendet, um die Black-Scholes zu diskretisieren, welche mit deutlich weniger Gitterpunkten auskommt. In der Vergangenheit hat sich allerdings herausgestellt, dass äquidistante, reguläre Dünne Gitter bisweilen zu wenig Gitterpunkte an den benötigten Stellen des zu diskretisierenden Gebiets investieren. Daher werden in dieser Arbeit voll-adaptive Dünngitterstrukturen behandelt, die eine Verfeinerung des Gebiets an solchen Stellen erlauben. Ferner wird eine logarithmische Koordinatentransformation durchgeführt, um die Kondition des zu lösenden Systems zu verbessern. Darüber hinaus wird ein effizienter Ansatz zur Parallelisierung der betrachteten Algorithmik eingeführt. Mittels dieser Erweiterungen und Anpassungen lassen sich Optionen mit bis zu fünf Underlyings ohne Probleme und effizient auf heutigen, parallelen Workstation Systemen sehr genau bewerten.

Abstract

The evaluation of a basket option's price (an option on several underlyings) is a numerical challenging task, since there is no closed-form solution of the corresponding Black-Scholes equation. On the one hand the option's value can be calculated by a Monte Carlo Simulation. However you might face some difficulties in calculating hedging measurements (the Greeks) and in terms of convergence rates. On the other hand you can achieve a direct solution of the Black-Scholes's parabolic partial differential equation (PDE). Here, normally the number of required dimensions is higher than 2 or 3. This permits a solution on full grids since this would lead to an enormous computing time and memory consumption. Due to these limitations this thesis uses Sparse Grids in order to discretize the Black-Scholes PDE directly. Sparse Grids have significantly less grid points in comparison to full grids. Unfortunately, several tests have proofed that regular Sparse Grids spend too few grid points on needed sub-domains, so adaptive Sparse Grids methods are used. In addition, a logarithm-based coordinate transformation is applied in order to improved the condition of the needed system of linear equations. Moreover, an efficient parallelization of the solver is developed, too. With these enhancements and extensions, the evaluation of basket options with up to five underlyings is possible at high precision and nearly perfect speed-ups on nowadays parallel workstation computers.

Danksagung

Ich möchte allen Leuten danken, die mich unterstützt haben, diese Masterarbeit anzufertigen.

Zuerst richte ich den Dank an meine beiden Betreuer Stefanie Schraufstetter und Dirk Pflüger. Ich danke ihnen für zahlreiche Tipps und Verbesserungsvorschlägen zu dieser Arbeit.

Ferner möchte ich Carsten Trinitis vom Lehrstuhl für Rechnertechnik und Rechnerorganisation danken, da er mir die Möglichkeit gab, meine Algorithmen auf Maschinen an diesem Lehrstuhl auszuführen.

Besonderer Dank geht an meine Eltern und meine Freundin Bianca, die mich während meines Studiums großartig unterstützt haben.

Inhaltsverzeichnis

1	Einleitung	1
2	Optionsbewertung mit Dünnen Gittern	3
2.1	Einführung in die Optionspreistheorie	3
2.1.1	Das Black-Scholes Modell	5
2.1.2	Das multi-Asset Black-Scholes Modell	7
2.2	Direkte Lösung der Black-Scholes PDE mittels Finiten Elementen	9
2.2.1	Kartesische Koordinaten	11
2.2.2	Log-transformierte Koordinaten	13
2.3	Diskretisierung der Zeit	15
2.4	Dünne Gitter - Ansatzraum des Lösers	18
2.4.1	Gitterkonstruktion	18
2.4.2	Up/Down Schema für Matrizenfunktionale	25
2.4.3	Aufspaltung der benötigten Operationen	29
2.5	Strategien für spezielle Optionen	31
2.5.1	Randbehandlung bei der europäischen Option	31
2.5.2	Freie Dimensionen für die asiatische Option	33
2.5.3	Nebenbedingungen am Beispiel der amerikanischen Option	35
3	Adaptivität bei der Lösung der Black-Scholes PDE	37
3.1	Allgemeines zur Verwendung von adaptiven Gitterstrukturen	37
3.2	Mögliche Adaptivitätskriterien beim Lösen der Black-Scholes Gleichung	42
3.2.1	Initiale Verfeinerung	44
3.2.2	Adaptivität während des Lösens	51
3.3	Bewertung der Adaptivitätskriterien	52
3.3.1	Parameterannahmen und -einschränkungen	53
3.3.2	Parameterstudie anhand eines 2-dimensionalen Baskets	55
3.3.3	Verallgemeinerung auf 3-dimensionale Baskets	66
3.3.4	Höher-dimensionale Baskets	70
3.4	Analyse des iterativen Lösers	77
4	Parallelisierung des Black-Scholes PDE Lösers	79
4.1	Paralleles Up/Down Schema	80
4.2	Messung der Speed-Ups	83
5	Zusammenfassung und Ausblick	87

A	Monte Carlo Simulation für Europäische Basket-Optionen	89
A.1	Idee der Monte Carlo Simulation	89
A.2	MC C++ Code für Basket-Optionen	91

1 Einleitung

In den letzten Jahrzehnten hat die Komplexität des globalen Finanzmarktes stetig zugenommen. Dies ist zu einem großen Teil darauf zurückzuführen, dass die gehandelten Produkte immer ausgefallener und spezieller wurden. Einen großen Anteil der Derivate stellen aber immer noch Optionen dar. Die Bewertung dieser wird meist mit dem Modell von Black und Scholes durchgeführt. Dieses ist, solange es sich um eine (europäische)¹ Option auf nur eine Asset handelt, geschlossen lösbar. Werden aber Basket-Optionen, also Optionen auf mehrere Underlyings, betrachtet, muss der Preis dieser Optionen im Allgemeinen mit numerischen Werkzeugen ermittelt werden.

Am weitesten verbreitet ist hier die Monte Carlo Simulation, da sie einfach zu implementieren ist. Allerdings bietet sie nur schlechte Konvergenzeigenschaften ($\mathcal{O}(1/\sqrt{N})$ wobei N die Anzahl der verwendeten Pfade ist). Darüber hinaus ist die Bestimmung der Risikomaße, der so genannten Griechen, recht aufwendig. Davon abgesehen hat die Monte Carlo Simulation den Vorteil, dass ihre Implementierung unabhängig von der Anzahl der betrachteten Underlyings der Option ist.

Ein anderer Ansatz besteht darin, die Black-Scholes Gleichung, eine partielle Differentialgleichung (PDE), mittels bekannter Verfahren wie finiten Differenzen oder finiten Elementen, zu diskretisieren. Dadurch entsteht ein zu lösendes lineares Gleichungssystem. Durch dieses ist die numerische Lösung bekannt und die Griechen können einfach bestimmt werden. Zusätzlich sind deutlich bessere Konvergenzraten als im Falle der Monte Carlo Simulationen möglich. Allerdings ergibt sich beim Lösen von Basket-Optionen mit mehreren Assets (mehr als 3) die Herausforderung des **Fluchs der Dimensionalität**. Das bedeutet, dass die Anzahl der Freiheitsgrade (Gitterpunkte) exponentiell mit der Anzahl der benötigten Dimensionen zusammenhängt.

In der vorliegenden Masterarbeit wurde der zweite Ansatz auf Grund seiner Flexibilität und der aufgezeigten Vorteile zur Bewertung von Basket-Optionen mit Hilfe des Modells von Black und Scholes gewählt. Um den bereits dargestellten massiven Nachteil der Dimensionalität beim direkten Lösen von höher- & hochdimensionalen PDEs abzumildern, werden allerdings keine klassischen vollen, regulären Gitterstrukturen, sondern die so genannten **Dünnen Gitter** verwendet.

¹Es handelt sich bei der europäischen Option um ein recht einfaches Finanzderivat. Für kompliziertere Optionen kann selbst der ein-Asset Fall nicht geschlossen lösbar sein.

Es hat sich in den letzten zwei Jahrzehnten gezeigt, dass reguläre Dünne Gitter eine erhebliche Verringerung der Gitterpunkte / Freiheitsgrade bei quasi gleich bleibender Lösungsgenauigkeit erlauben. Allerdings setzt dies voraus, dass die betrachteten Funktionen hinreichend „glatt“ sind. Das bedeutet, dass sich keine Knicke oder Sprünge im Funktionsverlauf befinden dürfen. Diese Eigenschaft wird von der Black-Scholes PDE allerdings nicht erfüllt, da die Payoff-Funktion einer Option in der Regel Knick(e) beinhaltet. Um auch dieses Problem bei der direkten Lösung in den Griff zu bekommen, werden nicht reguläre sondern adaptive Dünngitterstrukturen verwendet. So können die Freiheitsgrade des Gitters genau an den Stellen eingesetzt werden, an denen sie benötigt werden, an denen die Glattheitsannahmen verletzt werden.

Zunächst soll in Kapitel 2 das mathematische Fundament eines Black-Scholes Lö-sers auf Basis von finiten Elementen mit Dünnen Gittern beschrieben werden. Anschließend werden in Abschnitt 3 zahlreiche Basket-Optionen mit Hilfe des im-plementierten Lö-sers bewertet und die Kalibrierung der Adaptivitäts-Parameter der Dünnen Gitter aufgezeigt. Da in der heutigen Zeit der Parallelrechner serielle Algorithmen benachteiligt sind, wird in Kapitel 4 vorgestellt, wie der implementier-te Lö-ser parallelisiert worden ist, um die Rechenleistung aktueller Systeme besser auszunutzen.

2 Optionsbewertung mit Dünnen Gittern

Wie bereits in der Einleitung angeklungen ist, stellt die direkte Lösung der Black-Scholes Gleichung einen flexibleren Ansatz dar, als die Lösung mittels Monte Carlo Simulationen. Der nun folgende Abschnitt wird die direkte Lösung der Black-Scholes PDE mit Hilfe von adaptiven Dünnen Gittern vorstellen. Hierbei soll das Lösungsverfahren von mehreren Seiten aus beleuchtet werden: Nach einer kurzen Einführung in die Optionspreistheorie werden von zwei Varianten der Black-Scholes Gleichung, der klassischen und der log-transformierten Formulierung, die Finite Elemente Diskretisierungen eingeführt. Anschließend wird diese durch den Ansatzraum der Dünnen Gitter konkretisiert und ein Ausblick gegeben, wie adaptive Dünne Gitter zur Lösung der Black-Scholes Gleichung mit deutlich weniger Freiheitsgraden eingesetzt werden können. Abgerundet wird dieses Kapitel durch die Betrachtung von Implementierungsdetails der europäischen und asiatischen Optionen.

2.1 Einführung in die Optionspreistheorie

Die Option ist ein Finanzmarktinstrument. Hierbei erwirbt der Käufer das Recht, ein bestimmtes Underlying zu einem bestimmten Ausübungspreis K , dem Strike, in der Zukunft $t = T$ zu kaufen (in diesem Fall spricht man von einem *Long Call*) bzw. zu verkaufen (hier wird die Option als *Long Put* bezeichnet). Aus der Sicht des Verkäufers gilt Folgendes: Wird das Recht, ein Underlying in der Zukunft zu erwerben, verkauft, so handelt es sich um einen *Short Call*, bzw. im Falle des Rechtes der Verkaufs um einen *Short Put*. Diese Unterscheidung wird allerdings erst wichtig, wenn man Gewinne des Käufers und des Verkäufers betrachtet. In dieser Arbeit steht aber die Bewertung von Optionen im Vordergrund, weswegen es ausreicht, eine Perspektive (Käufer oder Verkäufer) genauer zu betrachten. Es wird die Sichtweise des Käufers angenommen. Somit können die Bezeichnungen der Optionen an die klassischen Kurz-Bezeichnungen angepasst werden: Der *Long Call* wird zum **Call** und der *Long Put* zum **Put**.

Für den Preis V einer Option mit dem Strike K gilt zu einem Zeitpunkt T in der Zukunft auf ein Underlying mit Kurs S :

$$V(T, S) := \begin{cases} \max\{K - S_T, 0\}, & \text{falls Put Option,} \\ \max\{S_T - K, 0\}, & \text{falls Call Option.} \end{cases} \quad (2.1)$$

In Abbildung 2.1 werden die Auszahlungsfunktionen für Call und Put dargestellt. Aufgrund ihrer Erscheinung spricht man hier umgangssprachlich auch von der sogenannten *Hockeystick-Funktion*. An diesen Graphen kann gut erkannt werden, wann es Sinn macht, das Recht der Option auszuüben. Für den Call gilt: Liegt der Kurs des Underlying zum Zeitpunkt $t = T$ unterhalb des vereinbarten Strikes K , so ist dieser Call als wertlos zu betrachten, da das Underlying am Markt günstiger als zum garantierten Preis K erworben werden kann. Ist der Kurs allerdings höher als K , so bietet die Ausübung der Option einen Vorteil. Dieser wird als Payoff bezeichnet.

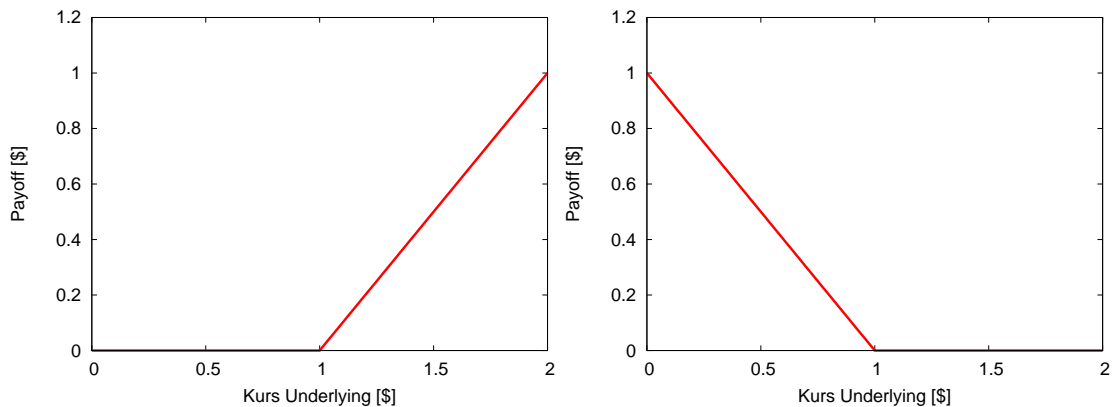


Abbildung 2.1: Beispiel für Payoff-Funktionen einer Call (links) und einer Put (rechts) Option. Für beide Optionen gilt Strike $K = 1$. Der Zeitpunkt dieser Betrachtung ist $t = T$.

Die Optionen aus Abbildung 2.1 werden als *europäische* Optionen bezeichnet. Europäische Optionen können nur zum Zeitpunkt $t = T$ ausgeübt werden und hängen nur vom Kurs des Underlyings zum Zeitpunkt $t = T$ ab. Zudem existieren zahlreiche andere Erscheinungsformen der Optionen. Die wichtigsten sind die *amerikanische*, *asiatische* und die *Bermuda*-Option. Die amerikanische Option ist zu jedem Zeitpunkt ausübbar, während die asiatische Option nur zum Zeitpunkt $t = T$ ausübbar ist. Hier wird allerdings nicht der aktuelle Kurs, sondern ein Mittelwert über die gesamte Laufzeit der Option als Bewertungsgrundlage herangezogen. Bei der Bermuda-Option handelt es sich um eine Mischform der europäischen und amerikanischen Optionen: Eine Bermuda-Option ist zu mehreren Zeitpunkten während der Laufzeit, gewöhnlich zu einem monatlichen Stichtag, ausübbar. In dieser Arbeit liegt der Fokus jedoch auf europäischen Optionen. Es wird ersichtlich, dass der hier entwickelte Löser mit kleineren Anpassungen sehr schnell zur Bewertung von

amerikanischen und Bermuda-Optionen eingesetzt werden kann. Ebenfalls werden implementierte Erweiterungen vorgestellt, die die Bewertung von asiatischen Optionen erlauben.

Aus dieser kurzen Einführung wird klar, dass es sich bei Optionen um Finanzderivate mit deutlichem spekulativen Charakter handelt. Optionen sind ein Werkzeug, um Kursschwankungen des Underlyings im Bereich des Strikes abzubilden. Dies wird als Hebeleffekt bezeichnet. Neben der Spekulation können Optionen aber auch zur Absicherung von Geschäften, dem so genannten *Hedging* eingesetzt werden. Ein mögliches Beispiel sind hier exportierende Unternehmen, die sich somit einen bestimmten Wechselkurs einer Fremdwährung garantieren wollen. Wie und in welchen Varianten Optionen zur Spekulation oder zum Hedging verwendet werden können, ist allerdings nicht Gegenstand dieser Arbeit. Hier steht die Bewertung im Mittelpunkt. Bei der Bewertung handelt es sich um die Fragestellung, wie viel eine Option zu einem bestimmten Zeitpunkt vor dem Ausübungszeitpunkt wert ist. Dies ist der Preis, den ein Käufer einer Option bezahlen muss.

Bis jetzt wurden Optionen basierend auf einem Underlying eingeführt. Es ist aber auch möglich, Optionen auf Körbe von Underlyings zu erwerben, z.B. so genannte Index-Optionen, die einen kompletten Index wie den DAX-30¹ abbilden. In diesem Fall spricht man von Basket-Optionen. Anders als im Fall nur eines Underlyings existiert keine geschlossene Lösung für die Bewertung einer Basket-Option, sie muss also numerisch erfolgen.

Somit kann die Fragestellung dieser Masterarbeit präzisiert werden: **Was ist der Wert einer europäischen Basket-Option?** Dieser Fragestellung soll in den nächsten Abschnitten unter der Verwendung des Bewertungsmodells von Black und Scholes sowie von adaptiven Dünnen Gittern nachgegangen werden.

2.1.1 Das Black-Scholes Modell

Bereits Anfang der siebziger Jahre gelang es Fischer Black, Myron Scholes und Robert Merton ein Modell zu entwickeln, welches die Bewertung einer Option erlaubt. Inzwischen ist das so genannte Black-Scholes Modell zum Standardinstrument der Optionsbewertung am Finanzmarkt geworden. Im Jahre 1997 erhielten Scholes und Merton für dieses Modell den Nobelpreis in Wirtschaftswissenschaften. Black wurde diese Ehre nicht mehr zuteil, da er bereits 1995 verstorben war.

Dem Modell liegen folgende Annahmen zu Grunde, nach [7]:

Annahme 1: Es existiert ein vollkommener, vollständiger Kapitalmarkt. Dies impliziert Arbitrage-Freiheit und keine Transaktionskosten, sowie keine Be-

¹DAX: Deutscher Aktien Index; DAX-30 enthält die 30 größten deutschen Aktiengesellschaften.

schränkung von Leerverkäufen.² Durch ein eindeutiges Martingalmaß können die Preise eines jeden Portfolios jederzeit eindeutig bestimmt werden.

Annahme 2: Der Kurs des Underlyings folgt einer geometrischen Brownschen Bewegung.

Annahme 3: Die Underlyings schütten keine Dividenden aus.

Annahme 4: Es existiert ein konstanter Zinssatz r , zu dem zu jedem Zeitpunkt Geld in beliebiger Höhe angelegt und aufgenommen werden kann.

Diese Annahmen stellen starke Einschränkungen der echten Welt dar, helfen aber, das Problem der Optionsbewertung verständlich zu modellieren. Es existieren jedoch Modellerweiterungen, mit deren Hilfe die Annahmen 3 und 4 entfernt werden können.

Nun soll die Modellierung der Assetkurse, Annahme 2, genauer betrachtet werden. Der stochastische Prozess, welcher zur Abbildung der Kursentwicklung im Modell gewählt wurde, ist die geometrische Brownsche Bewegung³, welche durch folgende stochastische Differentialgleichung definiert ist:

$$dS(t) = \mu_{GBM}S(t)dt + \sigma_{GBM}S(t)dW(t) \quad (2.2)$$

mit μ_{GBM} als Drift des stochastischen Prozesses und σ_{GBM} gibt die Standardabweichung an. $W(t)$ ist hierbei eine standard Brownsche Bewegung (auch Wiener Prozess genannt) mit unabhängigen und normalverteilten Inkrementen, es gilt also:

$$W(t + \tau) - W(t) \sim N(0, \sqrt{\tau}). \quad (2.3)$$

Nach [7] kann nun unter Anwendung des Lemma von Ito und der Annahme eines vollkommenen Kapitalmarktes, insbesondere der Arbitragefreiheit des Marktes, die Black-Scholes Gleichung zur Bewertung einer Option auf ein Asset formuliert werden:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2S^2\frac{\partial^2V}{\partial S^2} + \mu S\frac{\partial V}{\partial S} - rV = 0. \quad (2.4)$$

Anzumerken ist, dass $V(t, S), t \in [0, T]$ den Preis einer europäischen Call Option angibt und r ist die risikolose Verzinsung. μ entspricht nicht μ_{GBM} aus der geometrischen Brownschen Bewegung. Falls keine Dividenden modelliert werden, ist μ normalerweise gleich r zu wählen, was aber nicht notwendig ist und auch bei den späteren numerischen Tests in Kapitel 3 teilweise nicht gemacht wird.

²Unter einem Leerverkauf versteht man am Finanzmarkt den Verkauf eines Gutes, welches man nicht besitzt. Indem man diese Position in der Zukunft glattstellt, kann ein Gewinn erwirtschaftet werden.

³GBM für *geometric Brownian motion*

2.1.2 Das multi-Asset Black-Scholes Modell

Das im letzten Abschnitt betrachtete ein-Asset Modell ist numerisch weniger interessant, da hier eine geschlossene Lösung existiert (falls $\mu = r$ gewählt wurde). Lediglich der Verständlichkeit wegen wurden die Grundlagen anhand der ein-dimensionalen Black-Scholes Gleichung beschrieben. Ab nun steht, wie auch eingehend schon angekündigt, das multi-Asset Black-Scholes-Modell im Mittelpunkt.

Die Black-Scholes Gleichung auf einen Korb von Assets lautet:

$$\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{i=1}^d \mu_i S_i \frac{\partial V}{\partial S_i} - rV = 0, \quad (2.5)$$

hierbei gibt, wie im ein-dimensionalen Fall, $V(t, \vec{S})$ den Preis der Option zum Zeitpunkt t an. Für \vec{S} gilt $\vec{S} = (S_1, \dots, S_d) \in \mathbb{R}_+^d$; die $\rho_{i,j}$ sind die Korrelationen zwischen S_i und S_j .

In dieser Arbeit soll das klassische europäische Basket gelöst werden. Dieses besitzt als Auszahlungsfunktion das arithmetische Mittel der Kurse zum Ausübungszeitpunkt:

$$V(T, S) := \begin{cases} \max\{K - \frac{1}{d} \sum_{i=1}^d S_i(T), 0\}, & \text{falls Put Option,} \\ \max\{\frac{1}{d} \sum_{i=1}^d S_i(T) - K, 0\}, & \text{falls Call Option.} \end{cases} \quad (2.6)$$

Allerdings sind hier auch andere Auszahlungsfunktionen denkbar. Es sei jedoch angemerkt, dass im Falle des geometrischen Mittels auch die multi-Asset Black-Scholes Gleichung geschlossen lösbar ist. Dies ist nach [33] möglich, da durch die multiplikative Verknüpfung der einzelnen stochastischen Prozesse einfach *ein* gemeinsamer stochastischer Prozess hergeleitet werden kann. Damit kann die multi-Asset Betrachtung auf eine ein-Asset Betrachtung zurückgeführt werden. Für alle anderen Fälle werden meist Monte Carlo Simulationen zur Bewertung eingesetzt. In Anhang A ist diese Vorgehensweise für europäische Call und Put Basket-Optionen beschrieben.

Leider ist das Lösen von Gleichung (2.5) schlecht konditioniert, da diese PDE variable Koeffizienten aufweist. Diese resultieren aus der geometrischen Brownschen Bewegung. Durch die geschlossene Lösung von Gleichung (2.2),

$$S(t) = S(0) \exp \left[\left(\mu_{GBM} - \frac{1}{2} \sigma_{GBM}^2 \right) t + \sigma_{GBM} W(t) \right], \quad (2.7)$$

kann die schlechte Kondition verdeutlicht werden: Die Volatilität der Preise der Underlyings wächst exponentiell mit der Zeit t . Es existieren mehrere Möglichkeiten, um dieses Problem abzumildern. So können z.B. nicht-äquidistante Gitter verwendet werden, um das Wachstum der Kurse (positive Koordinatenrichtung) abzubilden. Hierbei werden die Abstände zwischen die Gitterpunkten mit steigendem

Kurs exponentiell größer. Ein solches Verfahren wurde z.B. in [21] vorgeschlagen. In [23] werden noch weitere Ansätze vorgestellt. So können zum Einen logarithmische Koordinaten mit $S_i = \log S_i$ verwendet werden, um die variablen Koeffizienten zu eliminieren. Zum Anderen ist es möglich, durch erweiterte Transformation die Black-Scholes auf die klassische Wärmeleitungs-Gleichung zurückzuführen, siehe z.B. [13].

Da der hier entwickelte Löser allerdings in eine bestehende Toolbox integriert wird und auch später möglichst flexibel sein sollte, wird zusätzlich zur direkten Lösung von Gleichung (2.5) auf adaptiven Dünnen Gittern mit kartesischen Koordinaten die Lösung der Black-Scholes Gleichung mit logarithmischen Koordinaten vorgestellt. Der große Vorteil bei diesem Ansatz ist, dass die bestehenden Gitter weiterverwendet werden können, da die Gitterpunkte äquidistant sind und nur die Algorithmik an die veränderten Koordinaten angepasst werden muss. Die multi-Asset Black-Scholes Gleichung mit logarithmischen Koordinaten lautet:

$$\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{i=1}^d \left(\mu_i - \frac{1}{2} \sigma_i^2 \right) \frac{\partial V}{\partial S_i} - rV = 0. \quad (2.8)$$

Wie bereits erwähnt, liegt der Vorteil dieser Formulierung in der besseren Kondition der zu lösenden Gleichung. Dies wird auch im späteren Ergebnisteil deutlich: Es ist klar zu erkennen, dass zum Lösen von Gleichung (2.8) merklich weniger Iterationen als zum Lösen von Gleichung (2.5) benötigt werden.

Allerdings muss an dieser Stelle angemerkt werden, dass die Formulierung mit log-transformierten Koordinaten nicht nur Vorteile hat. So entsteht aus einem nur einseitigen unendlichen Gebiet (in Richtung der Kurse) in Gleichung (2.5) ein zu beiden Seiten unendliches Gebiet in Gleichung (2.8). Dies kann bei der Bewertung von Put Optionen im unteren Wertebereich zu Problemen führen. In Tabelle 2.1 werden die Vor- und Nachteile der beiden Gleichungen gegenübergestellt. Es sei bereits jetzt als Vorbemerkung erwähnt, dass die Vorteile der Gleichung mit log-transformierten Koordinaten deutlich deren Nachteile überwiegen und somit diese Variante verwendet werden sollte.

Vorteile	Nachteile
keine Transformation	
$[0, \infty]$ Gebiet	var. Koeffizienten
log-transformierte Koordinaten	
konst. Koeffizienten	$[-\infty, \infty]$ Gebiet

Tabelle 2.1: Vor- und Nachteile der betrachteten Lösungsverfahren der multi-Asset Black-Scholes Gleichung ohne Transformationen (links) und mit log-transformierten Koordinaten (rechts).

Gleichung (2.6) stellt die Endbedingung beim Lösen der multi-Asset Black-Scholes Gleichung dar. Allerdings wird zur numerischen Lösung einer PDE im Allgemei-

nen eine Anfangsbedingung benötigt. Diese wird als Ausgangspunkt verwendet, um die Lösung der PDE mittels intervallweiser Auswertung der Ableitungen zu approximieren. Im Falle der Black-Scholes Gleichung kann das benötigte Anfangswertproblem (AWP) durch eine Umkehrung der Zeit konstruiert werden. Damit wird die Endbedingung aus Gleichung (2.6) zur Anfangsbedingung. Mathematisch kann die Umkehrung der Zeit durch einen Vorzeichenwechsel beim Term mit der Ableitung nach der Zeit realisiert werden. Es folgt also mit $\tau = T - t$ für Gleichung (2.5):

$$\frac{\partial V}{\partial \tau} - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} - \sum_{i=1}^d \mu_i S_i \frac{\partial V}{\partial S_i} + rV = 0 \quad (2.9)$$

und für Gleichung (2.8):

$$\frac{\partial V}{\partial \tau} - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \frac{\partial^2 V}{\partial S_i \partial S_j} - \sum_{i=1}^d \left(\mu_i - \frac{1}{2} \sigma_i^2 \right) \frac{\partial V}{\partial S_i} + rV = 0. \quad (2.10)$$

2.2 Direkte Lösung der Black-Scholes PDE mittels Finiten Elementen

Nachdem im letzten Abschnitt die kontinuierlichen zu lösenden partiellen Differentialgleichungen eingeführt worden sind, soll nun deren Diskretisierung des Raumes im Mittelpunkt stehen. Im einfachsten Fall werden PDEs wie Gleichungen (2.5) und (2.8) mittels der so genannten Finiten Differenzen Methode gelöst. Hierbei werden die in der PDE enthaltenen Ableitungen mittels (mehrfacher) Differenzenquotienten an zuvor gewählten Stützstellen, den so genannten Gitterpunkten, approximiert. Allerdings entsteht durch dieses direkte Vorgehen auch ein Problem: Die Differenzenquotienten müssen an jeder Stützstelle in der kleinsten Maschenweite (so wird der Abstand zwischen zwei Stützstellen bezeichnet) berechenbar sein. Dies erzwingt im mehr-dimensionalen Fall eine sehr hohe Anzahl von Gitterpunkten, da diese exponentiell mit der Anzahl an Dimensionen wächst. Dies wird als **Fluch der Dimensionalität** bezeichnet. Somit kann die Finite Differenzen Methode nur mit einigen Tricks zur Lösung von höher- bzw. hoch-dimensionalen Problemen eingesetzt werden. Eine mögliche Vorgehensweise ist hier die Verwendung der so genannten Kombinationstechnik der Dünnen Gitter. Hierbei werden volle Gitter mit verschiedenen Maschenweiten (gröber und feiner) kombiniert, um Gitterpunkte einzusparen. Allerdings ist dieser Ansatz nicht Bestandteil dieser Arbeit und soll daher nicht weiter vertieft werden und es sei z.B. auf [6] verwiesen.

Aus diesen Gründen bedient sich diese Arbeit eines anderen Ansatzes zur Diskretisierung des Raumes: Der Methoden der Finiten Elementen (FEM). Die Ableitungen nicht wie bei den finiten Differenzen direkt diskretisiert, sondern durch die Transformation der PDE in die so genannte *schwache Form*. Hierbei werden deutlich schwächere Annahmen gefordert und die Basisfunktionen können frei gewählt

werden. Knapp formuliert bedeutet die schwache Form, dass die Gleichung nicht mehr an allen Punkten, sondern nur noch abgeschwächt, d.h. gemittelt, erfüllt werden muss.

Zur kurzen Einführung in die Methode der Finiten Elemente soll folgendes Modellproblem betrachtet werden:

$$Lv(\vec{x}) = f(\vec{x}), \quad (2.11)$$

wobei L ein beliebiger Differentialoperator, v die gesuchte Funktion und f eine (beliebige) rechte Seite sind. Die Lösung dieser Differentialgleichung wird auf folgendem Gebiet Ω betrachtet: $\Omega \in \mathbb{R}^d$. Die Funktion $v(\vec{x})$ kann folgendermaßen durch eine Basisrepresentation approximiert werden:

$$u(\vec{x}) := \sum_{i=1}^N u_i \phi_i(\vec{x}). \quad (2.12)$$

Die $u_i \in \mathbb{R}$ sind konstante Koeffizienten und die ϕ sind Funktionen von \vec{x} . Sie werden als *Basisfunktionen* oder auch *Ansatzfunktionen* bezeichnet. Die ϕ können quasi beliebig gewählt werden und sind für die Lösung festgelegt. Die Freiheitsgrade liegen in den Koeffizienten u_i . Ziel ist es, diese so zu bestimmen, dass gilt: $v \approx u$.

Nun kann der erste Schritt der Lösung mittels Finiten Elementen durchgeführt werden: Die Formulierung der schwachen Form der PDE. Hierzu wird, wie bereits erläutert, die Gleichheit aus Gleichung (2.11) nur noch in einer gemittelten Darstellung gefordert. Dies geschieht, indem diese Gleichung mit einer endlichen Menge von Testfunktion ψ_j mit $j = (1, \dots, N)$ multipliziert wird und über das Lösungsgebiet integriert wird. Es ergibt sich somit folgende schwache Form:

$$\int_{\Omega} Lu(\vec{x})\psi_j(\vec{x})d\vec{x} = \int_{\Omega} f(\vec{x})\psi_j(\vec{x})d\vec{x} \quad \forall j = (1, \dots, N). \quad (2.13)$$

Dies ist ein System mit N Gleichungen für die u_i Koeffizienten. Dieses System kann folgendermaßen umgeformt werden, mit der Voraussetzung, dass L linear ist:

$$\begin{aligned} \int_{\Omega} L \left(\sum_{i=1}^N u_i \phi_i(\vec{x}) \right) \psi_j(\vec{x}) d\vec{x} &= \int_{\Omega} f(\vec{x}) \psi_j(\vec{x}) d\vec{x} \quad \forall j = (1, \dots, N) \\ \sum_{i=1}^N u_i \int_{\Omega} L \phi_i(\vec{x}) \psi_j(\vec{x}) d\vec{x} &= \int_{\Omega} f(\vec{x}) \psi_j(\vec{x}) d\vec{x}. \end{aligned}$$

Hier wird der *Ritz-Galerkin* Ansatz gewählt, der folgende Wahl für die Testfunktionen trifft: $\psi = \phi$. Es ergibt sich somit:

$$\sum_{i=1}^N u_i \underbrace{\int_{\Omega} L \phi_i(\vec{x}) \phi_j(\vec{x}) d\vec{x}}_{:=a_{i,j}} = \underbrace{\int_{\Omega} f(\vec{x}) \phi_j(\vec{x}) d\vec{x}}_{:=b_j} \quad \forall j = (1, \dots, N). \quad (2.14)$$

Damit kann nun Gleichung (2.14) in Matrix-Schreibweise angegeben werden, wodurch ein lineares Gleichungssystem für die Koeffizienten u_i entsteht:

$$A\vec{u} = \vec{b} \quad (2.15)$$

Dieses System kann mit Standardverfahren für lineare Gleichungssysteme gelöst werden. Auf solche Verfahren soll an dieser Stelle nicht eingegangen werden. Es sei lediglich erwähnt, dass für kleinere Systeme ein direktes Verfahren wie die LR-Zerlegung infrage kommt. Für große Systeme ist ein iterativer Löser wie das Verfahren der konjugierten Gradienten (CG oder BiCGStab) mit Hilfe von Matrixfunktionalen die richtige Wahl. Es wurden allerdings einige Punkte nicht betrachtet, die bei einer Anwendung festgelegt werden müssen. So müssen für die Ansatz- und Testfunktionen konkrete Funktionen gewählt werden. Ebenfalls wurde noch nicht auf die Randbehandlung des betrachteten Gebiets eingegangen. Diese Festlegungen werden in Abschnitt 2.4 geklärt.

2.2.1 Kartesische Koordinaten

Die zuvor besprochene Methode der Finiten Elemente soll nun auf die Black-Scholes Gleichung in kartesischen Koordinaten angewendet werden. Da es sich hierbei jedoch um eine parabolische PDE handelt, muss, wegen der Zeitabhängigkeit, eine zu Gleichung (2.12) leicht veränderte Form zur Approximierung der Funktion $V(\tau, \vec{S})$ verwendet werden. Es gilt:

$$u(\tau, \vec{S}) := \sum_{n=1}^N u_n(\tau) \phi_n(\vec{S}), \quad (2.16)$$

Ebenfalls wird mit $\psi = \phi$ auch hier der Ritz-Galerkin Ansatz gewählt. Einsetzen der Black-Scholes Gleichung (vgl. Gleichung (2.9)) in die Definition aus Gleichung (2.13) liefert:

$$\int_{\Omega} \frac{\partial u(\tau, \vec{S})}{\partial \tau} \phi_m(\vec{S}) d\vec{S} = \int_{\Omega} \left(\frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} S_i S_j \frac{\partial^2 u(\tau, \vec{S})}{\partial S_i \partial S_j} + \sum_{i=1}^d \mu_i S_i \frac{\partial u(\tau, \vec{S})}{\partial S_i} - r u(\tau, \vec{S}) \right) \phi_m(\vec{S}) d\vec{S} \quad \forall m = (1, \dots, N).$$

Hierbei wurden vor dem Einsetzen alle Terme, die nicht von der Zeit abhängig sind, auf die rechte Seite gebracht. Nun sind umfangreiche aber handwerklich einfache

Umformungen notwendig. $\delta_{i,j}$ bezeichnet eine Funktion, die den Wert 1 hat, falls $i = j$ und sonst 0 ist. Es folgt:

$$\begin{aligned}
 & \sum_{n=1}^N \frac{\partial u_n(\tau)}{\partial \tau} \int_{\Omega} \phi_n(\vec{S}) \phi_m(\vec{S}) d\vec{S} = \\
 & -\frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \left((1 + \delta_{i,j}) \sum_{n=1}^N u_n(\tau) \int_{\Omega} S_i \frac{\partial \phi_n(\vec{S})}{\partial S_i} \phi_m(\vec{S}) d\vec{S} \right. \\
 & \left. + \sum_{n=1}^N u_n(\tau) \int_{\Omega} S_i S_j \frac{\partial \phi_n(\vec{S})}{\partial S_i} \frac{\partial \phi_m(\vec{S})}{\partial S_j} d\vec{S} \right) \\
 & + \sum_{i=1}^d \mu_i \left(\sum_{n=1}^N u_n(\tau) \int_{\Omega} S_i \frac{\partial \phi_n(\vec{S})}{\partial S_i} \cdot \phi_m(\vec{S}) d\vec{S} \right) \\
 & - r \sum_{n=1}^N u_n(\tau) \int_{\Omega} \phi_n(\vec{S}) \cdot \phi_m(\vec{S}) d\vec{S} \quad \forall m = (1, \dots, N).
 \end{aligned}$$

Während dieser zahlreichen Umformungen ist ein Teilschritt eine partielle Integration. Somit entstehen Funktionsauswertungen auf den Rändern des gewählten Gebiets Ω . In dem hier vorgestellten Löser der Black-Scholes Gleichung werden allerdings Dirichlet-Randbedingungen verwendet, die mit fortschreitender Zeit τ diskontiert werden. Aus diesem Grund existieren keine Freiheitsgrade auf den Rändern des betrachteten Gebiets. Dadurch entfallen die angesprochen Funktionsauswertungen; sie sind bereits in der vorherigen schwachen Form nicht mehr enthalten. Zusätzlich können identische Integrale ausgeklammert werden, um die Anzahl der auszuwertenden Integrale zu verringern. Damit ergibt sich folgende schwache Formulierung der multi-Asset Black-Scholes Gleichung:

$$\begin{aligned}
 & \sum_{n=1}^N \frac{\partial u_n(\tau)}{\partial \tau} \int_{\Omega} \phi_n(\vec{S}) \phi_m(\vec{S}) d\vec{S} = \\
 & \sum_{n=1}^N u_n(\tau) \left[\sum_{i=1}^d \underbrace{\left(\mu_i - \sum_{j=1}^d \left(\frac{1}{2} \sigma_i \sigma_j \rho_{i,j} (1 + \delta_{i,j}) \right) \right)}_{\nu_i} \int_{\Omega} S_i \frac{\partial \phi_n(\vec{S})}{\partial S_i} \phi_m(\vec{S}) d\vec{S} \right. \\
 & - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \int_{\Omega} S_i S_j \frac{\partial \phi_n(\vec{S})}{\partial S_i} \frac{\partial \phi_m(\vec{S})}{\partial S_j} d\vec{S} \\
 & \left. - r \int_{\Omega} \phi_n(\vec{S}) \cdot \phi_m(\vec{S}) d\vec{S} \right] \quad \forall m = (1, \dots, N). \quad (2.17)
 \end{aligned}$$

Gleichung (2.17) entspricht nun Gleichung (2.14) aus der Betrachtung des obigen Modellproblems. Allerdings ist sie in diesem Fall deutlich komplexer. Aus diesem

Grund soll die Behandlung der schwachen Form der Black-Scholes Gleichung mit einer Matrix-Formulierung abgeschlossen werden:

$$\begin{aligned}
 A\dot{\vec{u}}(\tau) &= \sum_{i=1}^d \nu_i \cdot F\vec{u}(\tau) - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \cdot E\vec{u}(\tau) - r \cdot A\vec{u}(\tau) \\
 A\dot{\vec{u}}(\tau) &= \underbrace{\left[\sum_{i=1}^d \nu_i \cdot F - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \cdot E - r \cdot A \right]}_{:=L} \cdot \vec{u}(\tau), \quad (2.18)
 \end{aligned}$$

mit den Matrizen:

$$\begin{aligned}
 A &:= (a_{n,m}) \in \mathbb{R}^{N \times N} \quad \text{mit} \quad a_{n,m} := \int_{\Omega} \phi_n(\vec{S}) \phi_m(\vec{S}) d\vec{S}, \\
 E &:= (e_{n,m}) \in \mathbb{R}^{N \times N} \quad \text{mit} \quad e_{n,m} := \int_{\Omega} S_i S_j \frac{\partial \phi_n(\vec{S})}{\partial S_i} \frac{\partial \phi_m(\vec{S})}{\partial S_j} d\vec{S}, \\
 F &:= (f_{n,m}) \in \mathbb{R}^{N \times N} \quad \text{mit} \quad f_{n,m} := \int_{\Omega} S_i \frac{\partial \phi_n(\vec{S})}{\partial S_i} \phi_m(\vec{S}) d\vec{S}.
 \end{aligned}$$

Es ist anzumerken, dass zum Erhalt des zu lösenden linearen Gleichungssystems noch $\vec{u}(\tau)$ diskretisiert werden muss, vgl. Abschnitt 2.3. Gleichung (2.18) gibt ein System von N gewöhnlichen Differentialgleichungen an, die mit Hilfe von numerischen Standardmethoden (Verfahren von Euler oder Crank-Nicolson) gelöst werden können. Die konkrete Anwendung dieser Verfahren wird ebenfalls in Abschnitt 2.3 besprochen.

Da der Löser der multi-Asset Black-Scholes Gleichung in der SG⁺⁺ Toolbox⁴ implementiert worden ist, soll bereits an dieser frühen Stelle ein Bezug zur Implementierung hergestellt werden. So bilden folgende Klassen die Anwendung der Matrizen A , E und F in der Software ab: Matrix A wird durch die Klassen *OperationLTwoDotProduct**, Matrix E durch die Klassen *OperationGamma** und Matrix F durch die Klassen *OperationDelta** implementiert. Die * symbolisieren an dieser Stelle die verschiedenen Gittertypen, für welche die Operatoren implementiert worden sind. Nach welcher Algorithmik diese Operatoren implementiert werden, ist hier noch nicht relevant und wird in Abschnitt 2.4.2 behandelt.

2.2.2 Log-transformierte Koordinaten

Im letzten Abschnitt wurde die Finite Elemente Methode direkt auf die multi-Asset Black-Scholes PDE angewendet. Allerdings wurde bereits eingeführt, dass die Formulierung mit logarithmischen Koordinaten bessere numerische Eigenschaften

⁴http://www5.in.tum.de/wiki/index.php/Software_Developments#SG.2B.2B

aufzeigt und daher auch in dieser Arbeit, neben der direkten PDE, im Fokus steht. Da Gleichung (2.10) keine variablen Koeffizienten besitzt, ist die schwache Form leicht unterschiedlich von Gleichung (2.17). Allerdings ist die Herleitung der äquivalenten Gleichung mit log-transformierten Koordinaten sehr ähnlich und auch einfacher, da keine variablen Koeffizienten in den Integralen zu berücksichtigen sind.

So ergibt sich für die zu Gleichung (2.17) vergleichbare schwache Formulierung der multi-Asset Black-Scholes Gleichung mit log-transformierten Koordinaten:

$$\begin{aligned}
 & \sum_{n=1}^N \frac{\partial u_n(\tau)}{\partial \tau} \int_{\Omega} \phi_n(\vec{S}) \phi_m(\vec{S}) d\vec{S} = \\
 & \sum_{n=1}^N u_n(\tau) \left[\sum_{i=1}^d \underbrace{\left(\mu_i - \frac{1}{2} \sigma_i^2 \right)}_{\nu_i^{log}} \int_{\Omega} \frac{\partial \phi_n(\vec{S})}{\partial S_i} \phi_m(\vec{S}) d\vec{S} \right. \\
 & - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \int_{\Omega} \frac{\partial \phi_n(\vec{S})}{\partial S_i} \frac{\partial \phi_m(\vec{S})}{\partial S_j} d\vec{S} \\
 & \left. - r \int_{\Omega} \phi_n(\vec{S}) \cdot \phi_m(\vec{S}) d\vec{S} \right] \quad \forall m = (1, \dots, N). \quad (2.19)
 \end{aligned}$$

Die Gleichung unterscheidet sich in den zu berechnenden L_2 Skalarprodukten von der Version ohne Koordinaten Transformation. Da bei log-transformierten Koordinaten die Ableitungen keine variablen Koeffizienten besitzen, sind die Umformungen weniger komplex und es entstehen bei der partiellen Integration keine Terme mit nur einer Ableitung in den Ansatzfunktionen. Daher ändern sich die konstanten Koeffizienten ν_i^{log} in der Herleitung der schwachen Form nicht.

Für die Matrixschreibweise folgt also:

$$A \dot{\vec{u}}(\tau) = \underbrace{\left[\sum_{i=1}^d \nu_i^{log} \cdot F_{log} - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \cdot E_{log} - r \cdot A \right]}_{:=L_{log}} \cdot \vec{u}(\tau), \quad (2.20)$$

mit den Matrizen:

$$\begin{aligned}
 E_{log} & := (e_{n,m}^{log}) \in \mathbb{R}^{N \times N} \quad \text{mit} \quad e_{n,m}^{log} := \int_{\Omega} \frac{\partial \phi_n(\vec{S})}{\partial S_i} \frac{\partial \phi_m(\vec{S})}{\partial S_j} d\vec{S}, \\
 F_{log} & := (f_{n,m}^{log}) \in \mathbb{R}^{N \times N} \quad \text{mit} \quad f_{n,m}^{log} := \int_{\Omega} \frac{\partial \phi_n(\vec{S})}{\partial S_i} \phi_m(\vec{S}) d\vec{S}.
 \end{aligned}$$

Wie im nicht transformierten Fall gibt Gleichung (2.20) ein System von gewöhnlichen Differentialgleichungen an. Auch bei der Implementierung bleibt die Ähnlichkeit zur nicht transformierten Variante erhalten. So wird der Operator der

Matrix E_{log} von der Klasse *OperationGammaLog** und der Operator von Matrix F_{log} durch die Klasse *OperationDeltaLog** implementiert. Für Matrix A existiert keine spezielle Operation (z.B. A_{log}), da dieses Skalarprodukt für beide Fälle identisch ist und sich lediglich in den verwendeten Koordinaten unterscheidet. Es wird also bereits bei der aktuell sehr abstrakten Betrachtung des Lösers klar, dass sich die Implementierung der Lösung dieser multi-Asset Black-Scholes Gleichung nur in ein paar wenigen Details von der nicht transformierten Version unterscheidet.

2.3 Diskretisierung der Zeit

Für transformierte und nicht transformierte Koordinaten wurden im letzten Abschnitt jeweils Systeme von gewöhnlichen Differentialgleichungen (ODEs) der Form

$$A\dot{\vec{u}}(\tau) = L\vec{u} \quad (2.21)$$

hergeleitet. Im Folgenden soll besprochen werden, wie diese Systeme gelöst werden können. Für diesen Schritt ist die konkrete Wahl des (räumlichen) Differentialoperators L nicht wichtig. Daher wird allgemein von L gesprochen.

Explizites Euler-Verfahren

Das explizite Euler-Verfahren ist wohl das einfachste Verfahren, um eine gewöhnliche Differentialgleichung zu lösen. Die Idee ist, den gegebenen Differentialoperator als Projektion für den Verlauf der Funktion zu wählen. Dieser wird ausgehend von einem bekannten Startpunkt, dem Anfangswert, immer wieder ausgewertet, d.h. die Zeit wird in viele sehr kleine, meist gleich große, Zeitschritte $\tau_{t+1} - \tau_t = \delta\tau$ zerlegt. Für die Länge dieser Zeitintervalle wird die Ableitung durch den Differenzenquotienten angenähert und somit der Funktionswert zum Zeitpunkt τ_{t+1} bestimmt.

Eingesetzt in Gleichung (2.21) ergibt sich somit:

$$A \frac{\vec{u}(\tau_{t+1}) - \vec{u}(\tau_t)}{\delta\tau} = L\vec{u}(\tau_t).$$

Diese Matrixgleichung kann nun in ein lineares Gleichungssystem umgeformt werden:

$$\underbrace{A\vec{u}(\tau_{t+1})}_{\vec{x}} = \underbrace{(A + \delta\tau L)\vec{u}(\tau_t)}_{\vec{b}}. \quad (2.22)$$

Der größte Vorteil des expliziten Euler-Verfahrens liegt in der Einfachheit des Systems (es besteht nur aus Matrix A). Die Berechnungskomplexität von diesem System ist wichtig, da die Gleichung mit Hilfe von iterativen Verfahren gelöst

wird. Ein komplexes System würde daher zu einer sehr langen Rechenzeit führen. Allerdings wird im weiteren Verlauf dieser Arbeit nicht tiefer auf das explizite Euler-Verfahren eingegangen, da die benötigten Zeitschrittgrößen so klein sind, dass der Vorteil des einfachen Systems nicht zum Tragen kommt. Daher sollen das implizite Euler-Verfahren und ein Verfahren der zweiten Ordnung, die Methode von Crank und Nicolson genauer betrachtet werden. Diese ist das meist verwendete Verfahren zum Lösen von PDEs in der Finanzmathematik.

Implizites Euler-Verfahren

Die Idee beim impliziten Euler-Verfahren ist nicht die Projektion der Ableitung von einem bekannten Punkt aus durch die Anwendung des Differenzenquotienten. Stattdessen wird bereits im Differentialoperator der Koeffizientenvektor zum Zeitpunkt τ_{t+1} verwendet. Damit erfolgt der Schritt nicht mehr losgelöst von der Raumdiskretisierung durch die Finiten Elemente und größere Zeitschrittweiten werden möglich. In Zeichen ergibt sich:

$$A \frac{\vec{u}(\tau_{t+1}) - \vec{u}(\tau_t)}{\delta\tau} = L\vec{u}(\tau_{t+1}),$$

und für das zu lösende System folgt

$$(A - \delta\tau L) \underbrace{\vec{u}(\tau_{t+1})}_{\vec{x}} = \underbrace{A\vec{u}(\tau_t)}_{\vec{b}}. \quad (2.23)$$

Beim einem Vergleich mit dem expliziten Euler-Verfahren fällt nun das deutlich komplexere System auf, welches in jedem Schritt des iterativen Lösers (der für jeden Zeitschritt ausgeführt wird) angewendet werden muss. Belohnt wird dies allerdings durch deutlich weniger Zeitschritte, die notwendig sind, um die Gleichung zu lösen.

Crank-Nicolson Methode

Zuletzt soll die Crank-Nicolson Methode betrachtet werden. Diese ist ein Verfahren zweiter Ordnung und kombiniert Ansätze des expliziten und impliziten Eulers: Es wird nicht der gesamte Schritt $\delta\tau$ explizit *oder* implizit durchgeführt, sondern sowohl als auch. Bei der Methode von Crank und Nicolson erfolgt ein halber Schritt explizit und ein halber Schritt implizit. Eingesetzt folgt also

$$A \frac{\vec{u}(\tau_{t+1}) - \vec{u}(\tau_t)}{\delta\tau} = \frac{1}{2} [L\vec{u}(\tau_t) + L\vec{u}(\tau_{t+1})],$$

und für das endgültige Gleichungssystem gilt

$$\left(A - \frac{1}{2}\delta\tau L \right) \underbrace{\vec{u}(\tau_{t+1})}_{\vec{x}} = \underbrace{\left(A + \frac{1}{2}\delta\tau L \right) \vec{u}(\tau_t)}_{\vec{b}}. \quad (2.24)$$

Vom Aufwand her unterscheidet sich die Systemmatrix bei der Crank-Nicolson Methode nicht von der im impliziten Euler-Verfahren. Jedoch ist die Crank-Nicolson Methode ein klein wenig schneller und genauer als das implizite Euler-Verfahren, was auf die Mittelung beider Euler-Verfahren zurückzuführen ist und sie daher ein Verfahren zweiter Ordnung ist. Normalerweise überschätzen explizite Verfahren den Funktionswert zum Zeitpunkt τ_{t+1} und implizite Verfahren unterschätzen diesen. Allerdings ist beim Start der Lösung einer ODE Vorsicht geboten. Hier existiert aufgrund des expliziten Anteils beim Crank-Nicolson Verfahren eine Fehlerquelle (falls die Schrittweite zu groß gewählt worden ist und da ein Knick in der Startlösung liegt, vgl. explizites Euler-Verfahren). Daher sollte man mit ein paar voll impliziten Schritten starten und kann danach problemlos für die restlichen Schritte auf das Crank-Nicolson Verfahren umschwenken.

Nachdem nun alle Puzzleteile für die Implementierung des Löser eingeführt worden sind, kann mit Algorithmus 2.1 ein abstraktes Vorgehen für die Lösung der Black-Scholes Gleichung angegeben werden. Dabei wird, wie angesprochen, die Laufzeit der Option in mehrere Zeitschritte der Größe $\delta\tau$ zerlegt. Für jeden Zeitschritt muss eines der gerade eingeführten linearen Gleichungssysteme gelöst werden, die aus der Zeitdiskretisierung resultieren. Für diesen Vorgang wurde im dargestellten Pseudo-Code die Routine *solveBlackScholes* als abstrakte Beschreibung gewählt. *solveBlackScholes* ruft einen iterativen Löser für das gewählte lineare Gleichungssystem auf.

Algorithmus 2.1 Abstrakter Algorithmus zum Lösen der Black-Scholes Gleichung. *solveBlackScholes* löst die PDE für einen Zeitschritt der Größe $\delta\tau$.

```
1:  $G := \text{Gitter}$ 
2:  $p(\vec{S}) := \text{Payoff-Funktion}$ 
3:  $\text{initGrid}(G, p(\vec{S}))$ 
4: for  $\tau = 0$  to  $T$  do
5:    $\text{solveBlackScholes}(G, \vec{u}, \delta\tau)$ 
6:    $\tau \leftarrow \tau + \delta\tau$ 
7: end for
8: return  $\vec{u}, G$ 
```

Zum Abschluss seien noch ein paar Worte zum verwendeten iterativen Löser angemerkt. Wegen der Matrizen F und F_{log} sind die Matrizen L und L_{log} nicht symmetrisch. Daher kann kein Standard CG Verfahren eingesetzt werden. Es muss der so genannte BiCGStab Löser verwendet werden. Leider ist dieser doppelt so aufwendig wie ein normales CG Verfahren, da für eine Iteration zwei Auswertungen des Matrix-Vektor-Produktes $L\vec{u}$ benötigt werden.

Im Quellcode müssen in einer Ableitung der abstrakten Klasse *OperationODESolverSystem* die Operatoren A und L mit Hilfe der bereits besprochen Operationen für die Matrix A, E und F implementiert werden. Für den multi-Asset Black-Scholes Löser wurde dies in der Klasse

BlackScholesODESolverSystem realisiert. Da nur noch die Operatoren A und L ausprogrammiert werden müssen, kann der in der Masterarbeit entstandene Code auch einfach auf andere ähnliche parabolische Problemstellungen, wie die Wärmeleitungsgleichung, erweitert werden.

2.4 Dünne Gitter - Ansatzraum des Löser

In den letzten Abschnitten wurde die Lösung der multi-Asset Black-Scholes Gleichung ohne eine konkrete Wahl der Ansatzfunktionen vorgestellt. Dies soll nun vollzogen werden. Wie bereits in der Einführung erläutert, werden in dieser Arbeit adaptive Dünne Gitter gewählt, da sie ein geeignetes Mittel sind, um die Herausforderung des Lösen der multi-Asset Black-Scholes Gleichung zu meistern. So erlauben sie zum Einen eine Verfeinerung des Gitters im Bereich des Knicks (vgl. Abbildung 2.1) und tragen zum Anderen dazu bei, den Fluch der Dimensionalität abzumildern.

Im ersten Teil dieses Kapitels werden die adaptiven Dünne Gitter kurz eingeführt. Dies beinhaltet sowohl die Konstruktion des Raumes als auch die benötigten Algorithmen für das Lösen einer PDE. Im zweiten Teil wird explizit auf die im Falle der Black-Scholes Gleichung benötigten Operationen eingegangen.

2.4.1 Gitterkonstruktion

Zusammengefasst lässt sich zu Dünne Gittern sagen: Geschickt gewählte ein-dimensionale Basisfunktionen werden durch einen Tensorprodukt-Ansatz so kombiniert, dass Abschätzungen zur Wichtigkeit der Beiträge der einzelnen d -dimensionalen Basisfunktionen möglich werden. Dadurch können viele Gitterpunkte im d -dimensionalen Raum eingespart werden und es entstehen ausgedünnte Gitter. Die Kernüberlegungen, die zu dieser Raumkonstruktion führen, sollen nun vorgestellt werden. Eine detaillierte Einführung in die Technik der Dünne Gitter kann hier [10] entnommen werden.

Einführung einer ein-dimensionalen Basis

Die Kurzeinführung wird mit einer ein-dimensionalen Betrachtung begonnen. Nach Gleichung (2.12) wird für die Methode der Finiten Elemente die Funktion in einer Basisrepresentation benötigt. Im ein-dimensionalen Raum ergibt sich vereinfacht:

$$f(x) \approx u(x) := \sum_{i=1}^N u_i \phi_i(x) \quad x \in \mathbb{R}.$$

Jetzt soll die naheliegenste Wahl, die so genannte *Knotenbasis* eingeführt werden. Für den weiteren Verlauf dieses Kapitels müssen nun einige Definitionen getroffen werden: Als V sei ein Funktionenraum über dem \mathbb{R}^d definiert. Es gilt weiter $V_N \subset V$ mit $N := 2^l, l \in \mathbb{N}$. l ist die so genannte Leveltiefe, die bei der Definition der Basisfunktionen benötigt wird. Somit ist V_N der Ansatzraum der stückweisen linearen Funktionen mit der Maschenweite $\frac{1}{N} = 2^{-l}$. Der Einfachheit halber werden die folgenden Betrachtungen auf dem Gebiet $[0, 1]^d$ durchgeführt. Zusätzlich werden Gitter mit Dirichlet-0-Rändern betrachtet.

Die Knotenbasis zu einem Level l spannt, basierend auf Streckung und Translation der Hutfunktion $\phi(x)$, den V_N auf:

$$\phi(x) := \begin{cases} 1 - |x| & \text{falls } x \in [-1, 1] \\ 0 & \text{sonst.} \end{cases} \quad (2.25)$$

$$\phi_{l,i}(x) = \phi\left(\frac{x - i \cdot 2^{-l}}{2^{-l}}\right) = \phi(x \cdot 2^l - i) \quad i = (1, \dots, 2^l - 1). \quad (2.26)$$

Einen anderen Ansatz stellt die *hierarchische Basis* dar. In Abbildung 2.2 wird die Knotenbasis mit der hierarchischen Basis verglichen. Es ist klar zu sehen, dass jetzt nicht mehr nur Basisfunktionen mit einer Maschenweite verwendet werden, sondern Basisfunktionen auf allen Leveln l existieren. Auf jedem Level wird eine Knotenbasis verwendet, bei der Basisfunktionen $\phi_{l,i}$ mit geraden Index i weggelassen werden. Daraus resultiert, dass Basisfunktionen auf Level l'' genau da den Funktionswert 1 ausweisen, wo alle Basisfunktionen mit $l > l''$ den Wert 0 haben. Mathematisch kann die Menge aller Hutfunktionen der hierarchischen Basis folgendermaßen definiert werden:

$$\Phi_l = \left\{ \phi_{l',i} \text{ mit } l' \leq l, i \leq 2^{l'} - 1 \text{ und } i \text{ ungerade} \right\}. \quad (2.27)$$

Im ein-dimensionalen Raum definieren die Knotenbasis auf Level l und die hierarchische Basis Φ_l den gleichen Funktionenraum bei identischer Interpolationsgüte von $\mathcal{O}\left(\left(\frac{1}{2^l}\right)^2\right)$, gemessen in der L_2 -Norm.

Zusätzlich zu den reinen Ansatzfunktionen zeigt Abbildung 2.2 auch die aufgespannten Teilräume $W_l^{(1)} := \text{span}(\phi_{l,i})$ der Basisfunktionen auf einem Level l der hierarchischen Basis. Ebenso wird die Interpolation mit Hilfe der Knotenbasis und der hierarchischen Basis betrachtet. Hierbei fällt eine Tatsache auf: Die u_i sind in beiden Fällen stark unterschiedlich. So sind die Koeffizienten u_i im Fall der Knotenbasis die Funktionswerte an den Stellen der Basisfunktionen (auch Stützstellen genannt). Bei der hierarchischen Basis ergibt sich der interpolierte Funktionswert allerdings durch Summation über die einzelnen Level. Tendenziell wird dieser Betrag mit steigendem Level immer kleiner. Daher werden die Koeffizienten im Falle der hierarchischen Basis meist auch als *hierarchische Überschüsse* bezeichnet. Diese Überschüsse sind ein Maß für die zweite Ableitung; somit erlaubt die hierarchische

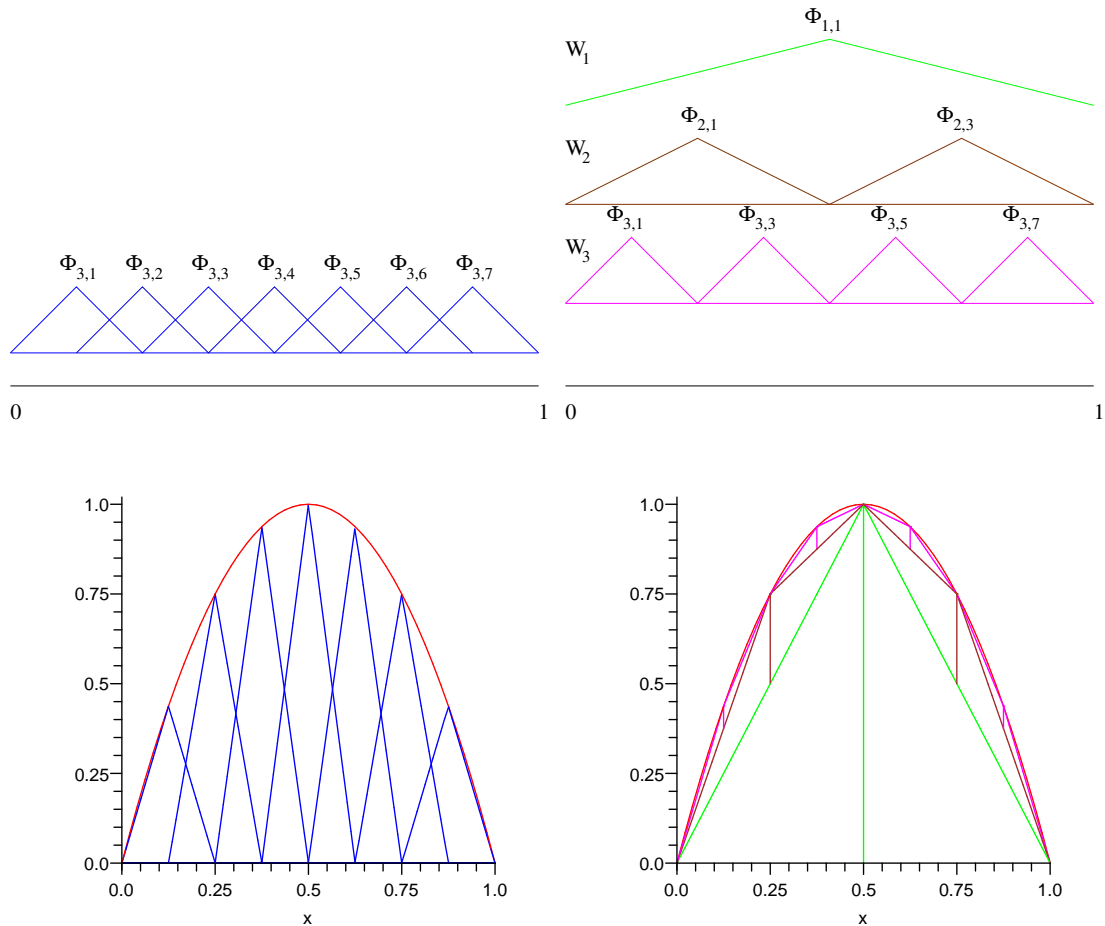


Abbildung 2.2: Links die Knotenbasis, rechts die hierarchische Basis. Bei der Knotenbasis ist gut zu erkennen, dass alle Koeffizienten bei der Interpolation in der gleichen Größenordnung liegen. Bei der hierarchischen Basis hingegen werden die Koeffizienten (auch hierarchische Überschüsse genannt) mit steigendem Level immer kleiner. Die Beträge der Überschüsse sind gekennzeichnet durch die vertikalen Linien.

Basis einen adaptiven Interpolationsansatz. Bei hohen Überschüssen können weitere Basisfunktionen auf höheren Leveln zur Verbesserung der Genauigkeit hinzugenommen werden. Auch in der folgenden mehr-dimensionalen Betrachtung spielen die hierarchischen Überschüsse eine wichtige Rolle.

Einschub: Beliebige Träger

Die hierarchische Basis wurde im letzten Abschnitt auf dem Träger $[0, 1]$ eingeführt. Allerdings werden für die Lösung der multi-Asset Black-Scholes Gleichung wegen der gemischten Skalarprodukte allgemeine Träger benötigt, da hier das Gebiet nicht

auf den d -dimensionalen Einheitswürfel $[0, 1]^d$ vor dem Lösen normiert werden kann. Nach [16] können die Ansatzfunktionen aus Gleichungen (2.25) und (2.26) mit Streckung und Translation auf einen allgemeinen Träger abgebildet werden. q sei hierbei als Streckungsfaktor definiert, welcher die Größe des neuen Trägers angibt und t sei eine Verschiebung in positive x -Richtung; $q, t \in \mathbb{R}$. Für die Hut- und Ansatzfunktionen folgt also (BB steht hier für *Bounding Box*):

$$\phi^{(BB)}(x) = \begin{cases} 1 - \frac{|x|}{q} & \text{falls } x \in [-q, q] \\ 0 & \text{sonst.} \end{cases} \quad (2.28)$$

$$\phi_{l,i}^{(BB)}(x) = \phi^{(BB)}(2^l(x - t) - qi). \quad (2.29)$$

Jedoch wird für die weitere Herleitung der Dünnen Gitter auf die Gleichungen (2.25) und (2.26) zurückgegriffen, da dadurch die benötigten Umformungen und Herleitungen besser nachzuvollziehen sind, sie sich aber von der Verwendung mit einer allgemeinen Bounding Box nicht grundlegend unterscheiden.

Mehr-dimensionale Gitter

Der Ansatzraum der mehr-dimensionalen Dünnen Gitter entsteht durch ein Tensorprodukt der ein-dimensionalen hierarchischen Basisfunktionen. Wie angekündigt erfolgt die Betrachtung im d -dimensionalen Einheitswürfel $[0, 1]^d$ und die Indizes l und i werden ebenfalls zu d -dimensionalen Indexvektoren. Durch ein Tensorprodukt ergeben sich folgende stückweise d -lineare Basisfunktionen:

$$\phi_{\vec{l}, \vec{i}}(\vec{x}) := \prod_{m=1}^d \phi_{l_m, i_m}(x_m), \quad (2.30)$$

wobei sich deren Träger aus dem Produkt der ein-dimensionalen Träger ergibt. Ebenfalls analog zum ein-dimensionalen Fall können Teilräume $W_{\vec{l}}^{(1)}$ angegeben werden, die von den d -dimensionalen Basisfunktionen aufgespannt werden:

$$\Phi_{W_{\vec{l}}^{(1)}} := \left\{ \phi_{\vec{l}, \vec{i}}(\vec{x}) : 1 \leq i_m \leq 2^{l_m} - 1, i_m \text{ unger.}, m = 1, \dots, d \right\}. \quad (2.31)$$

Mit Hilfe dieser Definitionen kann nun der Dünngitterraum angegeben werden. Für eine ausführliche Herleitung sei an dieser Stelle auf [10] verwiesen. Wegen der Tensorproduktkonstruktion der mehr-dimensionalen Basisfunktionen kann der Raum $V_n^\infty := V_n$ als die direkte Summe der Teilräume $W_{\vec{l}}^{(1)}$ geschrieben werden:

$$V_n^\infty = \bigoplus_{|\vec{l}|_\infty \leq n} W_{\vec{l}}^{(1)}. \quad (2.32)$$

Damit folgt für die Interpolation einer beliebigen Funktion $f_n^\infty(\vec{x}) \in V_n^\infty$:

$$f_n^\infty(\vec{x}) = \sum_{|\vec{l}|_\infty \leq n} f_{\vec{l}}(\vec{x}), \text{ mit } f_{\vec{l}} \in W_{\vec{l}}^{(1)}, f_{\vec{l}} = \sum_{\vec{i}: \phi_{\vec{l}, \vec{i}} \in \Phi_{W_{\vec{l}}^{(1)}}} \alpha_{\vec{l}, \vec{i}} \cdot \phi_{\vec{l}, \vec{i}}(\vec{x}). \quad (2.33)$$

Nach [9, 10] kann allgemein gezeigt werden, dass für den Beitrag von $f_{\vec{l}}$ zur Funktion $f \in V_n^\infty$ in der L_2 -Norm gemessen gilt:

$$\|f_{\vec{l}}\|_{L_2} \leq 3^{-d} \cdot 2^{-2 \cdot |\vec{l}|_1} \cdot |f|_{\mathbf{2},2}, \quad (2.34)$$

wobei $|f|_{\mathbf{2},2}$ eine Halbnorm beschreibt, die ein Maß für die gemischten zweiten Ableitungen von f ist. Eine genaue Definition dieser Halbnorm ist ebenfalls [10] zu entnehmen. Wichtiger ist an dieser Stelle die 1-Norm $|\vec{l}|_1$ im Exponenten. So wird klar, je höher die Level der Ansatzfunktionen in den einzelnen Dimensionen sind, desto kleiner wird der Beitrag dieser Ansatzfunktion zur Interpolation der eigentlichen Funktion f_n , vorausgesetzt die Funktion hat einen Verlauf ohne Knicke und Sprünge. Mit dieser Erkenntnis kann eine Kosten-Nutzen Rechnung durchgeführt werden, um die Wichtigkeit der einzelnen Teilräume zu bewerten. Diese Analyse, welche hier nicht durchgeführt wird, zeigt, dass es sinnvoll ist, nur Teilräume bis zu einem bestimmten Wert von $|\vec{l}|_1$ zu berücksichtigen, was in einem diagonalen Schnitt durch das zwei-dimensionale Teilraumschema (vgl. Abbildung 2.3) resultiert. Somit ist der Dünngitterraum $V_n^{(1)}$ folgendermaßen definiert:

$$V_n^{(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}^{(1)}. \quad (2.35)$$

Der hergeleitete Dünngitterraum $V_n^{(1)}$ beinhaltet folgende Anzahl an Freiheitsgraden (= Anzahl der Gitterpunkte):

$$|V_n^{(1)}| = \bigoplus_{|\vec{l}|_1 \leq n+d-1} |W_{\vec{l}}^{(1)}| \in \mathcal{O}(2^n \cdot n^{d-1}) \quad (2.36)$$

mit einem Interpolationsfehler der Größenordnung

$$\|f - f_n^{(1)}\|_{L_2} \in \mathcal{O}(2^{-2n} \cdot n^{d-1}) \quad (2.37)$$

mit

$$f_n^{(1)}(x) = \sum_{|\vec{l}|_1 \leq n+d-1} f_{\vec{l}}(x), \text{ mit } f_{\vec{l}} \in W_{\vec{l}}^{(1)}.$$

Erweiterung um Randbasisfunktionen

Der gerade hergeleitete Ansatzraum besitzt nach Voraussetzung Dirichlet-0-Ränder. Bei der Herleitung der schwachen Formulierung der multi-Asset Black-Scholes Gleichung wurden allerdings allgemeine Dirichlet Randbedingungen verwendet. Diese sind notwendig, um die z.B. in Abbildung 2.1 dargestellten Auszahlungsfunktionen, die die Startbedingungen des Lösers sind, auf dem gewählten Gitter darstellen zu können.

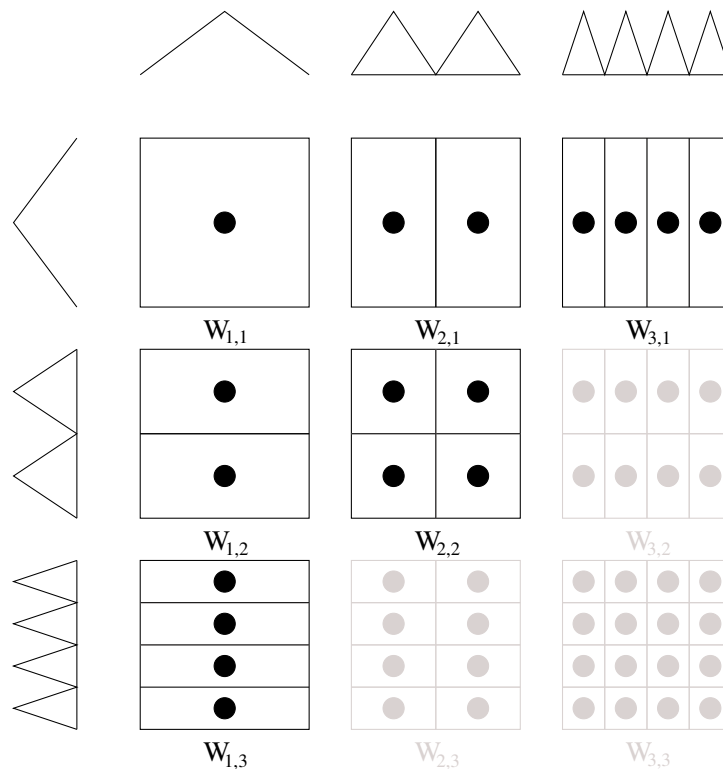


Abbildung 2.3: Teilraumschema der zwei-dimensional stückweisen d-linearen Ansatzfunktionen. Der Dünngitterraum $V_n^{(1)}$ ist hervorgehoben.

Um dies zu erreichen, werden so genannte Level-0-Basisfunktionen zur hierarchischen Basis hinzugenommen. Auf Level 0, d.h. dem Rand, sind zwei Basisfunktionen pro Dimension notwendig: Eine für den linken Randwert und eine für den rechten Randwert. Abbildung 2.4 zeigt die so entstehende hierarchische Basis. Im mehr-dimensionalen Raum gilt Folgendes: Zur jeder Basisfunktion auf Level 1 pro Dimension müssen die beiden Level-0-Basisfunktionen zusätzlich generiert werden. Dadurch entstehen so genannte Trapezränder⁵, da beide Level-0-Basisfunktionen pro Dimension zusammengenommen ein Trapez ergeben.

Auch im mehr-dimensionalen Raum spannen die Randbasisfunktionen zusätzliche Teilräume auf, die alle mindestens in einer Dimension auf dem Rand des gewählten

⁵Das Trapez existiert auf dem gesamten Definitionsbereich einer Dimension x_i des Gitters. Die Längen der beiden Grundseiten entsprechen den Koeffizienten der Level-0-Basisfunktionen in Dimension x_i .

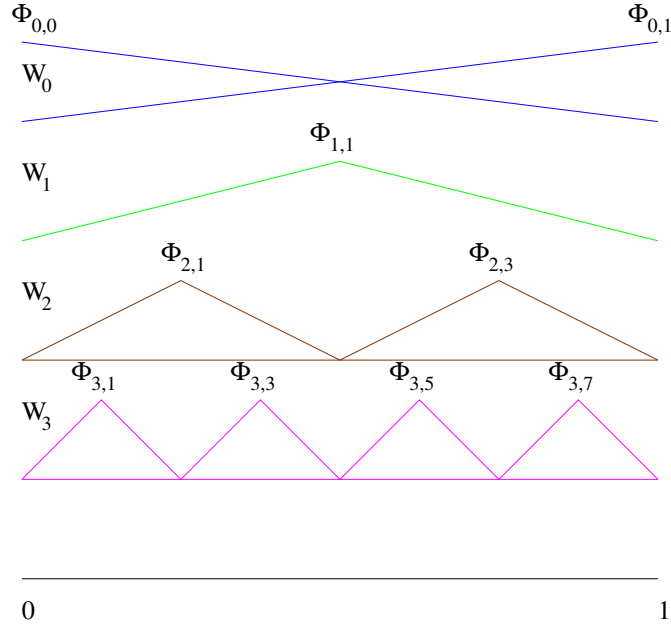


Abbildung 2.4: Ein-dimensionale hierarchische Basis mit Level-0-Basisfunktionen, um allgemeine Randwerte abzubilden. W_0 ist der Teilraum, der von den Level-0-Basisfunktionen aufgespannt wird.

Gebietes liegen. Diese zusätzlichen Teilräume sind folgendermaßen definiert (mit $l \in \mathbb{N}_0$):

$$\Phi_{W_{\vec{l}}^{(0)}} := \left\{ \phi_{\vec{l}, \vec{i}}(\vec{x}) : \begin{cases} i_m = 0, 1, & l_m = 0, \\ \begin{cases} i_j = 0, 1, & l_j = 0, \\ i_j = 1, \dots, 2^{l_j} - 1, \text{ } i_j \text{ ungerade, } & l_j \geq 1, \end{cases} & j \neq m, \\ j, m = 1, \dots, d \end{cases} \right\}. \quad (2.38)$$

Werden diese Teilräume zu dem bereits definierten Dünngitterraum mit Dirichlet-0-Rändern addiert, ergibt sich der mehr-dimensionale Dünngitterraum mit Trapezrändern $V_n^{(1), TR}$:

$$V_n^{(1), TR} := V_n^{(1)} \oplus \left(\bigoplus_{\substack{|\vec{l}|_1 \leq n+d-1 \\ |\vec{l}|_\infty \leq n}} W_{\vec{l}}^{(0)} \right) \quad (2.39)$$

Das so entstehende Teilraumschema ist in Abbildung 2.5 aufgezeichnet. Es sei an dieser Stelle angemerkt, dass der Ansatzraum $V_n^{(1), TR}$ lediglich zur Darstellung der Funktion benötigt wird. Das in Gleichungen (2.18) und (2.20) zu lösende System

besitzt nur Freiheitsgrade im Inneren des betrachteten Gebiets, d.h., die Lösung des linearen Gleichungssystems erfolgt im Ansatzraum $V_n^{(1)}$. Die Koeffizienten der Randbasisfunktionen werden lediglich zwischen zwei Zeitschritten modifiziert, um die Diskontierung der Funktionswerte zu realisieren. Anschließend können die Beiträge der Randbasisfunktionen auf den inneren Dünngitterraum vor dem Lösen neu bestimmt werden. Ein genauere Ausführung zu dieser Thematik ist Abschnitt 2.5.1 zu entnehmen.

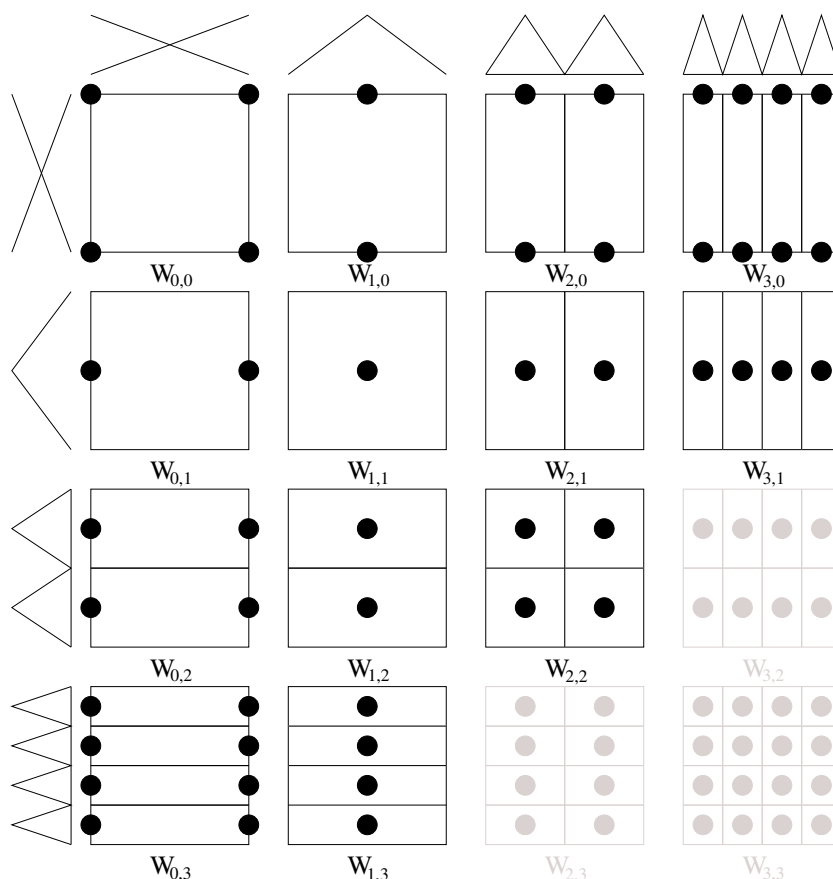


Abbildung 2.5: Der Dünngitterraum $V_n^{(1),TR}$ ist hervorgehoben.

2.4.2 Up/Down Schema für Matrizenfunktionale

Im letzten Abschnitt wurde der Ansatzraum eingeführt, in welchem die multi-Asset Black-Scholes Gleichung gelöst werden soll. D.h., die ϕ in den schwachen Formulierungen aus Gleichungen (2.17) und (2.19) werden durch die aus Gleichung (2.30) ersetzt. Doch es stellt sich die Frage, wie die somit nun konkreten linearen Gleichungssysteme aus Gleichungen (2.18) und (2.20) effizient gelöst werden können. Dazu soll angemerkt werden, dass in der vorgestellten Gitterkonstruktion die Teil-

räume nicht immer vollständig ausgeprägt sein müssen. Es kann lediglich nur eine Ansatzfunktion pro Teilraum existieren, vorausgesetzt alle hierarchischen Vorfahren in Teilräumen mit einer geringeren Levelsumme (1-Norm des Levelvektors) existieren. Dies erlaubt die Implementierung von adaptiven Dünnen Gittern. Wie diese Eigenschaft am Besten bei der multi-Asset Black-Scholes Gleichung einzusetzen ist, ist Inhalt des Kapitels 3, allerdings werden nachfolgend die Algorithmen auf den verwendeten Dünnen Gittern besprochen, welche die später vorgestellten Methoden unterstützen.

Eine Möglichkeit beim Lösen ist das Aufstellen der kompletten Matrizen A , E und F (vgl. Gleichungen (2.18) und (2.20)) und die anschließende Verwendung von Standardverfahren aus der Linearen Algebra. Dies ist jedoch aus zwei Gründen nicht ratsam. Zum Einen wird nicht die normale Knotenbasis sondern die hierarchische Basis verwendet. Wie in Abbildung 2.2 gezeigt, überlagern sich die Basisfunktionen und somit ist die aufgestellte Matrix nicht mehr dünn besetzt, so wie es bei der Knotenbasis der Fall wäre. Zum Anderen kommt hinzu, dass das Ziel die Lösung von Optionen mit mehreren Underlyings ist, also im höher-dimensionalen Raum gerechnet wird. Das erhöht, obwohl der Fluch der Dimensionalität abgeschwächt wird, die Anzahl der benötigten Gitterpunkte deutlich. Es müssten also bei diesem Vorgehen mehrere quadratische Matrizen mit Dimension von mehreren Zehntausenden aufgestellt werden. Dies scheitert heute meistens auf einer normalen Workstation, da nicht genügend Hauptspeicher zur Verfügung steht.

Die Matrizen müssen also angewendet werden, ohne sie direkt aufzustellen. Um dies effizient zu implementieren, kann die Tensorprodukt-Konstruktion des verwendeten Raumes ausgenutzt werden. Hierbei kann die Anwendung einer Matrix im d -dimensionalen Raum zurückgeführt werden auf eine mehrfache Anwendung einer Matrix im ein-dimensionalen Raum unter Zuhilfenahme einer geschickten Verknüpfung der entstehenden Teilergebnisse. Es werden alle Gitterdimensionen nacheinander abgearbeitet und zwischen den Wechslen der Dimension werden neue Eingabewerte für die Berechnungen in der nächsten Dimension auf Basis der bereits erhaltenen Werte bestimmt⁶. Für die hier benötigte Berechnung der L_2 -Skalarprodukte können die ein-dimensionalen Varianten allerdings nicht durch eine einzige Operation implementiert werden. Sie werden zerlegt in einen so genannten *Up*- und *Down*- Teil. Was dies bedeutet, lässt sich am Einfachsten anhand der Matrix A im ein-dimensionalen Fall darstellen. Für eine besser verständliche Darstellung im Folgenden sei eine geeignete, Level-weise Sortierung der hierarchi-

⁶Ein sehr einfaches Beispiel, welches nicht direkt mit der Anwendung einer FEM Matrix im Zusammenhang steht, ist das Hierarchisieren und Dehierarchisieren des Dünnes Gitters. Hierbei wird die Knotenbasis in die hierarchische Basis und andersherum umgerechnet, vgl. Abbildung 2.2. Hier werden alle Dimensionen nacheinander hierarchisiert bzw. dehierarchisiert. Die nächste Dimension bekommt als Eingabewert die schon in den bereits abgearbeiteten Dimensionen (de)hierarchisierte Koeffizienten des gesamten Gitters.

schen Basisfunktionen vorausgesetzt, also $(1, 1); (2, 1); (2, 3); (3, 1); \dots$. Damit kann die Berechnung des L_2 -Skalarprodukts folgendermaßen umgeformt werden:

$$\begin{aligned} & \sum_{n=1}^N u_n \cdot \int_x \phi_n(x) \phi_m(x) dx \\ = & \int_x \sum_{n \leq m}^N u_n \phi_n(x) \cdot \phi_m(x) dx + \int_x \sum_{n > m}^N u_n \phi_n(x) \cdot \phi_m(x) dx, \end{aligned}$$

d.h., das Skalarprodukt wird hierarchisch aufgespalten. Im linken der beiden Summanden werden die Skalarprodukte ausgewertet, bei denen Ansatzfunktionen mit gleichem oder niedrigerem Level betrachtet werden. Somit werden Werte die hierarchische Basis heruntergereicht. Im rechten Summanden ist es genau andersherum. Hier stehen Ansatzfunktionen mit höherem Level im Fokus, es werden also Werte emporgereicht. Für die Berechnung bedeutet dies, dass Matrix A nicht in einer Operation sondern in zwei Operationen angewendet wird:

$$A\vec{u} = A^{Down}\vec{u} + A^{Up}\vec{u}. \quad (2.40)$$

Würde man die Matrizen A^{Down} und A^{Up} explizit hinschreiben, so wäre A^{Down} eine linke untere Dreiecksmatrix inklusive der Diagonalen und A^{Up} eine rechte obere Dreiecksmatrix exklusive der Diagonalen. Wie die Funktionen Up und $Down$ für das Up/Down Schema tatsächlich implementiert werden, sei an dieser Stelle nicht weiter ausgeführt und auf [16, 25] verwiesen. Die Herleitung einer $Down$ für eine Matrix ist verhältnismäßig einfach und direkt, da nur Informationen die hierarchische Basis herunterfließen und somit eine einfache Rekursion genügt, den $Down$ -Teil zu berechnen. Schwieriger hingegen ist das Up , da die benötigten Funktionen auf einem höheren Level existieren und somit nicht konstant auf dem Träger der betrachteten Ansatzfunktion sind (vgl. Abbildung 2.2). Allerdings existiert ein systematisches Verfahren zur Herleitung des Up aus dem korrespondierenden $Down$, welches auch für unsymmetrische Matrizen geeignet ist, wie sie bei der Multi-Asset Black-Scholes Gleichung zu finden sind (Matrix F). Dieses Verfahren wurde unter Anderem vorgestellt in [16].

Bis jetzt wurde ausschließlich die Lösung im ein-dimensionalen Raum betrachtet. Es folgt die Verallgemeinerung auf d -dimensionale Räume. In Abbildung 2.6 wird im oberen Abschnitt nochmals der eben dargelegte Sachverhalt im ein-dimensionalen Raum visualisiert. Hier können Up und $Down$ getrennt voneinander berechnet werden, daher erfolgt eine Kopie der Koeffizienten und nach Anwendung der Operationen eine Addition der einzelnen Ergebnisse. Darunter ist die d -dimensionale Verallgemeinerung abgebildet. Auch hier erfolgt wieder eine Duplikation der Vektoren; der Algorithmus kann an diesen Stellen parallelisiert werden (vgl. Kapitel 4). Allerdings erfolgen nun die Berechnungen in den weiteren Dimensionen mit Zwischenergebnissen, wie bereits oben angemerkt. Hierbei ist wichtig, dass Informationen innerhalb des Dünnes Gitters nur entlang der hierarchischen Basis transportiert werden können. Daher müssen zuerst alle Up -Operationen ausgeführt werden, bevor mit den $Down$ -Operationen fortgefahren werden kann. Dass

dies der Fall ist, ist auf den ersten Blick in Abbildung 2.6 schwer zu erkennen, wird aber durch die rekursive Struktur sichergestellt.

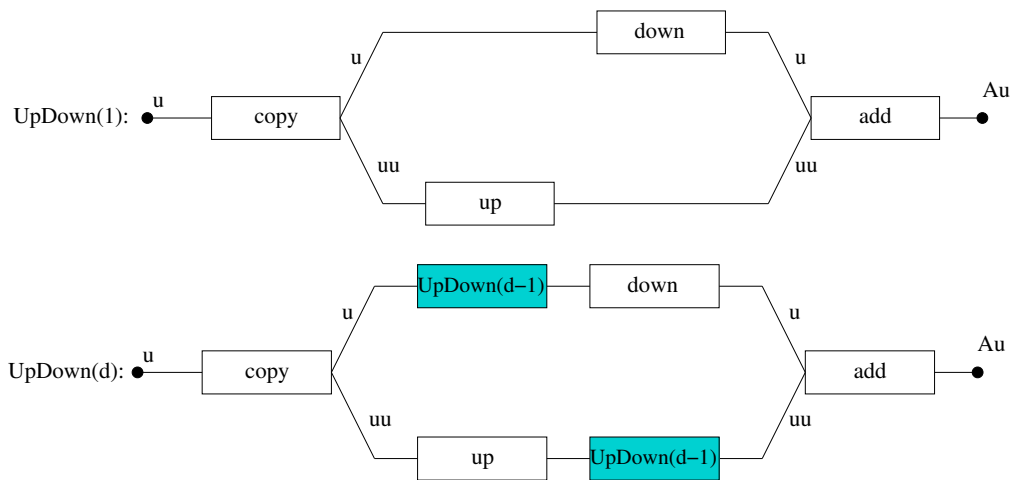


Abbildung 2.6: Schema des Up/Down Schemas: Oben der ein-dimensionale Fall, unten der d -dimensionale Fall. Eigene Darstellung nach [9].

Dies soll nochmals an einem Beispiel im zwei-dimensionalen Raum verdeutlicht werden. Folgende Operationen sind auf den Koeffizienten bzw. deren Kopie(n) auszuführen (die Rekursion wurde ausformuliert):

$$\begin{aligned} u &= up(0, up[1, u]) + down(0, up[1, u]) \\ uu &= down[1, up(0, uu) + down(0, uu)], \end{aligned}$$

es sind also insgesamt 6 Operationen auszuführen. Diese setzen sich aus 4 Operationen, die mit unterschiedlichen Koeffizienten aufgerufen werden, zusammen. In Abbildung 2.7 werden die benötigten Gittertraversierungen gezeigt. Die beiden oberen Bilder sind die Up für beide Dimensionen und die beiden unteren Bilder sind die entsprechenden $Down$ Routinen. Für alle Darstellungen gilt Folgendes: Wird ein Up oder $Down$ in einer Dimension angewendet, so sind hierfür mehrere ein-dimensionale Up oder $Down$ Operationen notwendig. Für das linke obere Bild soll dies detailliert besprochen werden. Hier wird das Up in x -Richtung angewendet. Das bedeutet, das Up muss für das gesamte Gitter in x -Richtung berechnet werden, indem die hierarchischen Basis in y -Richtung hinabgestiegen wird. D.h. es wird gestartet beim Wurzelpunkt des Gitters: $[(1, 1), (1, 1)]$ und ein Up in x ausgeführt. Anschließend wird zur Wurzel zurückgekehrt und in y -Richtung zum Gitterpunkt $[(1, 1), (2, 1)]$ abgestiegen. Jetzt erfolgt wieder eine Up -Anwendung in x -Richtung. Dies wird solange wiederholt, bis die Blätter des Gitters in y -Richtung erreicht worden sind. In allen vier Schemata sind die Up und $Down$ Routinen mit Hilfe von Pfeilen dargestellt, die Abstiege erfolgen in anderen Dimensionen und sind, um Verwirrungen gering zu halten, nicht mit Pfeilen gekennzeichnet. Allgemein lässt sich sagen, dass die Anzahl der benötigten Up und $Down$ Aufrufe der Anzahl der Kanten in einem Binärbaum der Tiefe d entspricht: $2^{d+1} - 2$.

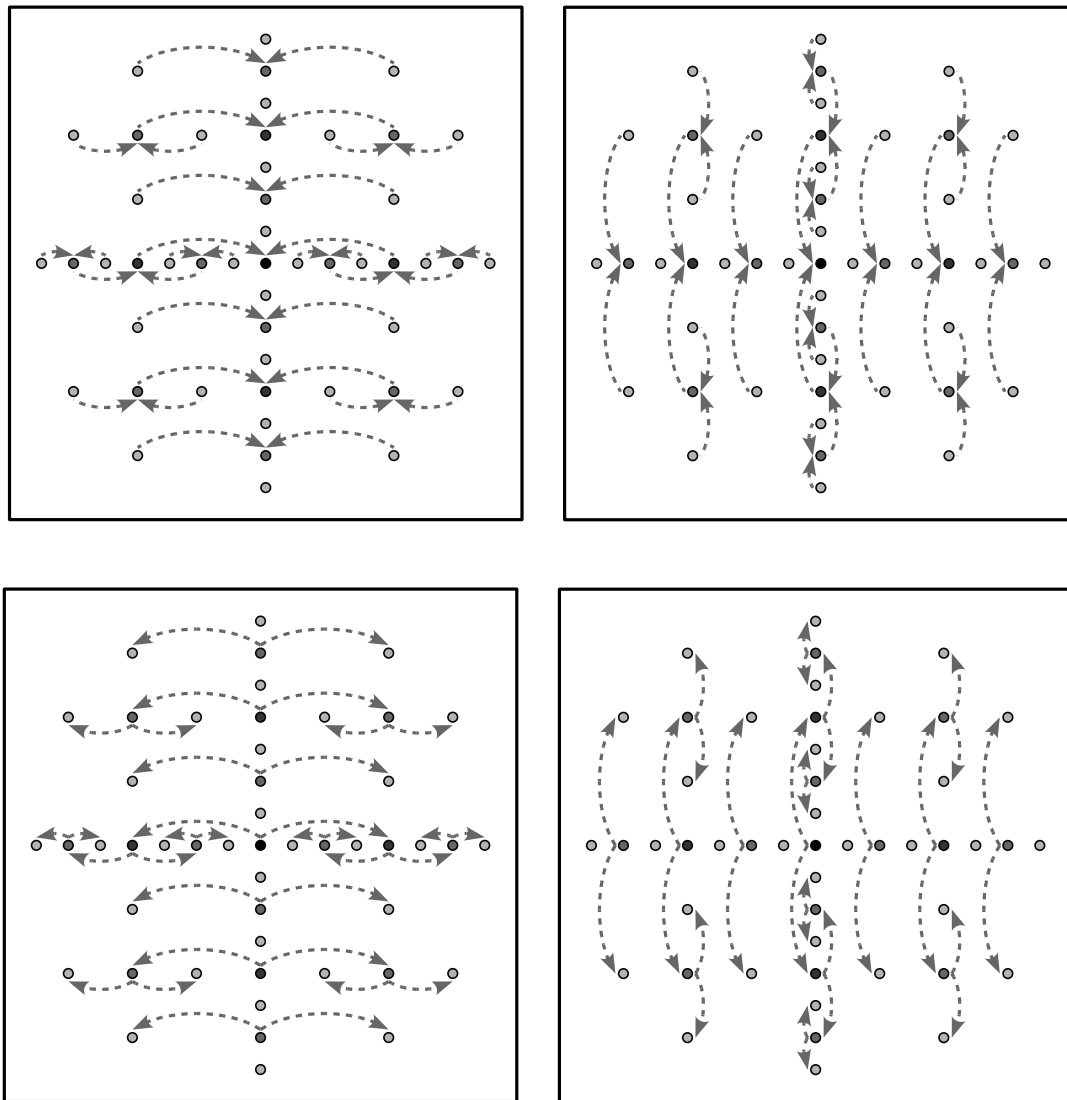


Abbildung 2.7: Anwendung des Up/Down Schemas auf ein zwei-dimensionales Dünnes Gitter. Die oberen beiden Darstellungen zeigen zuerst das Up in x -Richtung (links), es wird die hierarchische Basis in y -Richtung abgestiegen und das Up in x -Richtung berechnet. Rechts das Up nun mit umgedrehten Achsen. Die unteren beiden Abbildungen zeigen dasselbe für das $Down$. Einzeldarstellungen aus [11].

2.4.3 Aufspaltung der benötigten Operationen

Aus dem letzten Abschnitt ist deutlich geworden, dass für die Implementierung des Up/Down Schemas für die Matrizen aus Gleichungen (2.18) und (2.20) die mehr-dimensionalen Integrale so umgeschrieben werden müssen, dass jede Dimension einzeln betrachtet wird. Dadurch können anschließend die ein-dimensionalen

Anteile für die benötigten *Up* und *Down* Operationen bestimmt werden. An dieser Stelle sollen ausschließlich die entsprechenden aufgespalteten Formulierungen angegeben werden. Für eine genaue Herleitung der ein-dimensionalen Varianten sei auf [16] verwiesen, wo dies ausführlich dargestellt ist.

Für die Aufspaltung des L_2 -Skalarproduktes der Matrix A folgt:

$$\int_{\Omega} \phi_n(\vec{S}) \phi_m(\vec{S}) d\vec{S} = \prod_d \int_{S_d} \phi_{n_d}(S_d) \phi_{m_d}(S_d) dS_d.$$

Es handelt sich hierbei um eine recht einfache Umformung, da das Produkt aus der Tensorraumkonstruktion mit Regeln der Integralrechnung aus dem mehr-dimensionalen Integral herausgezogen werden kann und damit ein Produkt von d ein-dimensionalen L_2 -Skalarprodukten entsteht. Ähnliches gilt für die Aufspaltung der Matrix E ((vgl. Gleichungen (2.18) und (2.20))):

$$\int_{\Omega} S_i S_j \frac{\partial \phi_n(\vec{S})}{\partial S_i} \frac{\partial \phi_m(\vec{S})}{\partial S_j} d\vec{S} = \begin{cases} 2 \cdot \int_{S_i} S_i \frac{\partial \phi_{n_i}(S_i)}{\partial S_i} \phi_{m_i}(S_i) dS_i & \text{für } j < i \\ \cdot \int_{S_j} S_j \phi_{n_j}(S_j) \frac{\partial \phi_{m_j}(S_j)}{\partial S_j} dS_j \\ \cdot \prod_{d \neq i, j} \int_{S_d} \phi_{n_d}(S_d) \phi_{m_d}(S_d) dS_d \\ \\ \int_{S_i} S_i^2 \frac{\partial \phi_{n_i}(S_i)}{\partial S_i} \frac{\partial \phi_{m_i}(S_i)}{\partial S_i} dS_i & \text{für } i = j \\ \cdot \prod_{d \neq i} \int_{S_d} \phi_{n_d}(S_d) \phi_{m_d}(S_d) dS_d \\ \\ 0 & \text{für } j > i \end{cases}.$$

Allerdings wird hier die Symmetrie des Operators ausgenutzt: Der Dreiecksanteil für $j < i$ kann mal 2 genommen werden und dafür kann der Dreiecksteil für $i < j$ vernachlässigt werden. Die Diagonale, also $i = j$, muss gesondert mit einer speziellen ein-dimensionalen Operation behandelt werden. Für die Matrix F ist die Aufspaltung ein wenig einfacher als für Matrix E , da keine Fallunterscheidung benötigt wird:

$$\int_{\Omega} S_i \frac{\partial \phi_n(\vec{S})}{\partial S_i} \phi_m(\vec{S}) d\vec{S} = \int_{S_i} S_i \frac{\partial \phi_{n_i}(S_i)}{\partial S_i} \phi_{m_i}(S_i) dS_i \\ \cdot \prod_{d \neq i} \int_{S_d} \phi_{n_d}(S_d) \phi_{m_d}(S_d) dS_d$$

Bei einer genaueren Betrachtung fällt auf, dass bei der Implementierung von Matrix F keine weiteren ein-dimensionalen Operationen hinzukommen, und alles mit den bereits hergeleiteten Operationen realisiert werden kann. Zusammengefasst lässt sich sagen, dass folgende (ein-dimensionale) L_2 -Skalarprodukte mittels *Up* und *Down* implementiert werden müssen, um die multi-Asset Black-Scholes Gleichung auf adaptiven Dünnen Gittern zu lösen:

$$\int_S \phi_n(S) \phi_m(S) dS \quad (2.41)$$

$$\int_S S \frac{\partial \phi_n(S)}{\partial S} \phi_m(S) dS \quad (2.42)$$

$$\int_S S \phi_n(S) \frac{\partial \phi_m(S)}{\partial S} dS \quad (2.43)$$

$$\int_S S^2 \frac{\partial \phi_n(S)}{\partial S} \frac{\partial \phi_m(S)}{\partial S} dS \quad (2.44)$$

Falls die multi-Asset Black-Scholes Gleichung mit log-transformierten Koordinaten verwendet werden soll, verändern sich Gleichungen (2.42) bis (2.44) leicht:

$$\int_S \frac{\partial \phi_n(S)}{\partial S} \phi_m(S) dS \quad (2.45)$$

$$\int_S \phi_n(S) \frac{\partial \phi_m(S)}{\partial S} dS \quad (2.46)$$

$$\int_S \frac{\partial \phi_n(S)}{\partial S} \frac{\partial \phi_m(S)}{\partial S} dS \quad (2.47)$$

Hiermit sind alle wichtigen Aspekte, die zum Lösen der multi-Asset Black-Scholes Gleichung auf Dünnen Gittern unerlässlich sind, dargestellt worden. Zum Abschluss dieses einführenden Kapitels wird nachfolgend noch auf Erweiterungen eingegangen, mit deren Hilfe verschiedene Optionstypen effizient gelöst werden können.

2.5 Strategien für spezielle Optionen

In der Einführung zur Optionspreistheorie ist bereits auf die unterschiedlichen Arten von Optionen, in Bezug auf deren Ausübungszeitpunkt und die Art der Payoff-Funktionen, eingegangen worden. Wie solche Abwandlungen das konkrete Vorgehen bei der Implementierung des Löser beeinflussen, soll anschließend kurz erläutert werden.

2.5.1 Randbehandlung bei der europäischen Option

Begonnen wird mit dem einfachsten Fall, dem der europäischen Option. Hier können die Dirichletränder bestens ausgenutzt werden. Da die Funktion wie besprochen an den Rändern existieren muss, besitzen die linearen Gleichungssysteme (2.22), (2.23) und (2.24) zwar Gitterpunkte auf Level 0, allerdings sind diese Stützstellen keine Freiheitsgrade. Für die technische Umsetzung dieser Dirichlet-Randbedingungen existieren zwei Möglichkeiten: Zum Einen können die Randbedingungen während den Ausführungen der einzelnen Up/Down Schemata in den

Up/Down Algorithmus eingearbeitet werden, zum Anderen kann auch ein echtes inneres Gitter erstellt werden, auf dem das Gleichungssystem gelöst wird. Allerdings müssen für den letzteren Fall die Koeffizienten vor dem Lösen modifiziert werden, da das Gitter, auf dem anschließend gelöst wird, keine Level-0-Basisfunktionen mehr besitzt, diese aber zur Darstellung der Payoff-Funktion in der hierarchischen Basis benötigt werden. Die Gleichungssysteme können in allen Fällen folgendermaßen geschrieben werden⁷:

$$\left(\begin{array}{c|c} I & 0 \\ \hline A_R & A_I \end{array} \right) \cdot \begin{pmatrix} x_R \\ x_I \end{pmatrix} = \begin{pmatrix} b_R \\ b_I \end{pmatrix}, \quad (2.48)$$

D.h. Matrix A , die rechte Seite \vec{b} und der Lösungsvektor \vec{x} können in Randbasisfunktionen und innere Stützstellen aufgespalten werden. Jetzt folgt durch die Voraussetzung der Dirichlet-Ränder, dass \vec{b}_R und \vec{x}_R sich während des kompletten iterativen Lösungsprozesses nicht verändern. Daraus ergibt sich die Möglichkeit, ein deutlich kleineres System in jedem Schritt des iterativen Verfahrens zu betrachten, nämlich A_I . aus Gleichung (2.48) kann hergeleitet werden:

$$A_I \vec{x}_I = \vec{b}_{I_{neu}} = \vec{b}_I - \underbrace{A_R \cdot \vec{b}_R}_{\vec{k}_I} = \vec{b}_I - \underbrace{A_R \cdot \vec{x}_R}_{\vec{k}_I}. \quad (2.49)$$

In Gleichung (2.49) sieht man sehr gut den Beitrag der nicht Dirichlet-0-Randbedingungen, den Vektor \vec{k}_I . Dieser ist die weiter oben angesprochene Modifikation der Koeffizienten der inneren Basisfunktionen. Dies ist einmal vor dem Lösen der Gleichung notwendig, also in jedem Zeitschritt der Lösers, da sich die Randwerte wegen ihrer Diskontierung in jedem Zeitschritt ändern.

Diese Arbeit wählt ein separates inneres Gitter zum Lösen des linearen Gleichungssystems in jedem Zeitschritt. Diese Wahl wurde aus Hardware-technischen Gründen getroffen. Betrachtet man z.B. ein 5-dimensionales reguläres Gitter mit 6 Levels, so entfallen ca. 100.000 Gitterpunkte auf den Rand aber nur ca. 5000 bilden das innere Gitter. Bei der ersten der beiden Möglichkeiten, der Einarbeitung in die Up/Down Schemata, werden also während des Lösens nur 5% des allozierten Speichers verwendet, im zweiten Fall wird der gesamte allozierte Gitterspeicher verwendet, der aber nur 5% der ursprünglichen Größe umfasst. Somit können Cache-Level und Prefetch-Algorithmen der CPUs effizienter genutzt werden.

Tabelle 2.2 zeigt die durch diese Implementierung erreichten Speed-Ups im Vergleich zur Einarbeitung der Dirichlet-Randbedingungen in die Up/Down-Algorithmik, so wie sie in [16] erfolgt ist. Auch erkennt man den angesprochenen Hardwareeffekt in dieser Tabelle. Der hier angegebene 5-dimensionale Fall

⁷Für die hier gewählte vereinfachte Darstellung wird folgende Sortierung der Basisfunktionen vorausgesetzt: Zuerst werden alle Basisfunktionen auf Rand (Level 0) enumeriert, anschließend die inneren Basisfunktionen. Die Sortierung innerhalb dieser beiden Gruppen spielt keine Rolle. Auch das Separieren in diese beiden Gruppen ist nicht notwendig und wird daher auch in der tatsächlichen Implementierung nicht vollzogen.

entspricht den obigen Überlegungen. Somit wäre ein Speed-Up von ca. 20X zu erwarten, der gemessene Speed-Up liegt aber deutlich darüber. Das hängt damit zusammen, da nun die deutlich kleineren inneren Gitter, die im iterativen Löser verwendet werden, in den Caches der verwendeten Intel Westmere CPU (12 MB LLC⁸) Platz finden.

Assets	Speed-Up
2	1.6X
3	4.0X
4	10.3X
5	31X

Tabelle 2.2: Durch effiziente Randbehandlung erreichte Speed-Ups im Vergleich zur Einarbeitung der Randbedingungen in die Up/Down Algorithmik. Gemessen wurden diese Speed-Ups auf einer Tylersburg-EP Workstation, bestückt mit zwei Intel Xeon X5650 CPUs, parallele Ausführung mit 12 Threads, kein SMT, Turbomodus der CPUs war aktiviert.

2.5.2 Freie Dimensionen für die asiatische Option

Das Lösen von multi-Asset Optionen mit Dünnen Gittern ist in diesem Kapitel recht allgemein vorgestellt worden, da ein Toolbox-Ansatz mit diesem Löser verfolgt wird. Er sollte also mit wenig Anpassungen in der Lage sein, die verschiedenen Spielarten der Optionen ohne größere Probleme effizient zu lösen. Der letzte Unterabschnitt hat eine sehr mächtige Optimierung für die europäische Option vorgestellt. Nachfolgend soll auf die benötigten Änderungen eingegangen werden, die das Lösen einer asiatischen Option ermöglichen. Leider sind diese deutlich komplexer als die Optimierung im Fall der europäischen Option.

Da die asiatische Option bei der Berechnung des Payoffs am Ende der Laufzeit den Durchschnitt über die Entwicklung des Kurses des Assets benötigt, muss dieser in einer zusätzlichen Dimension mitgeführt werden. D.h. um eine asiatische Option auf ein Asset zu berechnen, wird ein zwei-dimensionales Gitter benötigt, bei dem in jedem Zeitschritt der Durchschnitt aktualisiert und das Lösungsgebiet neu bestimmt werden muss. Bis auf das Lösen eines Zeitschrittes ist diese Funktionalität bereits in einer anderen Toolbox namens *fideum* [6, 27] am Lehrstuhl IV der Technischen Universität München vorhanden. Es muss also der multi-Asset Black-Scholes Solver so angepasst werden, dass er das Lösen auf einer ausgewählten Anzahl von Dimensionen des Gitters und nicht nur in allen Gitterdimensionen unterstützt.

⁸Last-Level-Cache, beim Intel Westmere ist das ein für alle Kerne gemeinsamer Level 3 Cache.

Um dies zu erreichen, wurden die Up/Down Schemata im Black-Scholes Löser um so genannte *algorithmische Dimensionen* erweitert. Hier kann angegeben werden, in welcher Reihenfolge welche Gitterdimensionen im Up/Down Algorithmus abgearbeitet werden. Die nicht betrachteten Dimensionen beeinflussen das Ergebnis nicht weiter und es werden daher Freiheitsgrade auf dem Rand ohne Randbedingungen in diesen Dimensionen benötigt. Dieses Vorgehen soll in Anlehnung an die Einführung in das mehr-dimensionale Up/Down beispielhaft dargestellt werden: Hierfür wird im Folgenden ein zwei-dimensionales Gitter betrachtet, bei dem nur in einer Dimension, der zweiten, eine ein-dimensionale Black-Scholes Gleichung gelöst werden soll. Dies ist eine starke Vereinfachung, allerdings werden alle benötigten Aspekte für die Lösung eines Zeitschrittes beleuchtet. Im realen Fall der asiatischen Optionen ergibt sich kein Dimensions-paralleler Payoff, da sich die Payoff-Funktion deutlich von diesem einfachen Beispiel unterscheidet. Hier wird durch die nötige Verdrehung der Achsen während des gesamten Lösungsvorgangs das Gitter sukzessive verzerrt.

In Abbildung 2.6 wird also in unserem Beispiel nicht mehr der rekursive Einstiegspunkt in den Algorithmus aufgerufen, sondern nur noch der Rekursionsschluss in Dimension 2. Ausgedrückt in den oben eingeführten Gleichungen ergibt sich:

$$\begin{aligned} algDim &= 2 \\ u &= up(algDim, u) \\ uu &= down(algDim, uu), \end{aligned}$$

in Dimension 1 wird keine Rechnung während des Lösens der Black-Scholes Gleichung durchgeführt! Auch werden die Auswirkungen dieser Änderungen in Abbildung 2.7 deutlich. Hier werden nur die Ausführungen des Up/Down Schemas in Dimension 2 benötigt. D.h., es sind lediglich 2 Operationen statt den 6 im Falle der zwei-Asset europäischen Option notwendig. Damit sind für die ein-Asset asiatische Option genauso viele Up/Down Operationen wie für die ein-Asset europäische Option nötig. Allerdings steigt für mehr-Asset asiatische Optionen die Rechenzeit deutlich über die der europäischen Option an. Dies liegt an den Freiheitsgraden auf dem Rand in den nicht-algorithmischen Dimensionen. Hier muss nun ebenfalls das Gitter auf den Rändern beim Lösen der Black-Scholes Gleichung abgelaufen und Dirichlet-Randbedingungen direkt eingearbeitet werden. Somit ist die oben vorgestellte Optimierung nicht mehr ohne Weiteres anwendbar.

In Abbildung 2.8 werden die Funktionen aus dem obigen Beispiel gezeigt. Hier wurde entlang von Dimension x_0 in Dimension x_1 unabhängig der Black-Scholes Löser gerechnet. Besonders gut sind in der rechten Abbildung die bereits thematisierten Freiheitsgrade auf dem Rand zu sehen, da auch hier die ein-dimensionale Black-Scholes Gleichung gelöst wird.

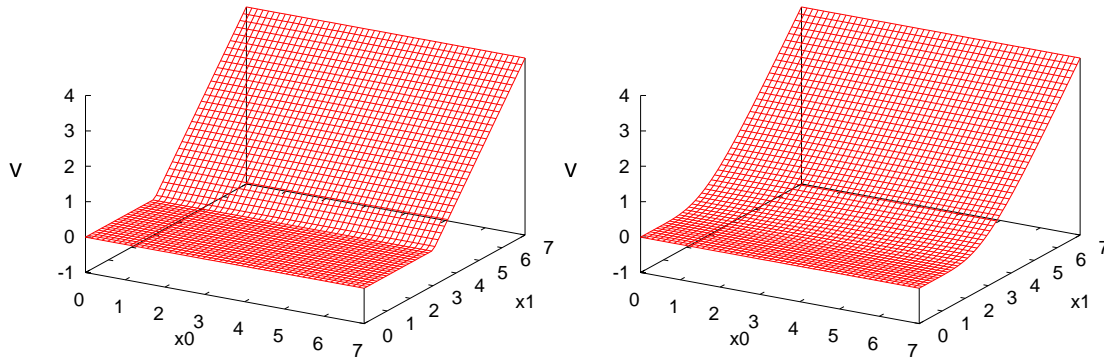


Abbildung 2.8: Verwendung der algorithmischen Dimensionen. Links die Startbedingung, rechts die Lösung. Besonderes Augenmerk gilt den Freiheitsgraden auf dem Rand in Dimension x_0 .

2.5.3 Nebenbedingungen am Beispiel der amerikanischen Option

Die amerikanische Option kann ebenfalls mittels der *fideum* Toolbox gelöst werden. Sie kann zu jeder Zeit während der Laufzeit ausgeübt werden. Dies ist jedoch nach [32] nur für Put-Optionen sinnvoll. Bei Calls gibt es keine Veranlassung für eine frühere Ausübung. Das Lösen von amerikanischen (oder auch Bermuda Optionen) ist verhältnismäßig einfach zu implementieren. Der Zeitraum der Laufzeit wird in passende Unter-Zeiträume zerlegt (z.B. Tage, Monate) und zwischen diesen Zeitpunkten wird der klassische Löser der europäischen Option aufgerufen. Jetzt können zu diesen Zeitpunkten die Werte der Option bestimmt werden und anschließend der gesuchte Preis der Option errechnet werden.

Diese Arbeit wird im folgenden Kapitel die Verwendung der Adaptivität des vorgestellten Dünngitterraums beim Lösen von europäischen Optionen analysieren. Die letzten beiden Unterabschnitte haben gezeigt, dass der hier entwickelte Löser nicht nur für die Bewertung von europäischen Optionen eingesetzt werden kann.

3 Adaptivität bei der Lösung der Black-Scholes PDE

Im letzten Abschnitt ist bereits an einigen Stellen das Verfahren der adaptiven Dünne Gitter am Rande erwähnt worden. Nun sollen diese adaptiven Gitterstrukturen für den Anwendungsfall der Black-Scholes Gleichung genauer betrachtet werden. Da ebenfalls im letzten Abschnitt schon auf Algorithmen eingegangen wurde, die adaptive Dünne Gitter unterstützen, wird dieser Abschnitt ausschließlich das Werkzeug der Adaptivität behandeln. Hierzu soll zuerst herausgearbeitet werden, wann und warum adaptive Dünngittermethoden angewendet werden sollten. Anschließend wird auf die verschiedenen Möglichkeiten der adaptiven Gittergenerierung eingegangen. Hier wird dargestellt, auf welche Art und Weise entsprechende Gitter konstruiert werden können, also an welchen Stützstellen die Gitter wie verfeinert werden. Diese Methoden werden an Hand von Optionen auf zwei Assets bewertet und daraus Heuristiken hergeleitet, welche auf Optionen mit einer höheren Anzahl von Assets angewendet werden. Abgerundet wird der vorliegende Abschnitt mit einer Analyse der Fehlerminimierung des BiCGStab Verfahrens, um die optimale Anzahl an Iterationen pro Zeitschritt zu bestimmen.

3.1 Allgemeines zur Verwendung von adaptiven Gitterstrukturen

Eine grundlegende Annahme bei der Formulierung des Dünngitterraums war die Glattheit der darzustellenden Funktionen; genauere Ausführungen zu dieser Thematik können hier gefunden werden: [10]. Einige Anwendungen, so auch das Lösen der Black-Scholes Gleichung, verletzen allerdings diese Voraussetzung, da die Start-Bedingung (Payoff-Funktion) einen Knick aufweist, dieser ist sogar $d - 1$ -dimensional, wie auch aus Abbildung 3.1 deutlich wird: Es sind so gut wie keine Gitterpunkte vorhanden, um den Knick auf dem gewählten Gitter darzustellen. Um solche Probleme auf Dünne Gittern lösen zu können, bestehen zwei Möglichkeiten. Zum Einen kann das Level l des Dünngitterraums hoch getrieben werden, um die Verletzung abzufedern. Allerdings wird bei genauerer Betrachtung ein Nachteil dieses Vorgehens deutlich: Die Glattheitsannahme wird nicht auf dem gesamten Lösungsgebiet, sondern nur am Knick der Payoff-Funktion verletzt, so werden auch

glatte Bereiche sehr fein aufgelöst, was nicht notwendig ist. Es bietet sich nach [26] eine weitere Möglichkeit an: Die der bereits thematisierten adaptiven Gitterstrukturen. Hierbei können für einzelne Gitterpunkte deren hierarchische Nachfahren bestimmt werden. Eine solche Methode soll im Folgenden als *Verfeinerung* bezeichnet werden. Als Hilfe zur Bestimmung der zu verfeinernden Punkte dienen meist die hierarchischen Überschüsse. Es hat sich gezeigt, dass sich durch Anwendung von adaptiven Dünnen Gittern deutlich bessere Ergebnisse in Bezug auf Genauigkeit und benötigte Lösungszeit im Data Mining realisieren lassen, siehe [16, 26]. Dies soll nun auf die Black-Scholes Gleichung übertragen werden, d.h. das Lösungsgitter der Black-Scholes Gleichung muss im Bereich des Knicks (stark) verfeinert werden. Abbildung 3.2 zeigt beispielhaft die Verfeinerung des Gitters um den Knick. Die Herleitung von Heuristiken für die beste Methode der Knick-Verfeinerung ist Thema der folgenden Unterabschnitte. Zu dieser Abbildung ist anzumerken, dass zusätzlich zum Knick auch andere Teile des Gitters verfeinert werden: Die achsenparallelen Gitterpunkte außerhalb des Knicks. Dies ist eine Folge der hierarchischen Basis. Wie bereits während der Einführung des Up/Down Schemas besprochen wurde, müssen für einen Gitterpunkt alle hierarchischen Vorfahren existieren, damit die formulierten Algorithmen ausgeführt werden können. Aus Abbildung 3.1 wird klar, dass auf dem Knick fast noch keine Punkte liegen und die benötigten keinesfalls auf den ersten Leveln liegen. Daher müssen zumindest für alle Gitterpunkte Stützstellen auf dem Rand und Level 1 erzeugt werden. Diese sind die gut erkennbaren Verfeinerungen auf den Hauptachsen in Abbildung 3.2.

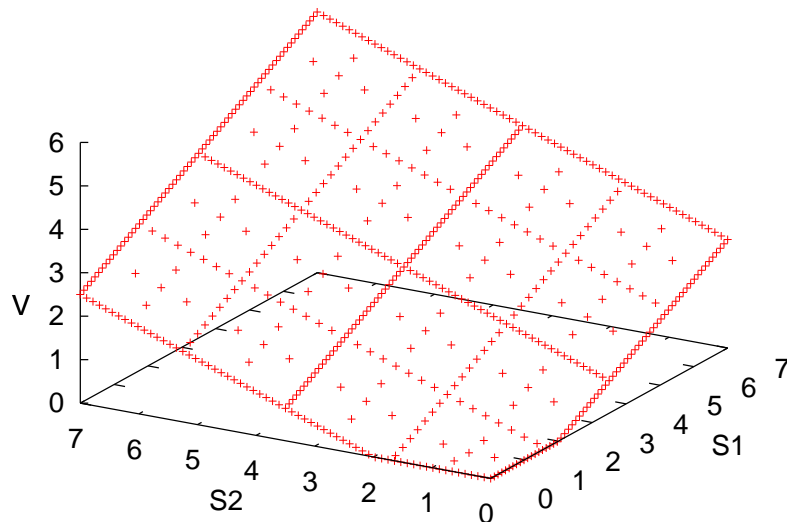


Abbildung 3.1: Payoff-Funktion eines Baskets auf 2 Assets mit $K = 1$, d.h. der Knick ist die Strecke zwischen den Punkten $S_1 = 2$ und $S_2 = 2$, dargestellt auf einem regulären Dünnen Gitter mit 6 Leveln.

In [16, 26] wurden adaptive Dünne Gitter für Data Mining Aufgaben verwendet und zu anderen Verfahren vergleichbare Ergebnisse erzielt. Allerdings können die

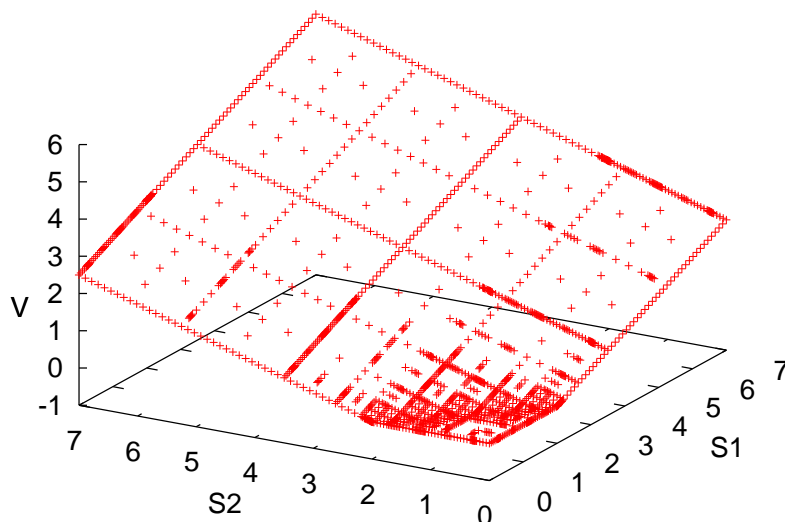


Abbildung 3.2: Basket aus Abbildung 3.1, nun aber mit einer Verfeinerung des Knicks. Dieser ist nun deutlich besser darstellbar auf dem gewählten Gitter.

dort entwickelten Vorgehen zur Verwendung von adaptiven Gittern nicht übernommen werden. Die globale Verfeinerungsstrategie ist beim Lösen der Black-Scholes Gleichung anders als im Falle das Data Minings zu wählen, da es sich hier um eine parabolische Differentialgleichung handelt, und somit eine Startlösung existiert, die bereits möglichst genau auf dem Gitter darzustellen ist. In Algorithmus 3.1 wird das Vorgehen beim Data Mining sehr abstrakt dargestellt. So wird beim Data Mining mit einem recht groben Gitter gestartet, auf welchen ein gegebener Datensatz gelernt wird. Anschließend wird die Qualität der gelernten Funktion bestimmt. Liegt nun die Genauigkeit (typische Maße sind Anzahl der richtig klassifizierten Instanzen bzw. MSE im Falle von Regressions-Fragestellungen) über der geforderten, so wird das Verfahren abgebrochen. Andernfalls erfolgt eine Verfeinerung des Gitters und ein Neustart des Lernvorgangs auf der neuen Gitterstruktur. Mit jedem Verfeinerungsschritt bekommt das Gitter also mehr Stützstellen, die abhängig von dem Verlauf der vorangegangenen Verfeinerungen sind.

Im Falle der Black-Scholes Gleichung sieht der Lösungsvorgang allerdings deutlich anders aus und sie kann zudem beliebig genau gelöst werden¹. Das bedeutet zum Einen, dass ein Lösungsvorgang deutlich länger dauert (parabolische Differentialgleichung) und zum Anderen, dass basierend auf einer Ausführung nicht direkt ein anderes Gitter mit höherer Genauigkeit hergeleitet werden kann. Algorithmus 3.2 zeigt die Anwendung von adaptiven Dünnen Gittern beim Lösen der Black-Scholes Gleichung. Es sticht sofort ins Auge, dass verschiedene Gittermodifikationsroutinen benötigt werden: *refineInitialGrid* und *restructureGrid*. Am

¹Im Data Mining macht eine zu genaue Lösung keinen Sinn, da es dann zum so genannten Overfitting kommen kann. Als Overfitting wird das zu starke Anpassen eines Klassifikators an die Trainingsdaten bezeichnet. Da Data Mining nicht Bestandteil dieser Arbeit ist, sei auf [31] verwiesen.

Algorithmus 3.1 Ein Algorithmus für Data Mining auf adaptiven Dünnen Gittern. Die Funktion *trainGrid* lernt den Datensatz D auf einem Gitter G mit den Koeffizienten u . *getAccuracy* bestimmt die Genauigkeit auf dem aktuellen Gitter, wohingegen *refineGrid* ein gegebenes Gitter Überschuss-basiert verfeinert.

```
1:  $D := \text{Datensatz}$ 
2:  $G := \text{Startgitter}$ 
3:  $\vec{u} := \vec{0}$ 
4: while TRUE do
5:    $\text{trainGrid}(G, \vec{u}, D)$ 
6:    $\text{acc} = \text{getAccuracy}(G, \vec{u}, D)$ 
7:   if  $\text{acc} \leq \text{acc}_{\text{needed}}$  then
8:     return  $\vec{u}, G$ 
9:   end if
10:   $\text{refineGrid}(G, \vec{u})$ 
11: end while
```

wichtigsten ist hierbei allerdings die Funktion *refineInitialGrid*. Diese verfeinert das Gitter, auf Basis der Payoff-Funktion, bevor der Löser für den ersten Zeitschritt gestartet wird. Falls an dieser Stelle der Knick aus Abbildungen 3.1 und 3.2 nicht entsprechend auf dem Gitter darstellt wird, so wirkt sich dies dramatisch auf die Lösungsqualität in allen folgenden Zeitschritten aus: Falls der erste Löseschritt auf ungenauen Daten arbeitet, so können in allen weiteren Schritten nur Werte basierend auf diesen errechnet werden. Während des Lösens der Black-Scholes Gleichung kann zwischen den einzelnen Zeitschritten das Gitter zusätzlich modifiziert werden. In Algorithmus 3.2 ist diese Funktionalität durch die Methode *restructureGrid* dargestellt. Abhängig von der Art der gewählten Option sind hier verschiedene Implementierungen denkbar. Betrachtet man eine gewöhnliche Option auf Underlyings ohne Dividendenausschüttung², so berechnet die Black-Scholes PDE eine Diffusion im Bereich des Strikes (also am Knick). Damit wird mit jedem Zeitschritt der Funktionsverlauf der Lösung glatter und somit die Verletzungen der Dünngitter-Annahmen geringer. Es werden im Bereich des Knicks nicht mehr so viele Gitterpunkte benötigt wie eingehend in der Routine *refineInitialGrid* erzeugt worden sind, um die Funktion im Dünngitterraum mit ausreichender Genauigkeit darzustellen. Daher kann *restructureGrid* verwendet werden, um das Gitter zu vergrößern. Auch hier wird analog zur Verfeinerung ein Überschuss-basierter Ansatz gewählt: Je kleiner der hierarchische Überschuss einer Ansatzfunktion ist, desto unwichtiger ist ihr Beitrag und die Stützstelle kann aus dem Gitter entfernt werden. Werden allerdings Dividenden ausgeschüttet, so kann sich der interessante Bereich während der Lösung verschieben. Da aber zu Beginn nur am Knick verfeinert worden ist, fehlen nun nötige Stützstellen, welche durch *restructureGrid* nachträglich erzeugt werden können. Das Lösen von Optionen auf Assets mit Di-

²Werden zusätzlich Dividenden betrachtet, so kann es zu einer Verschiebung des Knicks während des Lösens kommen.

videnden ist aber nicht Bestandteil der vorliegenden Arbeit, es sollte lediglich auf diese Möglichkeit aufmerksam gemacht werden.

Algorithmus 3.2 Adaptiver Algorithmus zum Lösen der Black-Scholes Gleichung. Mit Hilfe der Funktion *refineInitialGrid* wird das Gitter vor dem Lösen der Black-Scholes PDE so verfeinert, dass der Knick optimal dargestellt werden kann. *solveBlackScholes* löst die PDE für einen Zeitschritt der Größe $\delta\tau$; anschließend kann das Gitter basierend auf der letzten Lösung mittels *restructureGrid* angepasst werden.

```
1:  $G := \text{Startgitter}$ 
2:  $p(\vec{S}) := \text{Payoff-Funktion}$ 
3:  $\text{refineInitialGrid}(G, \vec{u}, p(\vec{S}))$ 
4: for  $\tau = 0$  to  $T$  do
5:    $\text{solveBlackScholes}(G, \vec{u}, \delta\tau)$ 
6:    $\text{restructureGrid}(G, \vec{u})$ 
7:    $\tau \leftarrow \tau + \delta\tau$ 
8: end for
9: return  $\vec{u}, G$ 
```

Zum Abschluss der Einführung in die Gitter-Adaptivität soll diese noch von einem anderen Blickwinkel aus betrachtet werden, welcher in Abbildung 3.3 dargestellt ist. Wie bereits erläutert, kann die Black-Scholes PDE beliebig genau numerisch gelöst werden³, da beliebig viele Stützstellen verwendet werden können. Mehr Stützstellen bedeuten automatisch einen steigenden Rechenaufwand, der zum Lösen der Gleichung benötigt wird. Besonders bei den vorgestellten Dünngitter-Algorithmen hat dies massive Auswirkungen: Mehr Gitterpunkte können ab einem bestimmten Punkt nur noch durch die Erhöhung der Level der Basisfunktionen erreicht werden. Damit werden deutlich mehr Rekursionen für die Berechnung der Up/Downs benötigt, was zu einem nicht-linearen Anstieg der Rechenzeit führt. Somit sind reguläre Dünne Gitter mit einer hohen Anzahl von Leveln auf jeden Fall zu vermeiden, da sonst sehr viel Zeit für die Lösung von uninteressanten Bereichen vergeudet wird. Aber auch im Falle der adaptiven Gitter gilt diese Regel in abgeschwächter Form. Wird zu tief oder zu viel im Bereich des Knicks verfeinert, steigt zwar die Lösungsgenauigkeit minimal weiter an, allerdings wird dies mit einer deutlich längeren Laufzeit erkauft. Es muss also neben der erreichten Genauigkeit auch auf die dafür benötigte Rechenzeit geachtet werden. Als optimalen Trade-Off zwischen diesen beiden gegenläufigen Zielen kann der Schnittpunkt beider Kurven (Genauigkeit, Rechenzeit) aus Abbildung 3.3 herangezogen werden. Abhängig vom konkreten Verlauf der Kurven, also den absoluten Beträgen, können auch Punkte, die leicht entfernt vom Schnittpunkt liegen, interessant sein, da z.B. mit ein wenig mehr Rechenzeit ein deutliche besseres Ergebnis erzielt werden kann. Aus diesem Grund ist der Bereich, in dem der Löser operieren sollte, mittels eines roten Rechtecks in Abbildung 3.3 hervorgehoben.

³Die maximale Genauigkeit ist beschränkt von der Maschinengenauigkeit, die von der verwendeten Architektur unterstützt wird.

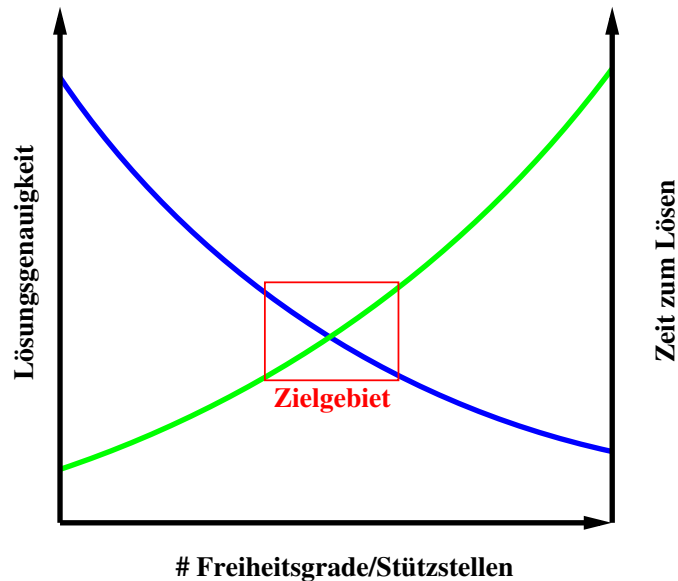


Abbildung 3.3: Bei der Verwendung von adaptiven Gittern ist neben der Lösungsgenauigkeit auch die benötigte Rechenzeit interessant. Optimal, als Trade-Off zwischen diesen gegenläufigen Zielen, ist das Gebiet um den Schnittpunkt (rot markiert).

3.2 Mögliche Adaptivitätskriterien beim Lösen der Black-Scholes Gleichung

Im letzten Abschnitt wurde allgemein die Verwendung der Adaptivität beim Lösen der Black-Scholes Gleichung eingeführt. Hierbei wurden die beiden Routinen *refineInitialGrid* und *restructureGrid* identifiziert, welche die Adaptivität abbilden. Im nun folgenden Abschnitt soll genauer auf diese Methoden eingegangen und dabei verschiedene Implementierungsmöglichkeiten beleuchtet werden. Allerdings werden beide nicht gleichwertig behandelt, da hier europäische Basket-Optionen ohne Dividendenzahlung im Fokus stehen. Es verändert sich also die Position des Strikes während der Laufzeit nicht und der Funktionsverlauf wird mit fortschreitender Zeit immer „glatter“. Somit ist *restructureGrid* von mindermem Interesse, da hier bei normalen europäischen Optionen lediglich eine Gittervergrößerung infrage kommt. Viel interessanter und auch schwieriger ist die Wahl von *refineInitialGrid*, denn es kann, wie bereits erwähnt, falls hier eine schlechte Konfiguration gewählt wird, die Genauigkeit für den kompletten Lösungsvorgang verunreinigt sein.

Zu Anfang soll anhand von einfachen Beispielen die Verfeinerungen von Gitterpunkten besprochen werden. So zeigt Abbildung 3.4 simple Gitterverfeinerungen. In beiden Darstellungen werden die roten Gitterpunkte verfeinert. Die linke Grafik zeigt eine Standard-Verfeinerung: Es müssen lediglich die hierarchischen Nachfah-

ren auf dem nächst-höheren Level erstellt werden. Nach dieser Operation besitzt das entstandene Gitter in allen Dimensionen eine vollständige hierarchische Basis. In diesem Zusammenhang bedeutet vollständig nicht, dass auf jedem Level alle Basisfunktionen existieren, sondern, dass zu einer Basisfunktion alle hierarchischen Vorfahren existieren und somit entlang der hierarchischen Basis navigiert werden kann.

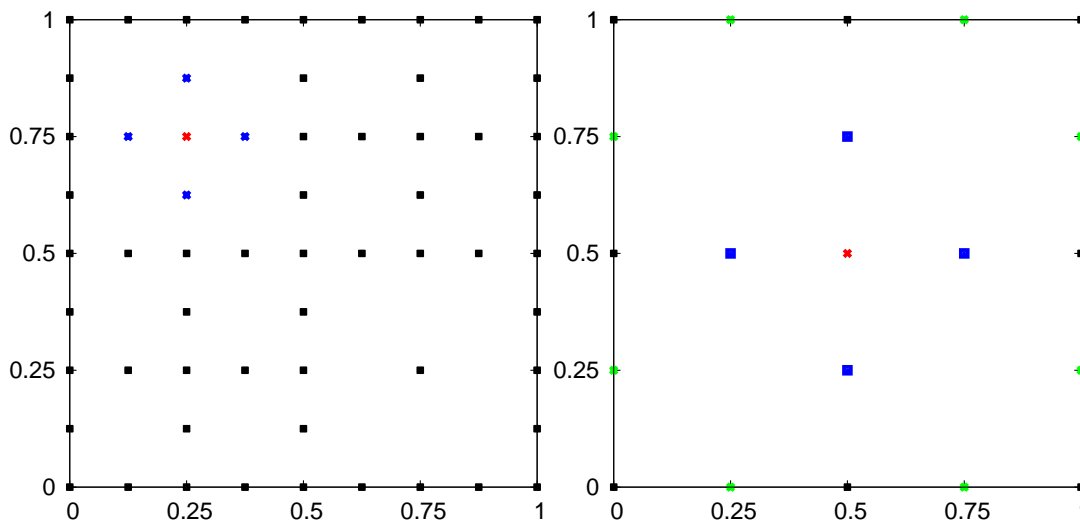


Abbildung 3.4: Beispiele für Verfeinerungen: Jeweils der rote Gitterpunkt wird verfeinert. Im linken Fall müssen nur die hierarchischen Nachfahren (markiert in blau) generiert werden. Etwas mehr ist im rechten Bild zu tun: Auch hier sind die blauen Punkte die hierarchischen Nachfahren. Allerdings müssen zusätzliche Punkte (grün dargestellt) auf Level 0 generiert werden, um in der zweiten Dimension eine vollständige hierarchische Basis herzustellen.

Anders sieht es beim rechten Gitter aus Abbildung 3.4 aus. Vor der Verfeinerung handelte es sich hierbei um ein zwei-dimensionales mit der Basisfunktion $(1, 1; 1, 1)$ und den dazugehörigen Basisfunktionen auf Level 0. Nun wird der Gitterpunkt $(1, 1; 1, 1)$ (rot markiert) verfeinert. Wie besprochen, werden die hierarchischen Nachfolger dieses Punktes auf Level 2 (blau dargestellt) generiert. Diese hängen aber „in der Luft“, da die hierarchischen Vorfahren auf Level 0 fehlen. Ein Beispiel: $(2, 1; 1, 1)$ besitzt zwar in der ersten Dimension alle Vorfahren, aber nicht in der zweiten. Aus diesem Grund müssen die Level-0-Basisfunktionen $(2, 1; 0, 0)$ und $(2, 1; 0, 1)$ erstellt werden. Diese acht zusätzlichen Punkte sind in der Grafik grün hervorgehoben.

Der zentralste Punkt bei der Verfeinerung eines Dünnes Gitters ist die Bestimmung der Punkte, die verfeinert werden sollen. Hierfür ist ein Algorithmus wünschenswert, der unabhängig von der darzustellenden Funktion die entsprechenden Punkte auswählt. An dieser Stelle kommen die hierarchischen Überschüsse aus dem letzten Abschnitt ins Spiel. Als Verallgemeinerung der ein-dimensionalen Betrachtung gilt, dass die hierarchischen Überschüsse ein Maß für die mehr-dimensionalen partiellen

zweiten Ableitungen sind. Nach [26] hat es sich als ein robustes Verfahren herausgestellt, den Betrag der hierarchischen Überschüsse der einzelnen Gitterpunkte als Indikator zu verwenden, ob dieser Gitterpunkt verfeinert werden soll. Intuitiv kann diese Wahl auch durch die Analysis nachvollzogen werden: Die (partiellen) zweiten Ableitungen sind ein Maß für die Krümmung der Funktion. Ist diese groß, so ist der Verlauf der Funktion in diesem Bereich als sehr „bauchig“⁴ einzustufen. Um eine bauchige Funktion mit linearen Basisfunktionen darzustellen, werden viele Stützstellen mit geringer Maschenweite benötigt. Da diese nur in dem Bereich der großen zweiten Ableitungen benötigt werden, sind die Gitter genau in diesem Bereich zu verfeinern.

Neben der Auswahl der richtigen Gitterpunkte ist auch die Anzahl der zu verfeinernden Gitterpunkte entscheidend. Hierfür sind zwei Methoden denkbar: Es können zum Einen die **ersten n Gitterpunkte** mit den größten hierarchischen Überschüssen verfeinert werden, oder zum Anderen kann ein **Schwellwert-basiertes** Vorgehen angewendet werden. In diesem Fall werden *alle* Gitterpunkte verfeinert, deren Überschüsse größer als ein bestimmter Wert sind. Bei einer Analyse von Abbildung 3.1 fällt auf, dass nur letzteres Verfahren bei der Black-Scholes Gleichung zu guten Resultaten führen kann: Die Payoff-Funktion besteht abgesehen vom Knick nur aus linearen Anteilen und kann somit mit wenigen linearen Basisfunktionen genau dargestellt werden. Somit sind nur die hierarchischen Überschüsse im Inneren des Gitters im Bereich des Knicks ungleich 0. Da der komplette Knick verfeinert werden sollte, ist die Schwellwert-basierte Methode anzuwenden, da nicht klar ist, wie viele Gitterpunkte auf oder in der Nähe des Knicks liegen, und somit die richtige Anzahl nur per Zufall getroffen werden kann. Die Bestimmung der richtigen Anzahl ist daher so schwierig, da sie von mehreren Parametern beeinflusst wird: Die Lage des Knicks im Gitter, die Anzahl der Assets im Basket (der Knick ist $d - 1$ -dimensional) und die Anzahl der Level im regulären Basis-Gitter, das die Grundlage für das verfeinerte Gitter bildet.

3.2.1 Initiale Verfeinerung

Nach der allgemeinen Einführung in die Funktionsweise der Gitterverfeinerungen wird nun die initiale Gitterverfeinerung vor dem Lösen der Black-Scholes Gleichung genauer vorgestellt. Da diese, wie im letzten Absatz erwähnt, ausschließlich auf den Überschüssen basiert, muss ein iteratives Verfahren angewendet werden. Dies rührt daher, da durch die Verfeinerung, dargestellt in Abbildung 3.4, pro Verfeinerung eines Gitterpunktes nur ein zusätzliches Level erstellt werden kann. Aus diesem Grund wird ein iteratives Verfahren benötigt, bei dem das Gitter mehrfach verfeinert wird, um eine ausreichende Auflösung am Knick zu gewährleisten.

⁴Es liegt kein glatter Funktionsverlauf mit kleinen (partiellen) zweiten Ableitungen vor.

Algorithmus 3.3 Die initiale Gitterverfeinerung ist ein iteratives Verfahren. Das ist dadurch bedingt, dass pro Verfeinerung nur ein zusätzliches Level entsteht. Insgesamt werden R Verfeinerungen durchgeführt.

```
1:  $G := \text{Startgitter}$ 
2:  $p(\vec{S}) := \text{Payoff-Funktion}$ 
3:  $\vec{u} \leftarrow \text{initSurpluses}(G, p(\vec{S}))$ 
4: for  $r = 1$  to  $R$  do
5:    $\text{refineGrid}(G, \vec{u})$ 
6:    $\vec{u} \leftarrow \text{initSurpluses}(G, p(\vec{S}))$ 
7: end for
8: return  $\vec{u}, G$ 
```

In Algorithmus 3.3 ist die initiale Verfeinerung in Pseudo-Code formuliert. Die Funktion *initSurpluses* initialisiert alle hierarchischen Überschüsse des Gitters so, dass die Dünngitter-Funktion die Payoff-Funktion interpoliert. *refineGrid* verfeinert das Gitter G basierend auf den hierarchischen Überschüssen. Hierbei bekommen die neu erstellten Gitterpunkte einen hierarchischen Überschuss von 0 zugewiesen. Dadurch wird erreicht, dass sich der Interpolant der Dünngitter-Funktion nicht ändert, da die neuen Punkte (vorerst) keinen Beitrag liefern. *refineGrid* arbeitet nach der oben beschriebenen Schwellwert-basierten Methode, um den gesamten Knick möglichst genau zu interpolieren. Als Nächstes werden die Überschüsse neu initialisiert. Damit erhalten auch die eben hinzugefügten Gitterpunkte im Allgemeinen Koeffizienten, die einen Beitrag ungleich 0 zum Dünngitter-Interpolanten liefern. Diese Kombination aus Gitterverfeinerung und Neuinitialisierung wird nun R -mal wiederholt, um den Knick hinreichend genau zu verfeinern.

Bei R Durchläufen enthält das entstandene Gitter maximal R zusätzliche Level. In der initialen Gitterverfeinerung können mit nur wenigen Durchläufen somit Gitter generiert werden, die lokal (am Knick) sehr tiefe Level-Strukturen besitzen. Allerdings bringen diese Gitter auch einen Nachteil mit sich: Der Knick in der Payoff-Funktion liegt im Allgemeinen schräg zu den Koordinatenachsen. Ein Umstand, der aus der Basket-Definition folgt. Die Verfeinerungen des Dünnen Gitters verlaufen nur parallel zu den Koordinatenachsen. Daher werden immer wieder Stellen im Gitter auftreten, an denen der Knick nicht exakt interpoliert wird, da keine Gitterpunkte auf ihm bzw. in unmittelbarer Nähe liegen. Ein solcher Fall ist in Abbildung 3.5 nochmals verdeutlicht. Hierbei handelt es sich um ein zweidimensionales Gitter, das mit Algorithmus 3.3 verfeinert worden ist. Es muss angemerkt werden, dass durch eine weitere Verfeinerung dieses Problem zwar an der gezeigten Stelle (feiner aufgelöste rote Bereiche) behoben werden kann, dadurch aber an einer anderen Stelle des Knicks ein ähnliches Problem entsteht (blaue Linie, hier wird der Knick immer noch sehr grob interpoliert). Es ist also mit der hier beschriebenen Gitter-Verfeinerung (die als klassische Dünngitter-Verfeinerung bezeichnet werden soll) **nicht** möglich, den Knick mit der im Gitter vorkommen-

den geringsten Maschenweite zu interpolieren, da immer wieder „Löcher“ auf der Diagonalen aufgrund der Dünngitter-Raumkonstruktion entstehen.

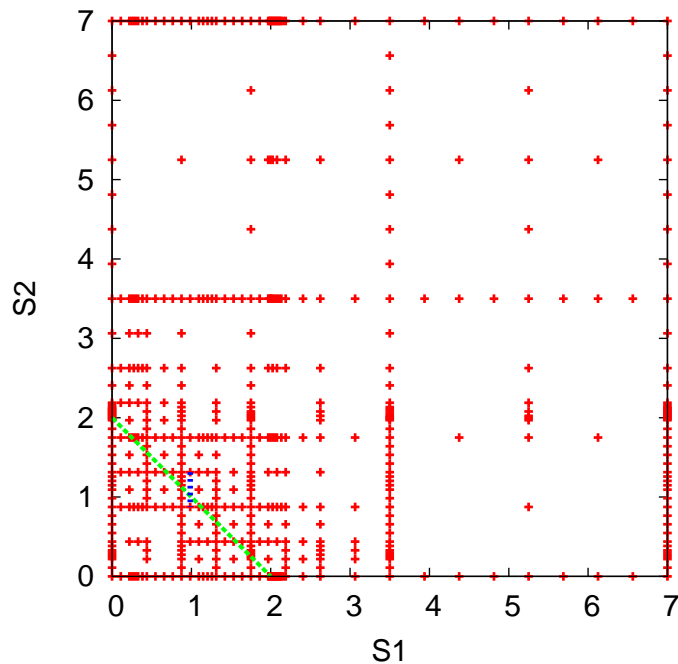


Abbildung 3.5: Dünnes Gitter, welches mittels der klassischen Verfeinerung verfeinert worden ist. Es wurde ein Startgitter mit 4 Leveln gewählt. Zusätzlich wurden 5 klassische Verfeinerungen dieses Gitters mit einem Schwellwert von 1^{-8} durchgeführt. Die grüne Linie markiert den Knick (für $K = 1$) in der Payoff-Funktion und die blaue Linie stellt die Interpolationsproblematik von nicht-achsenparallelen Verletzungen der Dünngitter-Glattheitsforderungen dar.

Daher soll noch eine weitere Variante für die Gitterverfeinerung vorgestellt werden. Diese ähnelt der bereits behandelten insoweit, dass ebenfalls die zu verfeinernenden Gitterpunkte durch ihrer Überschüsse bestimmt werden und nur die Punkte verfeinert werden, deren Überschüsse über einem bestimmten Schwellwert liegen. Allerdings kommt nun eine weitere Bedingung hinzu: Es werden nur Gitterpunkte verfeinert, deren Level kleiner als ein vorgegebenes maximales Level ist. Daher soll diese Art der Verfeinerung im Weiteren als maximal Level Verfeinerung, kurz max. Level Verfeinerung, bezeichnet werden. Damit kann die minimal auftretende Maschenweite des Gitters festgeschrieben werden. In Algorithmus 3.3 verändert sich hierbei nur die *refineGrid*-Methode. Während der Wiederholungen der Verfeinerung zerfällt die initiale Gitterverfeinerung in zwei Phasen:

Erzeugung neuer Level: In den ersten m Wiederholungen (m ergibt sich aus der Differenz des maximalen Levels und dem Level des regulären Startgitters) werden zusätzliche Level im Gitter generiert. In dieser Phase ist die max. Level Verfeinerung identisch mit dem Vorgehen bei der klassischen Verfeinerung.

Füllung der Level: In der zweiten Phase der max. Level Verfeinerung werden „offene Stellen“ im Dünneren Gitter aufgefüllt. Das geschieht durch die Verfeinerung von Gitterpunkten, die noch nicht auf dem maximalen Level liegen. Diese Verfeinerung stopft die bei der klassischen Verfeinerung als problematisch eingestufte Lücken in den adaptiven Dünngitter-Strukturen.

Somit kann durch die max. Level Verfeinerung ein lokales, volles Gitter erzeugt werden, wenn entsprechend viele Wiederholungen bei der initialen Gitterverfeinerung ausgeführt werden. Eine initiale Gitterverfeinerung basierend auf der max. Level Verfeinerung terminiert immer: Es kann während der Verfeinerungen die Gittergröße (Anzahl der Gitterpunkt) protokolliert werden. Das Verfahren wird solange wiederholt, bis das Gitter nicht mehr wächst. An diesem Punkt angekommen, ist das Gitter maximal verfeinert und ohne Veränderungen der Parameter (max. Level oder Schwellwert) werden bei einer nochmaligen Ausführung keine zusätzlichen Gitterpunkte mehr generiert.

In Abbildung 3.6 werden die max. Level Verfeinerung und klassische Verfeinerung miteinander verglichen. Beide Gitter besitzen im Inneren ungefähr gleich viele Freiheitsgrade (klassische Verfeinerung 344 Gitterpunkte, max. Level Verfeinerung 354 Gitterpunkte). Es ist gut zu erkennen, dass die max. Level Verfeinerung eine deutlich bessere Interpolation des Knicks erlaubt. Durch entsprechende Erhöhung der Anzahl der durchgeführten Verfeinerungen kann eine ähnliche Interpolationsgüte auch mit der klassischen Verfeinerung erreicht werden, allerdings sind in diesem Fall deutlich mehr Gitterpunkte nötig, um die in Abbildung 3.5 beschriebenen Schwierigkeiten durch mehr Verfeinerungen zu lösen.

Ebenfalls zeigt Abbildung 3.6 deutlich, dass der Knick auf seiner gesamten Länge verfeinert wird. Dies ist für sich genommen sehr gut. Meistens ist man jedoch bei der Bewertung von Optionen nicht an dem gesamten Bereich interessiert. Es sei hier z.B. auf die Monte Carlo Simulationen verwiesen. Diese berechnen lediglich den Preis einer Option für eine bestimmte Kombination der Kurse der Underlyings. Falls man also nur an einer bestimmten Kombination der Kurse zum Ausübungszeitpunkt interessiert ist, kann dieses a priori Wissen ebenfalls bei der Gitterverfeinerung verwendet werden.

Dabei ist dennoch darauf zu achten, dass ein Bereich um den gewählten Punkt verfeinert und dieser nicht zu eng gewählt wird. Hierbei wird das zu verfeinernde Gebiet mit Hilfe des Tensor-Produktes der ein-dimensionalen Normalverteilung modelliert, wie es in Gleichung (3.1) angegeben ist:

$$WeightNormal(\vec{s}) = \prod_d \frac{1}{\sigma_d^{refine} \cdot 2\pi} \exp \left[-0.5 \cdot \left(\frac{s_d - \mu_d^{refine}}{\sigma_d^{refine}} \right)^2 \right] \quad (3.1)$$

μ_d^{refine} wird für jedes Underlying so gewählt, dass es dem Kurs des Underlyings d entspricht, für den der Wert des Baskets bestimmt werden soll (Auswertestelle in

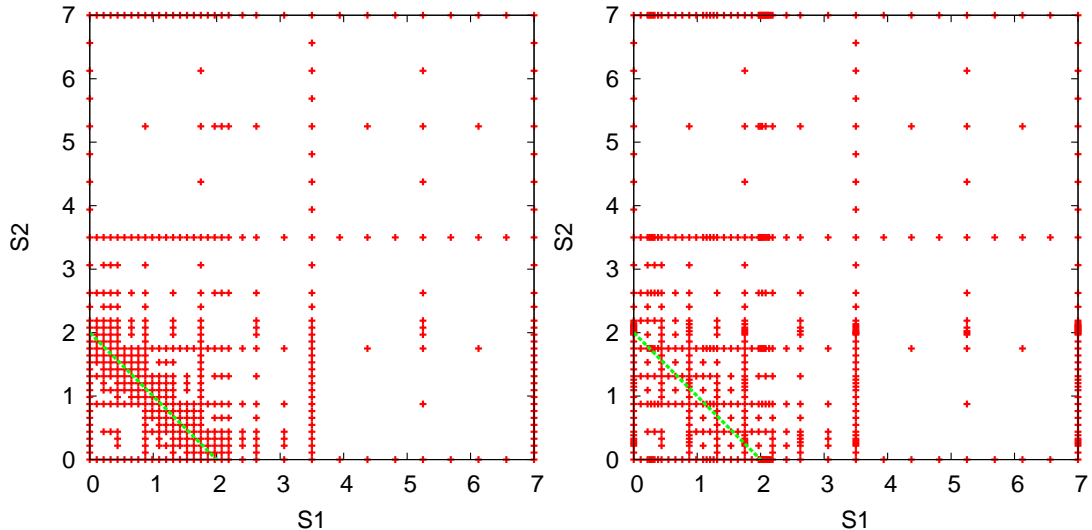


Abbildung 3.6: Vergleich der max. Level Verfeinerung (links) mit der klassischen Verfeinerung (rechts). In beiden Fällen wurde ein reguläres Gitter mit 4 Levels als Startgitter verwendet. Anschließend wurden Verfeinerungen mit Schwellwert 1^{-8} durchgeführt. Maximales Level, im linken Fall, war 6. Beide Gitter besitzen ca. 350 Gitterpunkte im Inneren. Für die klassische Verfeinerung wurden 5 Durchläufe für die max. Level Verfeinerung 7 Durchläufe ausgeführt. Der Knick für $K = 1$ ist in grün markiert.

Dimension d). σ_d^{refine} kann prinzipiell frei gewählt werden, in den später folgenden Analysen wird allerdings eine Abhängigkeit zum verwendeten Lösungsgebiet angegeben. Mit diesen beiden Parametern, die für jedes Asset des Baskets gewählt werden müssen, kann nun die „Wichtigkeit“ der Gitterpunkte bestimmt werden. Vor dem Verfeinern des Gitters wird für jeden Gitterpunkt seine Wichtigkeit bestimmt und anschließend die hierarchischen Überschüsse mit dieser gewichtet. Diese Werte bilden dann die Grundlage für die Entscheidung, welche Gitterpunkte verfeinert werden sollen und welche nicht. Die Anzahl der Verfeinerungen und die Schwellwerte sind von dieser Modifikation nicht betroffen und funktionieren weiterhin wie oben beschrieben.

Es ergibt sich somit für die gewichtete Verfeinerung Algorithmus 3.4. Dieser entspricht in weiten Zügen Algorithmus 3.3. Hinzugefügt wurde die Funktion *weightSurpluses*, welche die hierarchischen Überschüsse mit Gleichung (3.1) gewichtet. Diese veränderten Überschüsse sind anschließend die Eingabe für die Funktion *refineGrid*, die sich im Vergleich zu Algorithmus 3.3 in ihrer Implementierung **nicht** unterscheidet. In Abbildung 3.7 wurde als Punkt, an dem der Wert der Option bestimmt werden soll, $P(1|1)$ gewählt. Somit folgt $\mu^{refine} = 1$ für $\sigma^{refine} = 0.206$ gewählt (in Anlehnung an spätere Untersuchungen in den Abschnitten 3.3.2ff). Es zeigt sich deutlich, dass die Ränder des Knicks nicht mehr verfeinert werden (müssen), um den Preis der Option am gewählten Punkt zu bestimmen.

Algorithmus 3.4 Initiale Gitterverfeinerung erweitert um die gewichtete Verfeinerung. *weightSurpluses* gewichtet die Überschüsse auf Basis der in Gleichung (3.1) angegebenen Normalverteilung. Statt der einfachen Überschüsse werden die gewichteten Überschüsse zur Verfeinerung verwendet, um das Gitter nur in einem bestimmten Bereich zu verfeinern.

- 1: $G := \text{Startgitter}$
 - 2: $p(\vec{S}) := \text{Payoff-Funktion}$
 - 3: $\vec{u} \leftarrow \text{initSurpluses}(G, p(\vec{S}))$
 - 4: **for** $r = 1$ to R **do**
 - 5: $\vec{u}' \leftarrow \text{weightSurpluses}(G, \vec{u}, \vec{\mu}^{\text{refine}}, \vec{\sigma}^{\text{refine}})$
 - 6: $\text{refineGrid}(G, \vec{u}')$
 - 7: $\vec{u} \leftarrow \text{initSurpluses}(G, p(\vec{S}))$
 - 8: **end for**
 - 9: **return** \vec{u}, G
-

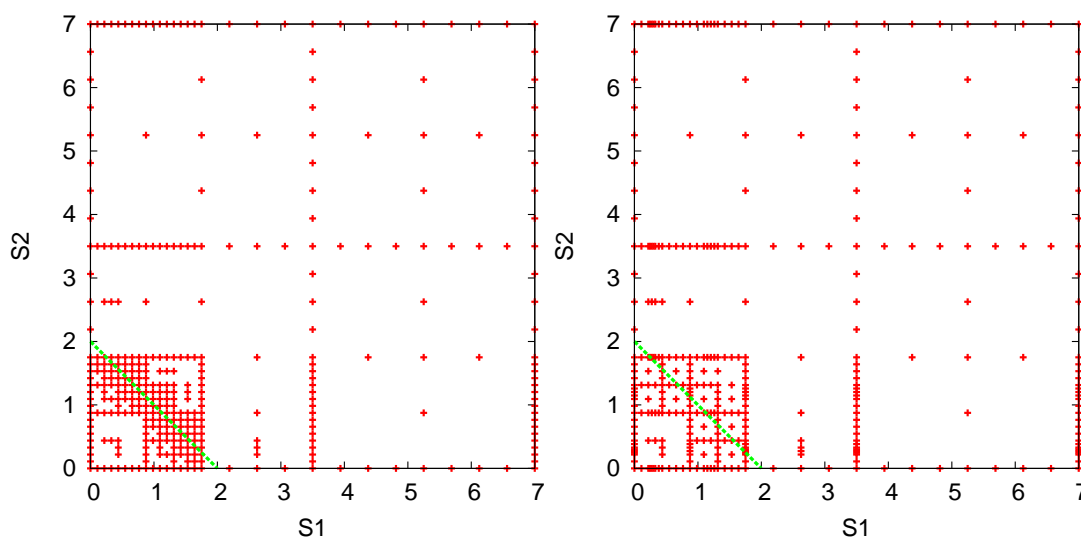


Abbildung 3.7: Gleiches 2-Asset Basket wie in Abbildung 3.6, allerdings wird nun nur noch in einem kleinerem Bereich um den Punkt $P = (1,1)$ verfeinert mit $\mu^{\text{refine}} = 1$ und $\sigma^{\text{refine}} = 0.206$. Links ist die max. Level- und rechts die klassische Verfeinerung dargestellt.

Mit der gewichteten Verfeinerung ist nun die Behandlung der initialen Gitterverfeinerung abgeschlossen. Allerdings muss noch ein weiterer Punkt in diesem Themenbereich geklärt werden. Alle Beispiele in den Abbildungen wurden auf Gitter mit kartesischen Koordinaten dargestellt. Bei der Formulierung der schwachen Form der Black-Scholes Gleichung wurde aber auch die Black-Scholes Gleichung mit log-transformierten Koordinaten behandelt, da ihr System aufgrund von konstanten Koeffizienten eine bessere Kondition besitzt, was zu weniger Iterationen während dem iterativen Lösungsvorgang führt. Daher gibt Abbildung 3.8 die Payoff-Funktionen in log-transformierten und kartesischen Koordinaten nach

der initialen gewichteten Gitter-Verfeinerung für zwei-Asset Calls und Puts an. Es zeigt sich in dieser Abbildung deutlich, dass die vorgestellten Verfeinerungsverfahren ohne Probleme auf Gitter mit log-transformierten Koordinaten angewendet werden können, da in diesen Beispielen die log-transformierten Überschüsse als Basis für die gewichtete Verfeinerung verwendet worden sind; Abbildung 3.8 gibt in der rechten Spalte die Gitter in kartesischen Koordinaten an. Hiermit wird klar, dass die log-transformierten hierarchischen Überschüsse ebenfalls am Knick der Payoff-Funktion besonders groß sind, da dieser Bereich in kartesischen Koordinaten stark verfeinert ist. Zudem macht die Abbildung noch einen weiteren massiven Vorteil der log-transformierten Koordinaten deutlich: Die meisten Gitterpunkte werden von Haus aus an der Stelle $P'(0|0)$ investiert, dieser Punkt entspricht nach Rücktransformation genau den Kursen, an denen Interesse am Wert der Option besteht:

$$P'(0|0) \rightarrow P(e^0|e^0) \rightarrow P(1|1)$$

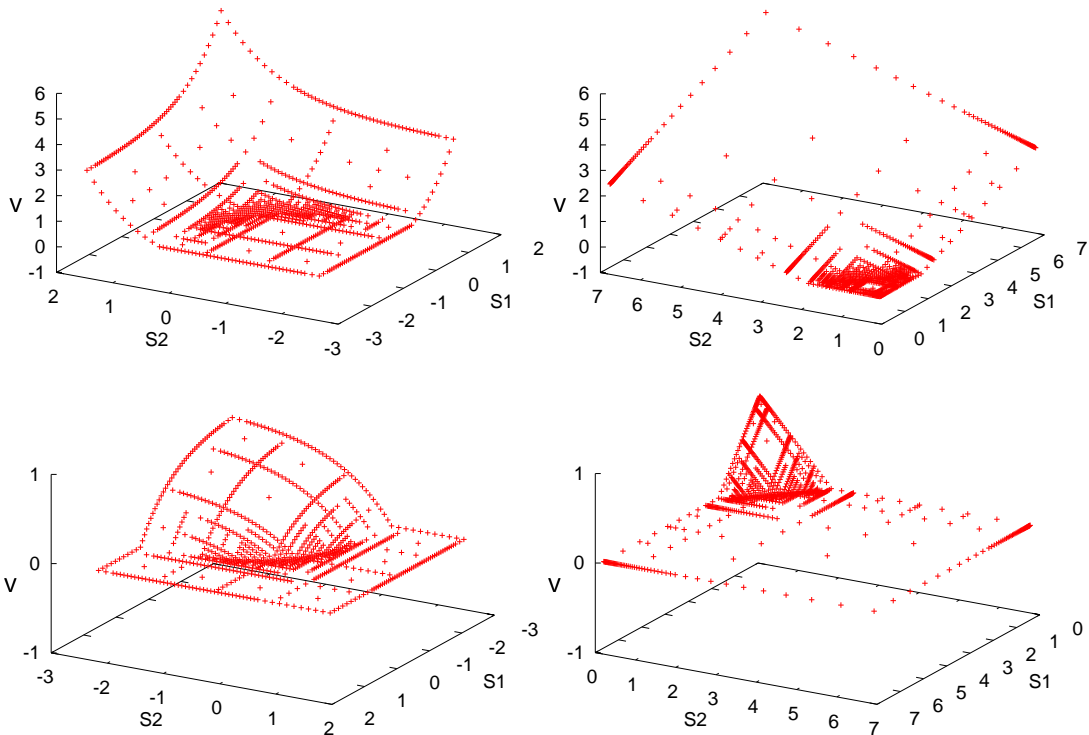


Abbildung 3.8: Payoff-Funktionen für Call (oben) und Put (unten). Dargestellt in log-transformierten Koordinaten (links) und den dazugehörigen kartesischen Koordinaten (rechts). Es handelt sich wieder um ein zwei-Asset Basket mit $K = 1$. Für diese Abbildung wurde auf einem regulären Gitter mit fünf Levels gestartet und sieben gewichtete Verfeinerungen bei einem max. Level von sechs durchgeführt.

Es muss an dieser Stelle angemerkt werden, dass das doppelt unendliche Gebiet im Falle des Calls bei log-transformierten Koordinaten zu Problemen führen kann. So erkennt man, dass bei dem recht kleinen log-transformierten Gebiet (der linke

Rand liegt bei 0,11 statt bei 0) der Funktionswert der Payoff-Funktion an den linken Rändern groß wird. Will man Achsenkoordinaten (Aktienkurse) kleiner als 0,11 darstellen, so wird an dieser Stelle der Funktionswert schnell in die Tausende steigen! Beim Put hingegen besteht dieses Problem nicht, jedoch ist der Knick nicht so weich wie beim Call, was zu mehr Iterationen und mehr benötigten Gitterpunkten während des Lösens führen kann.

Um die Vorteile der gewichteten Verfeinerung noch etwas genauer herauszustellen, wird in Abbildung 3.9 noch ein Put auf ein Basket bestehend aus drei Underlyings dargestellt. Hierfür wurden ebenfalls wie in der vorherigen Abbildung die Payoff-Funktion in log-transformierten und den dazugehörigen kartesischen Koordinaten gewählt. Es ist gut erkennbar, dass nur im Bereich des Punktes $P = (1, 1, 1)$ das Gitter stark verfeinert wird und aufgrund der max. Level Verfeinerung ergibt sich fast ein volles Gitter am Punkt P , was optimal für die Lösung der Black-Scholes Gleichung an dieser Stelle ist.

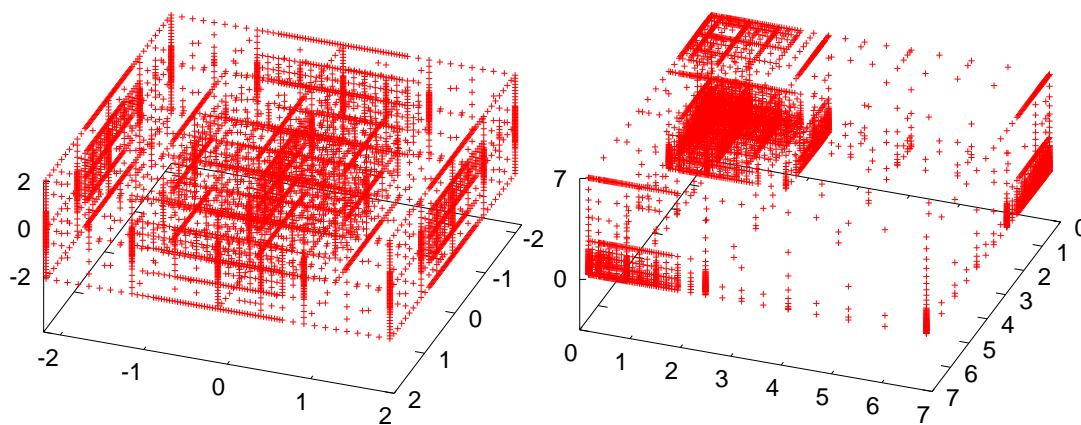


Abbildung 3.9: Dünnes Gitter zu einem Basket auf drei Underlyings. Verfeinert mit max. Level Verfeinerung (Level 7) und gewichteten Überschüssen. Log-transformierte Koordinaten sind links, kartesische Koordinaten sind rechts abgebildet.

3.2.2 Adaptivität während des Lösens

Zur Abrundung der Adaptivitätskriterien, die bei der Lösung der Black-Scholes Gleichung hilfreich sind, soll abschließende auf Möglichkeiten der Adaptivität während des Lösens eingegangen werden. Zwischen den einzelnen Zeitschritten kann das Gitter, analog zum Abzinsen der Ränder, modifiziert werden. Hierbei sind folgende Gitteroperationen möglich:

Verfeinerung: Analog zu allen Verfeinerungsszenarien, die bereits besprochen worden sind. Alle Gitterpunkte, deren Überschüsse größer als ein bestimmter Schwellwert sind, werden verfeinert. Hierbei besteht allerdings die Gefahr,

dass, wenn der Schwellwert zu klein gewählt wird, das Gitter zwar stark verfeinert wird, die Lösungsgenauigkeit aber praktisch nicht zunimmt und somit nur die Rechenzeit durch die Verfeinerungsmaßnahmen verlängert wird. Daher ist diese Art der Gittermodifikation sehr vorsichtig einzusetzen. Bei den europäischen Optionen haben Tests ergeben, dass sie keine nennenswerten Vorteile mit sich bringt. Anders dürfte es bei Optionen aussehen, die Dividenden auszahlen, da sich hier eine Verschiebung in der Payoff-Funktion ergibt.

Vergroberung: Hierbei handelt es sich um eine Gittermodifikation, die noch nicht genauer besprochen worden ist und als die Inverse der Gitterverfeinerung angesehen werden kann. Die Vergroberung arbeitet ebenfalls Überschuss-basiert. Jedoch werden nun nicht alle Punkte, die einen Überschuss *größer* als ein bestimmter Schwellwert haben verfeinert, sondern alle Gitterpunkte, die keine hierarchischen Nachfahren haben und deren Überschuss *kleiner* als ein bestimmter Schwellwert ist, werden aus dem Gitter entfernt. Somit kann das Gitter an diesen Stellen ausgedünnt werden, da diese Punkte nicht unbedingt zur Interpolation der Funktion benötigt werden. Besonders bei einer ausgedehnten initialen Verfeinerung bietet die Vergroberung mit steigender Anzahl von ausgeführten Zeitschritten die Möglichkeit, das Gitter schrittweise auszudünnen und somit langsam mit fortschreitender Glättung ein kleineres Gitter zu erzeugen.

Verfeinerung und Vergroberung: Diese letzte Variante ist keine neue, sondern lediglich die Kombination aus den soeben besprochenen Verfahren. Damit diese funktioniert, muss folgender Zusammenhang beachtet werden: Der Schwellwert für die Verfeinerung muss größer sein, als der Schwellwert für die Vergroberung. Ist dem nicht so, könnten Punkte, die verfeinert wurden, gleich wieder aus dem Gitter entfernt werden, bzw. Punkte die vergrößert worden sind sofort erneut verfeinert werden. In einer solche Situation wäre die Kombination beider Verfahren ergebnislos.

Der in dieser Arbeit implementierte Löser bietet alle drei Varianten für die Adaptivität während des Lösens an. Allerdings haben zahlreiche Tests gezeigt, dass beim Lösen von europäischen Baskets ohne Dividenden nur die Vergroberung eine Verbesserung in Hinblick auf beide Ziele, Lösungsgenauigkeit *und* Lösungsgeschwindigkeit, liefert. Diese Tests werden nun im nächsten Abschnitt detailliert vorgestellt.

3.3 Bewertung der Adaptivitätskriterien

Nach der Einführung der verschiedenen Adaptivitätskriterien sollen diese nun anhand von Tests bewertet werden. Aus den letzten Absätzen hat sich ergeben, dass zahlreiche Stellschrauben existieren. Daher sollen zunächst einige Annahmen und

Festlegungen vorgestellt werden, welche eine Einschränkung des Parameterraums erlauben. Anschließend erfolgt eine ausführliche Analyse des zwei-Asset Baskets. Mit deren Hilfe werden Heuristiken für die beste Parameterwahl entwickelt, welche danach auf Baskets mit drei bis fünf Underlyings angewendet werden.

3.3.1 Parameterannahmen und -einschränkungen

Die angesprochenen Stellschrauben sollen nun nochmals gesammelt aufgestellt werden. Das ist wichtig, damit die Einstellungsmöglichkeiten bewertet werden können. Die Bewertung soll dazu beitragen, den Parameterraum einzugrenzen, um somit eine Parameterstudie zur Bestimmung der optimalen Adaptivitätseinstellungen durchführen zu können. Zusammengefasst ergeben sich folgende Adaptivitätseinstellungen bei der Lösung der Black-Scholes Gleichung:

Feste Punktanzahl vs. Schwellwert: Wie bereits ausführlich dargestellt, existieren bei der Bestimmung der Punkte, welche verfeinert werden sollen, zwei unterschiedliche Verfahren. Zum Einen kann eine bestimmte Anzahl der Gitterpunkte mit den höchsten Überschüssen verfeinert werden, zum Anderen kann ein Schwellwert-basiertes Vorgehen angewendet werden. Hierbei werden alle Punkte mit einem Überschuss größer als einem vorgegebenen Schwellwert verfeinert. Aus bereits erläuterten Punkten ist letzteres Verfahren bei der Black-Scholes Gleichung sinnvoller.

Klassische vs. max. Level Verfeinerung: Ebenfalls wurden zwei verschiedene Arten der Gitterverfeinerung vorgestellt. Hierbei handelte es sich um die klassische Gitterverfeinerung bei der für einen zu verfeinernden Gitterpunkt alle hierarchischen Nachfolger auf dem nächsten Level erzeugt werden. Als zweite Option wurde die max. Level Verfeinerung vorgestellt, bei der ein max. Level beachtet werden muss und so ggf. Kinder nicht in allen Dimensionen erstellt werden.

Anzahl der initialen Verfeinerung: Es ist wichtig, dass die Startfunktion möglichst genau vor dem Lösen auf dem Gitter dargestellt wird. Dies kann durch mehrere und wiederholte Verfeinerungsschritte vor dem Lösen erreicht werden.

Anzahl der max. Level: Wird die max. Level Verfeinerung als Werkzeug gewählt, ergibt sich die Frage, wie dieses max. Level gewählt werden muss. Hierbei muss ein guter Kompromiss zwischen Lösungsgenauigkeit und -geschwindigkeit gefunden werden.

Anpassungen während des Lösens: Während des Lösens sind zwischen den Zeitschritten auch Gitteranpassungen möglich. Diese können aus (mehrfacher) Gittervergrößerung, -verfeinerung oder einer Kombination beider bestehen. Auch hier existieren wieder Parameter wie Schwellwerte, max. Level und Anzahl der adaptiven Gitteranpassungen.

Einschränkung des Verfeinerungsbereichs: Weiter oben wurde die gewichtete Verfeinerung besprochen. Durch diese entstehen weitere Parameter bei der initialen Gitterverfeinerung. Es ist darauf zu achten, dass ein Gleichgewicht zwischen der Standardverfeinerung und einer zu starken Einschränkung des Lösungsgebiets gefunden wird.

Somit ergeben sich weit über zehn Justagemöglichkeiten. Um diese etwas einzuschränken, werden folgende Einschränkungen getroffen:

- Das **Startlevel** spielt eine wichtige Rolle, da mit ihm die Bereiche abgesteckt werden, in denen (einfach) verfeinert werden kann. Dies gilt besonders für die max. Level Verfeinerung, da hier deutlich weniger zusätzliche Punkte erstellt werden.
- Bei den folgenden Tests wird ausschließlich das Schwellwert-basierte Verfahren (wie auch weiter vorne ausführlich dargestellt) verwendet. Dieser Schwellwert wird ebenfalls für alle eventuellen Verfeinerungen während des Lösens verwendet, d.h. es existiert ein Parameter: der **Verfeinerungsschwellwert**.
- Es kann zwischen der klassischen und der max. Level Verfeinerung als Verfahren für die initiale Verfeinerung gewählt werden. Falls eine Methode während des Lösens gewählt wurde, die eine adaptive Verfeinerung benötigt, wird hier dasselbe Verfahren wie für die initiale Gitterverfeinerung angewendet. Somit sind zwei weitere Parameter anzugeben: **Verfeinerungsart** und ggf. das **max. Level**.
- Die Anzahl der ausgeführten **Schritte der initialen Gitterverfeinerung** ist konfigurierbar.
- Während des Lösens kann **pro Zeitschritt nur eine Gitterveränderung** durchgeführt werden⁵. Für diese stehen folgende Möglichkeiten zur Verfügung: **Verfeinerung, Vergrößerung, Kombination aus Verfeinerung und Vergrößerung**.
- Die Intensität der Vergrößerung während des Lösens erfolgt Schwellwert-basiert analog zur Verfeinerung. Es muss ein für alle Zeitschritte konstanter **Vergrößerungs-Schwellwert** festgelegt werden.

Damit sind beim implementierten adaptiven Löser der Black-Scholes Gleichung diese sieben Parameter festzulegen. Im Folgenden soll nun studiert werden, wie die Wahl dieser Parameter am besten getroffen wird. Ein-dimensionale Optionen werden in dieser Arbeit nicht mehr betrachtet und es sei auf [16] verwiesen. Dort wird ebenfalls auf Unterschiede zwischen der numerischen und der analytischen Lösung im ein-dimensionalen Fall eingegangen.

⁵Im Falle einer Verfeinerung machen mehrere keinen Sinn, da die neu hinzugekommenen Gitterpunkte noch keinen nicht-null Überschuss besitzen.

3.3.2 Parameterstudie anhand eines 2-dimensionalen Baskets

Um ein Gefühl für die eben eingeführten Parameter zu bekommen, soll für Optionen mit zwei Underlyings eine Parameterstudie vorgestellt werden, bei der die sieben Verfeinerungsparameter systematisch überprüft werden. Hierzu wurde eine Put-Option auf Assets ohne Korrelationen und Drifts gewählt, um die Rechenzeit für die mehreren tausend Ausführungen so gering wie möglich zu halten⁶. Ebenfalls wurde der risikofreie Zinssatz $r = 0$ gesetzt und als Strike wurde $K = 1$ gewählt. Die Parameterstudie musste durchgeführt werden, da nicht klar war, wie sich die einzelnen Stellschrauben auf das Resultat auswirken. Um die Gitter bewerten zu können, wurde deren Ergebnis mit der Lösung eines regulären Dünnen Gitters mit 14 Leveln verglichen. Die Parameter, die die Adaptivität nur indirekt beeinflussen (da sie zu einer anderen Lösung und somit zu anderen Überschüssen führen) wurden für alle Tests festgehalten: Jede Option in der Parameterstudie wurde in 20 Zeitschritten mit einem gemischten impliziten Euler und Crank-Nicolson gelöst (die ersten drei Schritte sind reine implizite Eulerschritte), als ϵ für den BiCG-Stab wurde 10^{-5} gewählt. Als Wertebereiche für die sieben Adaptivitätsparameter wurde Folgendes gewählt:

Startlevel: $l \in [6 - 10]$

Verfeinerungsmethode: max. Level (mL) und klassische Verfeinerung (kV)

max. Level: $mL \in [10 - 14]$

initiale Verfeinerungen: #iV $\in [3 - 7]$

Schwellwert Verfeinerungen: $S_v \in [5 \cdot 10^{-6}, 10^{-5}, \dots, 5 \cdot 10^{-3}, 10^{-2}]$

Adaptivität während des Lösens: keine, Vergrößerung und integrierte Vergrößerung und Verfeinerung

Schwellwert Vergrößerungen: $S_c \in [10^{-9}, 5 \cdot 10^{-8}, \dots, 5 \cdot 10^{-7}, 10^{-6}]$

Tabelle 3.1 zeigt die besten acht Ergebnisse (mit kleinstem relativen Fehler, gemessen in der max. Norm), welche sehr eindeutig ausfallen. Die Tabelle ist aufsteigend nach der Maximumnorm des relativen Fehlers sortiert. Die ersten drei Zeilen sind jedoch unterschiedlicher, als man auf den ersten Blick vermuten würde. Im Sinne von Abbildung 3.3 ist die zweite die beste, da quasi die gleiche Genauigkeit mit weniger Freiheitsgraden in weniger Zeit⁷ erreicht wird. Allerdings ist noch nicht offensichtlich, ob gar ein anderer Test noch effizienter ist.

⁶Werden keine Korrelationen verwendet, so kann ein Up/Down Schema bei zwei Dimensionen übersprungen werden, da der Koeffizient desselben 0 ist.

⁷Die aufgeführten Zeiten gelten für die parallel Ausführung mit 12 Threads auf zwei Intel Xeon X5650 Prozessoren, vgl. Kapitel 4 für Details. Dies gilt für alle weiteren angegebenen Zeiten in diesem Abschnitt.

l	V	mL	#iV	S_v	AL	S_c	$\# \overline{GP}$	t [s]	$\ e\ _\infty$	$\ e\ _{L_2}$
10	mL	10	4	$5 \cdot 10^{-6}$	-	-	14822	75,4	1,6	0,6
10	mL	10	4	$5 \cdot 10^{-6}$	c	10^{-8}	12580	68,2	1,63	0,57
10	mL	10	4	$5 \cdot 10^{-6}$	c	10^{-9}	13725	72,9	1,68	0,59
10	mL	10	4	10^{-5}	c	10^{-8}	11328	59,5	1,7	0,82
10	mL	10	4	10^{-5}	-	-	13388	64,1	1,8	0,9
10	mL	11	4	10^{-5}	-	-	13630	88,0	1,8	0,5
10	mL	11	4	10^{-5}	c	10^{-8}	11558	79,3	1,87	0,49
10	mL	10	4	10^{-5}	c	10^{-7}	9504	49,5	1,88	0,8

Tabelle 3.1: Die acht besten Ergebnisse der Parameterstudie: l : Startlevel, V:Verfeinerungsmethode, #iV: Anzahl initiale Verfeinerungen, S_v : Schwellwert Verfeinerung, AL: Adaptivität während des Lösens (c: Vergrößerung), S_c : Schwellwert Vergrößerung, $\# \overline{GP}$: Durchschnittliche Anzahl von Freiheitsgraden, t: Zeit zum Lösen, e : relativer Fehler zu einem regulären Gitter mit 14 Leveln. Die $\|e\|_\infty$ und $\|e\|_{L_2}$ Werte verstehen sich $\cdot 10^{-5}$. Es wurde keine gewichtete Verfeinerung verwendet.

Aus diesem Grund muss eine Kennzahl entwickelt werden, mit der die Testläufe verglichen werden können. Für diesen Zweck sind verschiedene Vorgehensweisen denkbar. Hier wird ein Produktansatz gewählt:

$$Q(\|e\|) = \# \overline{GP} \cdot t \cdot \|e\| \quad (3.2)$$

Je kleiner Q ist, desto höher ist die Effizienz des korrespondierenden Gitters einzustufen. Dies kann man sich sehr einfach klar machen: Man sucht ein sehr kleines Gitter, auf dem in sehr kurzer Zeit mit minimalen Fehler die Black-Scholes Gleichung gelöst werden kann. Es sei an dieser Stelle angemerkt, dass es nur sinnig ist, die Q -Werte für Gitter mit ähnlichen relativen Fehlern zu bestimmen und zu vergleichen, da sonst verzerrte Ergebnisse entstehen können. Berechnet man Q für die Werte aus Tabelle 3.1, so ergeben sich die Werte aus Tabelle 3.2. Dadurch entsteht ein deutlich anderes Bild: Nun ist die letzte Zeile die beste, gefolgt von der vierten.

Dass eine Verfeinerung während des Lösens mit demselben Schwellwert wie bei der initialen Gitterverfeinerung nicht effizient ist, kann aus Abbildung 3.10 entnommen werden. Hier ist zu erkennen, dass der gesamte Glättungsbereich stark verfeinert wird. Diese Punkte kosten allerdings sehr viel Rechenzeit und liefern nur sehr kleine Beiträge an zusätzlicher Genauigkeit ($\leq 10^{-6}$, bei adäquater initialer Verfeinerung). Ggf. würde sich eine Verfeinerung mit einem deutlich höheren Schwellwert als bei der initialen Gitterverfeinerung und Gittern in kartesischen Koordinaten lohnen. Allerdings wurde ein solches Szenario der Einfachheit halber außen vor gelassen, da die Vergrößerung während des Lösens als best-geeignetste Verfahren in der Parameterstudie identifiziert wurde, vgl. hier auch das eingeführte Bewertungsmaß Q .

$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
18.18	7.18
14.02	4.92
16.79	5.97
11.58	5.5
15.53	7.68
22.15	6.18
17.12	4.49
8.83	3.76

Tabelle 3.2: Berechnung von Q für die Zeilen aus Tabelle 3.1 für $\|e\|_\infty$ und $\|e\|_{L_2}$.

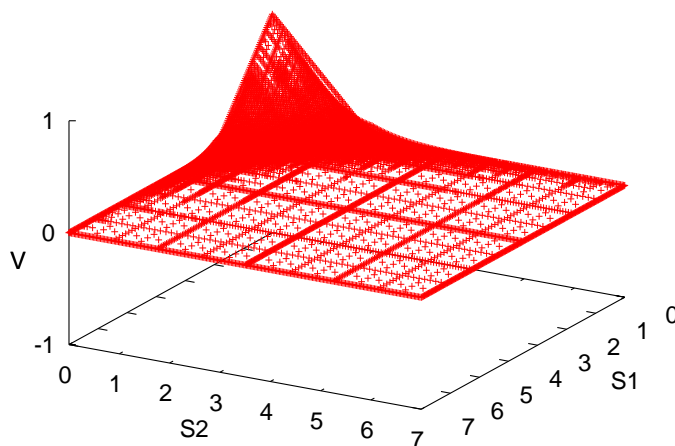


Abbildung 3.10: Lösungsgitter einer Put Basket-Option mit Verfeinerungen während des Lösen. Es werden zahlreiche zusätzlich Punkte erstellt, die nur geringfügig mehr Genauigkeit liefern, jedoch zu langen Rechenzeiten führen.

Aus diesen Erkenntnissen ergeben sich folgende optimale Bereiche für die Adaptivitätsparameterwahl für 2-dimensionale Basket-Optionen:

Startlevel: 10

Verfeinerungsmethode: max. Level

max. Level: $mL = 10$

initiale Verfeinerungen: $\#iV = 4$

Schwellwert Verfeinerungen: $S_v \in [5 \cdot 10^{-6}, 10^{-5}]$

Adaptivität während des Lösen: Vergrößerung

Schwellwert Vergrößerungen: $S_c \in [10^{-9}, 5 \cdot 10^{-8}, \dots, 5 \cdot 10^{-7}]$

Diese Wertebereiche der Adaptivitätsparameter führen zu Gittern, die zu einem regulären Dünnen Gitter mit 14 Leveln vergleichbare (und teilweise sogar besse-

re) Lösungsgenauigkeit liefern. Wird eine solche Qualität nicht verlangt, können natürlich auch schwächere Parameter (z.B. niedrigere Anzahl von Leveln, weniger initiale Verfeinerungen oder größere Schwellwerte) verwendet werden.

Damit ergibt sich eine deutliche Einschränkung des Parameterraums, welche nun verwendet wird, um Call und Put Optionen auf Gittern mit kartesischen und logarithmischen Koordinaten zu lösen. Als Vergleich werden reguläre Dünne Gitter auf den verschiedensten Leveln (2-14) herangezogen. Neben der kompletten Verfeinerung des Knicks, wie in der Parameterstudie durchgeführt, werden nun auch Gitter, welche sich der gewichteten Verfeinerung bedienen, in den Vergleich mit einbezogen.

Als Testfall wird eine europäische Basket-Option (beschrieben durch Gleichung (2.6)) mit Strike $K = 1$ gelöst. Zur Festlegung des Lösungsgebiets wurde vor der Berechnung der Optionspreise eine Gebietschätzung durchgeführt. Diese berücksichtigt die Drifts und Volatilitäten und ergab ein Gebiet von $\Omega = [0, 7]^2$. Im Fall der log-transformierten Koordinaten ist das Gebiet ein wenig kleiner am linken Rand zu wählen, da die 0 nicht dargestellt werden kann. Für die Drifts, Volatilitäten und Korrelationen wurde folgende Wahl getroffen:

$$\mu_i = \begin{pmatrix} 0.08 \\ 0.09 \end{pmatrix}, \quad \sigma_i = \begin{pmatrix} 0.3 \\ 0.4 \end{pmatrix}, \quad \rho_{i,j} = \begin{pmatrix} 1.0 & -0.5 \\ -0.5 & 1.0 \end{pmatrix}.$$

Der risikolose Zinssatz r wurde auf 0.05 gesetzt.

Tabellen 3.3 bis 3.6 zeigen die relativen Fehler der Lösung der Black-Scholes Gleichung für Call und Put Optionen auf ein zwei-Asset Basket. Als Referenzlösung dient die Lösung auf einem regulären Dünnen Gitter mit 14 Leveln (wie es bereits in der Parameterstudie verwendet worden ist). Es wird die Qualität der Lösung auf Gittern mit kartesischen und logarithmischen Koordinaten untersucht. Ebenfalls wird für beide Koordinatenvarianten die normale und gewichtete Adaptivität analysiert. Als Konfiguration der adaptiven Verfahren wurden die optimalen Parameter der oben erläuterten Studie verwendet, mit $S_v = 5 \cdot 10^{-6}$ und $S_c = 1 \cdot 10^{-8}$ in allen adaptiven Ausführungen. Das max. Level wurde immer gleich dem Startlevel gesetzt und es wurden vier initiale Verfeinerungen durchgeführt. Die Berechnung des relativen Fehlers erfolgte nicht an einer speziellen Auswertestelle, sondern auf einem äquidistanten Testgitter, welches den Strike ($K(1|1)$) umgibt⁸: $\Omega_T = [0.897, 1.13]^2$ mit 20 Testpunkten pro Dimension (insgesamt 400 Testpunkte). Im Falle der gewichteten Verfeinerung gilt für die Parameter der Verfeinerung-Normalverteilung: $\mu^{refine} = 1$ und $\sigma^{refine} = 0.206$, was der Ausdehnung des Testgebiets in jeder Dimension entspricht.

Aus dieser Vielzahl von Werten sollen zwei Punkte besonders herausgestellt werden. So ist zu erkennen, dass der Unterschied in Bezug auf den relativen Fehler

⁸Dies entspricht 3% der Ausdehnung des Lösungsgebietes Ω pro Dimension.

3. Adaptivität bei der Lösung der Black-Scholes PDE

Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#GP$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	2.40247	1.21291	5	0.018	-	-
3	3.76344	1.96499	17	0.019	-	-
4	2.32426	1.19848	49	0.043	-	-
5	0.390201	0.0951895	129	0.106	-	-
6	0.398284	0.21215	321	0.329	-	-
7	0.0810139	0.0409449	769	1.35	-	-
8	0.00846513	0.00250069	1793	5.76	-	-
9	0.0288496	0.0142233	4097	24.37	-	-
10	0.0145602	0.00783263	9217	108.4	-	-
11	0.00103236	0.000453487	20481	583.4	12335	5418
12	0.000643708	0.000226391	45057	2749.5	79744	28046
13	0.000206794	6.93884e-05	98305	14588.3	296564	99510

S.-Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#\overline{GP}$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	2.80222	1.46591	8	0.039	-	-
3	0.282899	0.260274	27	0.160	-	-
4	0.21108	0.175195	87	0.175	-	-
5	0.0681386	0.0443243	228	0.44	-	-
6	0.0312861	0.0178644	517	1.29	-	-
7	0.0051334	0.00198442	1143	4.28	-	-
8	0.00424327	0.00108553	2277	12.85	-	-
9	0.0012441	0.000386662	4562	44.96	255	79
10	0.000761656	0.000235168	8721	140.8	935	289
11	0.00096492	0.000543222	16395	503	7958	4480

2	2.40247	1.21168	5	0.021	-	-
3	3.76344	1.96311	17	0.035	-	-
4	0.216476	0.181737	66	0.112	-	-
5	0.0714262	0.0471599	180	0.341	-	-
6	0.0260769	0.0127265	442	0.98	-	-
7	0.0107813	0.0038516	1008	3.32	-	-
8	0.00526223	0.00133131	2095	10.55	-	-
9	0.00125214	0.000349245	4281	36.40	195	54
10	0.000699989	0.000245005	8237	117.7	679	238
11	0.00107343	0.000603235	15451	428.3	7103	3992

Tabelle 3.3: 2d Call Option, kartesische Koordinaten: Relative Fehler, (mittlere) Gittergrößen und Laufzeiten für reguläre Dünne Gitter für die Level 2-13 (oben). Ergebnisse für adaptive Gitter stehen in der unteren Tabelle für Startlevel 2-11; normale Adaptivität (Mitte), gewichtete Adaptivität (unten).

für entsprechend fein aufgelöste Gitterstrukturen zwischen kartesischen und log-transformierten Koordinaten recht klein ausfällt. Der größte Unterschied ist bei der adaptiv gerechneten Put Option festzustellen (Faktor 10), da ein Strike ($K = 1$)

3. Adaptivität bei der Lösung der Black-Scholes PDE

Level	$\ e\ _\infty$	$\ e\ _{L_2}$	#GP	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	0.904288	0.759957	5	0.018	-	-
3	0.7426	0.484541	17	0.018	-	-
4	0.0921601	0.037423	49	0.039	-	-
5	0.0326608	0.0179457	129	0.12	-	-
6	0.0431631	0.0171448	321	0.36	-	-
7	0.0599658	0.0331147	769	1.13	-	-
8	0.0291637	0.0169868	1793	3.12	-	-
9	0.00976236	0.00576398	4097	8.37	-	-
10	0.00296767	0.00178616	9217	24.389	-	-
11	0.000322952	9.38191e-05	20481	109.7	726	211
12	0.000436913	0.000226301	45057	439.4	8650	4480
13	0.000113232	3.87426e-05	98305	2073	23075	7895

S.-Level	$\ e\ _\infty$	$\ e\ _{L_2}$	# \overline{GP}	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	1.07745	1.00867	9	0.034	-	-
3	0.416857	0.368216	44	0.066	-	-
4	0.134954	0.0945136	133	0.20	-	-
5	0.0410186	0.0251688	384	0.67	-	-
6	0.0132184	0.00728304	925	1.96	-	-
7	0.00302691	0.00143336	2051	5.16	-	-
8	0.00127821	0.000611046	4291	13.4	-	-
9	0.000431025	0.000138628	8257	31	110	35
10	0.000287789	3.91806e-05	14017	66.7	269	37
11	0.00013944	3.62227e-05	24121	164	551	143

2	0.943811	0.831955	8	0.033	-	-
3	0.402679	0.351164	33	0.047	-	-
4	0.0589586	0.0270804	91	0.12	-	-
5	0.035132	0.0212609	238	0.39	-	-
6	0.00977429	0.0046243	548	1.02	-	-
7	0.00308279	0.00146287	1276	3.08	-	-
8	0.000820993	0.000295701	2718	7.55	-	-
9	0.00046428	6.44377e-05	5667	18.4	48	7
10	0.000155141	2.65226e-05	11241	47.3	82	14
11	0.000144609	4.6812e-05	21778	149	469	152

Tabelle 3.4: 2d Call Option, log-transformierte Koordinaten: Relative Fehler, (mittlere) Gittergrößen und Laufzeiten für reguläre Dünne Gitter für die Level 2-13 (oben). Ergebnisse für adaptive Gitter stehen in der unteren Tabelle für Startlevel 2-11; normale Adaptivität (Mitte), gewichtete Adaptivität (unten).

im Testfall gewählt wurde, der nahe an der linken Gebietsgrenze liegt. Hier ist es im Fall der kartesischen Koordinaten auf Grund des kleinen Gebiets im Bereich des Strikes nicht möglich, eine hinreichend feine Gitterstruktur aufzubauen. Bei

3. Adaptivität bei der Lösung der Black-Scholes PDE

Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#GP$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	7.46535	4.22087	5	0.017937	-	-
3	10.4072	5.43931	17	0.020877	-	-
4	7.07158	3.81055	49	0.044723	-	-
5	1.26867	0.428585	129	0.139933	-	-
6	0.749971	0.59255	321	0.458338	-	-
7	0.150081	0.0864092	769	2.00467	-	-
8	0.022174	0.0074248	1793	8.97153	-	-
9	0.0579399	0.0409492	4097	40.82	-	-
10	0.0287685	0.021888	9217	201.35	-	-
11	0.00190403	0.00109776	20481	1121.28	43726	25210
12	0.00244226	0.000900764	45057	6566.6	722595	266510
13	0.000198414	0.000141042	98305	36525.7	712437	506434

S.-Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#\overline{GP}$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	10.2222	5.37837	8	0.028519	-	-
3	1.96075	0.739473	25	0.060181	-	-
4	1.31766	0.660152	73	0.163181	-	-
5	0.182777	0.128678	169	0.481626	-	-
6	0.0833813	0.0533591	364	1.44927	-	-
7	0.01783	0.00768988	780	4.2474	-	-
8	0.00864014	0.00290897	1435	10.0105	-	-
9	0.00398708	0.00148107	2668	27.3797	291	108
10	0.00225003	0.000786428	4944	87.525	974	340
11	0.00345409	0.00186879	9056	333.967	10447	5652

2	7.46535	4.23205	5	0.029826	-	-
3	10.4072	5.45733	17	0.046903	-	-
4	1.398	0.688851	64	0.142067	-	-
5	0.192881	0.137475	172	0.548541	-	-
6	0.0480443	0.0304286	414	1.58787	-	-
7	0.0428296	0.0142793	833	4.11389	-	-
8	0.0105899	0.0034962	1505	9.19459	-	-
9	0.00295123	0.00118548	2744	22.9903	186	75
10	0.00226455	0.000909822	4753	63.8421	687	276
11	0.00369433	0.00202141	8712	244.425	7867	4305

Tabelle 3.5: 2d Put Option, kartesische Koordinaten: Relative Fehler, (mittlere) Gittergrößen und Laufzeiten für reguläre Dünne Gitter für die Level 2-13 (oben). Ergebnisse für adaptive Gitter stehen in der unteren Tabelle für Startlevel 2-11; normale Adaptivität (Mitte), gewichtete Adaptivität (unten).

log-transformierten Koordinaten kann dies nicht passieren, da dort das Testgebiet mittig im kompletten Lösungsgebiet liegt, vgl. Abbildung 3.8.

Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#GP$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	3.45201	1.53236	5	0.018533	-	-
3	1.25995	0.762253	17	0.021233	-	-
4	1.30521	0.35581	49	0.051208	-	-
5	0.422462	0.131962	129	0.138113	-	-
6	0.039234	0.0316168	321	0.423441	-	-
7	0.149543	0.0981089	769	1.52132	-	-
8	0.0790724	0.0507115	1793	4.45425	-	-
9	0.0245301	0.0167208	4097	12.4913	-	-
10	0.008196	0.00534881	9217	39.7779	-	-
11	0.000726327	0.000298387	20481	178.16	2650	1089
12	0.000948408	0.000650511	45057	741.087	31668	21721
13	0.00017409	9.42788e-05	98305	3711.15	63512	34395

S.-Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#\overline{GP}$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	3.70008	1.71068	9	0.035224	-	-
3	1.28972	0.620197	44	0.06973	-	-
4	0.18837	0.132234	133	0.25717	-	-
5	0.0773505	0.0430269	377	0.944622	-	-
6	0.029516	0.0145591	909	2.75716	-	-
7	0.00609121	0.00233168	1985	7.28204	-	-
8	0.00302153	0.00142087	4119	19.5604	-	-
9	0.00081598	0.000285086	7698	42.7742	269	94
10	0.000175858	5.32117e-05	12570	89.3207	197	60
11	0.00026276	0.000120227	20686	206.093	1120	513

2	3.53457	1.63942	8	0.027303	-	-
3	1.03048	0.52747	33	0.060928	-	-
4	0.697539	0.1918	91	0.169531	-	-
5	0.0577422	0.0301539	237	0.549284	-	-
6	0.0116133	0.0052058	533	1.42725	-	-
7	0.00639563	0.00243928	1244	4.37801	-	-
8	0.00102892	0.000339571	2568	10.8446	-	-
9	0.000518112	0.000155154	5180	26.3589	71	21
10	0.000120886	3.38916e-05	9811	62.7836	74	21
11	0.000282056	0.000136385	18009	177.286	901	435

Tabelle 3.6: 2d Put Option, log-transformierte Koordinaten: Relative Fehler, (mittlere) Gittergrößen und Laufzeiten für reguläre Dünne Gitter für die Level 2-13 (oben). Ergebnisse für adaptive Gitter stehen in der unteren Tabelle für Startlevel 2-11; normale Adaptivität (Mitte), gewichtete Adaptivität (unten).

Im Allgemeinen kann die Lösung mit Hilfe der log-transformierten Koordinaten als die bessere Variante bezeichnet werden. Dies ist so sogar aus zwei Betrachtungswinkeln der Fall: Zum Einen sind die relativen Fehler bei log-transformierten Ko-

ordinaten geringer, da per Design die Gitter im Bereich des Strikes (und damit des Knicks) mehr Gitterpunkte aufweisen. Zum Anderen ist, wie bereits bei der Formulierung des log-transformierten Problems erwähnt, die Kondition des zu lösenden linearen Gleichungssystems aufgrund der konstanten Koeffizienten deutlich besser. Daher werden deutlich weniger Iterationsschritte beim Lösen des Gleichungssystems benötigt. Insgesamt ergibt sich somit mit log-transformierten Koordinaten eine bessere Lösung in deutlich geringerer Zeit. Dies wird nochmals durch die Werte des eingeführten Maßes Q für die regulären Gitter in den Tabellen 3.3 bis 3.6 hervorgehoben. Dieser wurde für die Ausführungen mit den kleinsten relativen Fehler berechnet. Hier sind die Werte für log-transformierte Koordinaten um bis zu einen Faktor von 10 kleiner. Der größte Anteil dieses Wertes ist auf die geringere Lösungszeit zurückzuführen.

Weiterhin ist gut aus den Tabellen zu entnehmen, dass die adaptiven Gitterstrukturen zusätzlich die Lösungszeit erheblich reduzieren können, da sie dieselbe Genauigkeit bei einer deutlich geringeren Anzahl von Gitterpunkten erreichen. Auch die aggressivere gewichtete Verfeinerung führt zu keinem nennenswerten Genauigkeitsverlust gegenüber den normal verfeinerten Gittern. Durch sie können, abhängig von der Option, nochmals bis zu 25 % der Gitterpunkte eingespart werden. Dies führt zu exzellenten Ergebnissen, wie z.B. im Falle der Put Option auf Gittern mit log-transformierten Koordinaten: Im Vergleich zu regulären Dünnen Gittern kann hier die Option mit einem Zehntel der Gitterpunkte ($98305 \leftrightarrow 9811$) und einem Fünftel der Zeit ($3711s \leftrightarrow 62,8s$) bei einem kleineren relativen Fehler gelöst werden. Der Vorteil des gewichteten adaptiven Gitters kann zusätzlich hervorgehoben werden, wenn die Q -Werte beider Gitter betrachtet werden: Der Wert des adaptiven Gitters ist knapp um einen Faktor 430 kleiner! Generell ist zu den Q -Werten aller adaptiven Gitter für den 2d Fall festzuhalten, dass diese nicht schlechter als im Falle der Parameterstudie sind. Wie bereits bei der Erläuterung der Studie festgestellt worden ist, basiert diese auf numerisch besser konditionierten Optionen und somit können deutlich kleinere relative Fehler bei gleichen Adaptivitätseinstellungen erreicht werden. Dies hat per Definition direkten Einfluss auf den Wert von Q . Berücksichtigt man diesen Effekt, so sind die Q -Werte der Konvergenzanalyse als gleichwertig zu denen aus der Parameterstudie einzustufen.

Da durch die Tabellen 3.3 bis 3.6 die Minimierung des relativen Fehlers in Abhängigkeit von der Anzahl der Freiheitsgrade (Gitterpunkte) nur schwer verdeutlicht werden kann, zeigen die Abbildungen 3.11 und 3.12 die Entwicklung des relativen Fehlers in Abhängigkeit der Freiheitsgrade und im Vergleich für kartesische und log-transformierte Koordinaten. Im Allgemeinen betrachtet kann für die gewichteten adaptiven Gitter eine sehr gute Konvergenz ausgemacht werden, welche auch deutlich glatter als im Falle der regulären Gitterstrukturen ist. Die Zacken im Fall der regulären Dünnen Gitter sind auf Dünngitter-Effekte zurückzuführen, da der Knick nicht exakt interpoliert werden kann. So kann es vorkommen, dass ein Dünnes Gitter mit weniger Leveln ein besseres Ergebnis liefert. Eine mögliche Erklärung hierfür sind hoch-frequente Fehlerterme, die auf groben Gittern nicht

entstehen können. Auch wird der bereits angesprochene numerische Vorteil der Gitter mit log-transformierten Koordinaten deutlich. Die dort erreichten relativen Fehler liegen für reguläre und adaptive Gitterstrukturen unterhalb der der Gitter mit kartesischen Koordinaten. Allerdings zeigt sich, dass für reguläre Gitter mit 13 Level (knapp 100000 Gitterpunkte) der Abstand zwischen kartesischen und log-transformierten Koordinaten sehr klein wird, da in beiden Fällen das Gebiet sehr fein diskretisiert wird und somit der Vorteil der log-transformierten Koordinaten nachlässt. Zum Abschluss muss noch auf eine Beobachtung bei den relativen Fehlern der adaptiven Gitter eingegangen werden: Es zeigt sich, dass jener für die Gitter mit den meisten Freiheitsgraden wieder zunimmt. Mit Hilfe der Tabellen 3.3 bis 3.6 und der Tatsache, dass 4 initiale max. Level Verfeinerung mit Startlevel und max. Level 11 durchgeführt werden, zeigt sich, dass diese Gitter am Knick feiner als ein reguläres Gitter mit 14 Level aufgelöst sind. Damit kommt das Maß des relativen Fehlers an seine Grenzen, da die berechneten Werte auf den adaptiven Gittern womöglich genauer als die Referenzwerte sind.

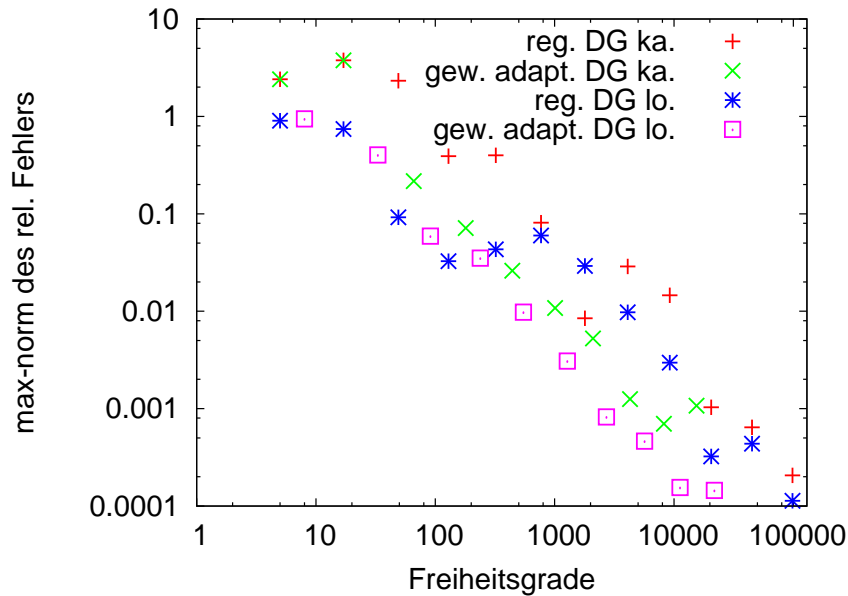


Abbildung 3.11: Konvergenz der 2d Call Option: Vergleich von Gittern mit kartesischen und log-transformierten Koordinaten sowie von regulären und gewichteten adaptiven Gitterstrukturen.

Es sollen nun noch einige allgemeine Bemerkungen zu den gezeigten relativen Fehlern gemacht werden. Vergleicht man die relativen Fehler der vorangegangenen Tabellen und Abbildungen, so stellt man fest, dass diese deutlich (um ca. einen Faktor 10) größer sind, als die in der Parameterstudie gezeigten relativen Fehler. Dies hängt mit den Parametern der gelösten Optionen zusammen. Für die ausführliche Konvergenzanalyse der regulären und adaptiven Gitter wurde Optionen mit Korrelation und Driftwerten $\neq 0$ verwendet. In der Parameterstudie hingegen wurden all diese Parameter auf 0 gesetzt, um die Rechenzeit so gering wie möglich zu halten.

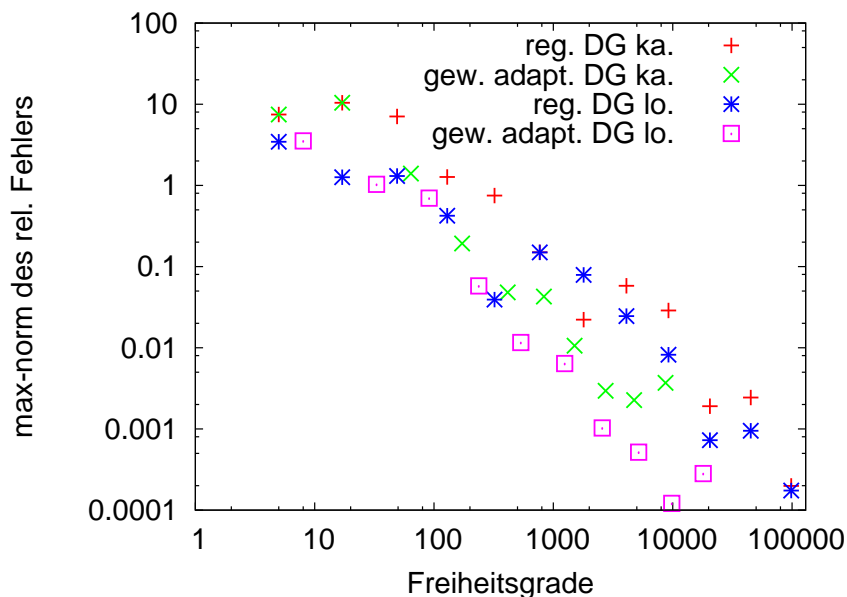


Abbildung 3.12: Konvergenz der 2d Put Option: Vergleich von Gittern mit kartesischen und log-transformierten Koordinaten sowie von regulären und gewichteten adaptiven Gitterstrukturen.

Auch die absolute Größe des relativen Fehlers soll zum Abschluss in den Mittelpunkt gestellt werden. Werte von 10^{-5} und einem max. Gitterlevel von 14-15 können auf den ersten Blick wie ein Widerspruch aussehen. Dass nur solch „große“ relativen Fehler mit sehr feinen Gitterstrukturen erreicht werden können, liegt an der Anfangslösung der PDE, der Payoff-Funktion. Der Knick ist nur sehr schwer auf einem (adaptiven) Dünnen Gitter darstellbar. Somit müssen deutlich mehr als 14 Level am Knick investiert werden, um den initialen Interpolationsfehler unter 10^{-6} zu drücken. Abbildung 3.13 zeigt den Interpolationsfehler entlang des Knicks für eine Put Basket-Option auf zwei Assets. Das Gitter wurde den Knick entlang an 2^{14} äquidistanten Punkten ausgewertet. Vor diesen Auswertungen wurde das Gitter am Knick zu einem vollen Level 12 Gitter verfeinert. Unter diesen Umständen kann ein ausgezeichneter Löser den relativen Fehler nicht sehr klein werden lassen. Somit sind die gezeigten relativen Fehler als sehr gute Ergebnisse einzuordnen. Für die Anwendung mit echten Finanzmarktdaten sind die hier gezeigten relativen Fehler zudem mehr als ausreichend, da die Konfiguration der stochastischen Prozesse (die Korrelationen, die Standardabweichungen und die Drifts) aus dem Markt geschätzt werden. Da deren Wert vom verwendeten Schätzer abhängt, ist insgesamt von geringerer Genauigkeit als 10^{-5} auszugehen.

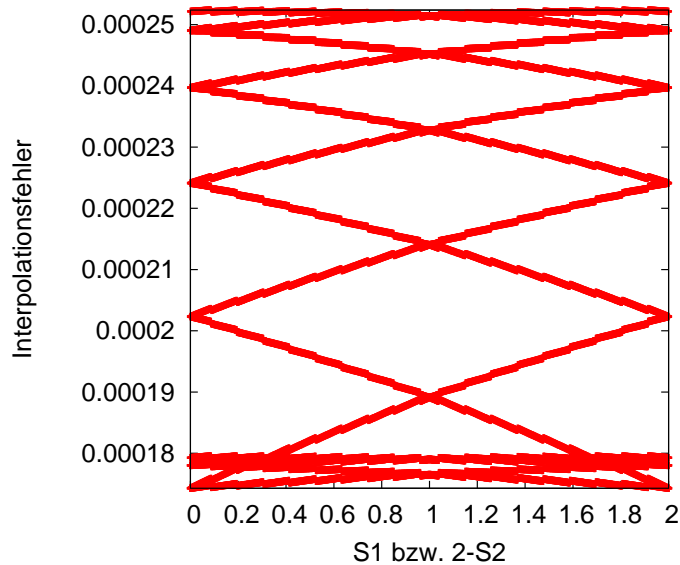


Abbildung 3.13: Interpolationsfehler entlang des Knicks einer Put Basket-Option auf zwei Assets mit $K = 2$ und 2^{14} Testpunkten entlang des Knicks. Das Gitter wurde am Knick zu einem vollen Level 12 Gitter verfeinert. Es werden einzelne Punkte dargestellt, die zu Linien verschmelzen. Das Zick-Zack-Muster entsteht durch die verschiedenen Level der hierarchischen Basis.

3.3.3 Verallgemeinerung auf 3-dimensionale Baskets

Die im letzten Abschnitt behandelten zwei-Asset Basket-Optionen können ohne Probleme auch auf vollen Gitterstrukturen gelöst werden und sind daher als Studienobjekt für Parameter interessant, aber nicht das Anwendungsziel des Dünngitter-Lösers. Aus diesem Grund sollen nun im vorliegenden Abschnitt Optionen auf drei Underlyings betrachtet werden. Um einfache Vergleichbarkeit zum zwei-dimensionalen Test zu gewährleisten, wird eine sehr ähnliche Konfiguration gewählt. Als Testoption dienen wieder europäische Baskets mit Strike $K = 1$ nach Formel (2.6). Das Lösungsgebiet wird um eine Dimension erweitert und lautet nun $\Omega = [0, 7]^3$. Auch bleibt das Testgebiet zur Berechnung der Normen der relativen Fehler erhalten und ist $\Omega_T = [0.897, 1.13]^3$. Als Referenz dienen wieder die regulären Dünnen Gitter. Allerdings muss aufgrund des erhöhten Rechenaufwands das maximale Level der regulären Tests um eins auf 13 Level verringert werden. Somit können die Fehlernormen für reguläre Gitter mit bis zu 12 Level bestimmt werden. Behält man die Konstruktion des Dünngitterraums im Kopf, so ist dies nicht optimal: Wird die Dimension um eins erhöht, so muss auch das Level um eins erhöht werden, um ein Gitter mit vergleichbarer Auflösung zu erhalten. Dies liegt an dem diagonalen Schnitt durch das Teilraumschema aus Abbildung 2.5. Da bei der Lösung der Black-Scholes Gleichung feinere Gitterstrukturen hauptsächlich am Knick der Payoff-Funktion benötigt werden, werden adaptive Dünngitterstrukturen mit steigender Anzahl von Assets immer wichtiger, da im regulären Fall sonst

die Levelanzahl zu stark erhöht werden muss. Neben der Gitterkonstruktion ist es im höher-dimensionalen Raum auch immer schwieriger die Payoff-Funktion genau zu interpolieren, weil der Knick $d - 1$ -dimensional ist. Dies lässt sich bereits bei den 3d Ergebnissen, weiter unten dargestellt, erkennen.

Neben diesen grundlegenden Einstellungen sollen die Erkenntnisse aus den zwei-Asset Tests berücksichtigt werden: So werden nur noch Gitter mit log-transformierten Koordinaten verwendet und die entsprechenden Adaptivitätsparameter werden (leicht verändert) übernommen. Deren Anpassung wird wegen der beschriebenen Anpassung des max. regulären Levels und der Konstruktion des Dünngitterraums notwendig. So wird das max. Startlevel der adaptiven Test auf 9 begrenzt und die Schwellwerte der Gitterverfeinerungen und Vergrößerungen angepasst. Da nun weniger Gitterpunkte zu Beginn vorhanden sind, werden etwas aggressivere Verfeinerungseinstellungen und weichere Vergrößerungseinstellungen gewählt: Der Schwellwert der Verfeinerungen sinkt auf 10^{-6} und der Schwellwert der Vergrößerung steigt auf 10^{-7} . Die Anzahl der initialen Verfeinerungen bleibt fixiert auf 4. Zahlreiche Tests haben gezeigt, dass dieses Vorgehen im Allgemeinen zu besseren Ergebnissen führt als z.B. die Verwendung eines höheren max. Verfeinerungsniveaus. Die Einstellungen der gewichteten Verfeinerung wurden im Vergleich zu den zwei Asset Optionen nicht verändert.

Die Parameter der stochastischen Prozesse lauten wie folgt:

$$\mu_i = \begin{pmatrix} 0.1 \\ 0.02 \\ 0.04 \end{pmatrix}, \quad \sigma_i = \begin{pmatrix} 0.2 \\ 0.3 \\ 0.4 \end{pmatrix}, \quad \rho_{i,j} = \begin{pmatrix} 1.0 & -0.7 & -0.1 \\ -0.7 & 1.0 & 0.1 \\ -0.1 & 0.1 & 1.0 \end{pmatrix},$$

und als risikoloser Zinssatz wurden wieder 5% gewählt.

Die für diese Konfiguration erreichten Parameter sind in den Tabellen 3.7 und 3.8 angegeben. Es ist gut zu sehen, dass durch die kleinen Anpassungen der Adaptivitätseinstellungen zum zwei-Asset Fall vergleichbare Ergebnisse erreicht werden können. So zeigt sich wieder im Falle der Put Option, dass durch die Verwendung der gewichteten Adaptivität ein wenig mehr als ein Zehntel der Freiheitsgrade (274431 vs. 40081) ausreichen, um die Option mit einem halb so großen relativen Fehler zu lösen. Weil dadurch wieder deutlich weniger Zeit zum Lösen benötigt wird, werden für diese Fälle die Q -Maße angegeben. Ebenfalls aus diesem Betrachtungswinkel kann erneut die Vergleichbarkeit zu den zwei Asset Optionen hergestellt werden. Der Faktor zwischen dem Q -Wert für reguläre und adaptive Gitter sinkt allerdings im Vergleich zur zwei-Asset Option deutlich spürbar auf ca. 130. Hier schlägt der bereits angesprochene 2-dimensionale Knick in der Startbedingung der drei Asset Option zu Buche, der eine Lösung auf noch größeren Gittern schwieriger macht.

Um die Analyse der drei-Asset Optionen abzuschließen, soll das Konvergenzverhalten aus den Tabellen 3.7 und 3.8 zusätzlich in den Abbildungen 3.14 und 3.15

Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#GP$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	2.61005	0.623917	7	0.031	-	-
3	1.85397	0.6742	31	0.068	-	-
4	0.352268	0.038854	111	0.307	-	-
5	0.0913812	0.0131712	351	1.505	-	-
6	0.0398136	0.0148387	1023	6.239	-	-
7	0.0721545	0.0339802	2815	21.92	-	-
8	0.0289888	0.00812017	7423	75.48	-	-
9	0.0580644	0.0269943	18943	265.5	-	-
10	0.0228643	0.0122212	47103	1078	-	-
11	0.00324796	0.00129236	114687	4418	-	-
12	0.00299958	0.000797527	274431	18100	14904703	3962856

S.-Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#\overline{GP}$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	1.53913	1.32487	26	0.088	-	-
3	0.533623	0.450835	297	1.84	-	-
4	0.187721	0.12573	1492	16.86	-	-
5	0.0840141	0.0492579	4670	74.2	-	-
6	0.034041	0.0185497	11999	229.4	-	-
7	0.00631436	0.00302543	28390	739.8	-	-
8	0.00413005	0.00106899	63318	2806	-	-
9	0.00163452	0.000218293	131173	8599	1843760	246237

2	1.20181	0.84073	16	0.06	-	-
3	0.597391	0.475388	194	0.84	-	-
4	0.199091	0.0366465	492	3.56	-	-
5	0.0799685	0.0489208	1692	21.66	-	-
6	0.0157646	0.00428461	3859	61.24	-	-
7	0.00751164	0.00442182	9018	166.3	-	-
8	0.00538188	0.00166269	19389	446.7	-	-
9	0.00188739	0.000418312	41588	1567	122979	27256

Tabelle 3.7: 3d Call Option, log-transformierte Koordinaten: Relative Fehler, (mittlere) Gittergrößen und Laufzeiten für reguläre Dünne Gitter für die Level 2-12 (oben). Ergebnisse für adaptive Gitter stehen in der unteren Tabelle für Startlevel 2-9; normale Adaptivität (Mitte), gewichtete Adaptivität (unten).

veranschaulicht werden. Hier zeigt sich, dass im Vergleich zur zwei Asset Option, im Falle der regulären Gitterstrukturen deutlich mehr Gitterpunkte benötigt werden, um eine gute Konvergenzrate zu erlangen. Jedoch ist der Konvergenzplot für viele Freiheitsgrade immer noch verhältnismäßig „unruhig“. Dies ist auf die bereits angesprochenen Eigenschaften der Dünngitterraumkonstruktion zurückzuführen. Ebenfalls zeigt der Verlauf der Kurven der adaptiven Gitter, dass die kleinen Anpassungen der Adaptivitätseinstellungen zu einem glatten Verlauf der Konvergenzkurve führt. An dieser Stelle muss drauf hingewiesen werden, dass für den

3. Adaptivität bei der Lösung der Black-Scholes PDE

Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#GP$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	3.64097	1.14717	7	0.042	-	-
3	0.398491	0.243738	31	0.062	-	-
4	0.98639	0.185131	111	0.328	-	-
5	0.332946	0.0829119	351	1.78	-	-
6	0.102867	0.0343827	1023	8.67	-	-
7	0.226654	0.0596487	2815	35.72	-	-
8	0.0480036	0.0149928	7423	122.1	-	-
9	0.142295	0.065064	18943	441.5	-	-
10	0.0936035	0.0326639	47103	2113	-	-
11	0.0136698	0.003727	114687	8847	-	-
12	0.0068063	0.00142353	274431	32940	61527299	12868365

S.-Level	$\ e\ _\infty$	$\ e\ _{L_2}$	$\#\overline{GP}$	t in [s]	$Q(\ e\ _\infty)$	$Q(\ e\ _{L_2})$
2	4.76836	1.69234	26	0.093	-	-
3	1.55331	0.540424	297	2.58	-	-
4	0.22955	0.140461	1490	25.15	-	-
5	0.199973	0.0875549	4661	118.59	-	-
6	0.119184	0.0414448	11960	407.99	-	-
7	0.0173646	0.00628273	28090	1325.8	-	-
8	0.00536064	0.00142746	62391	5328	-	-
9	0.000801879	0.000209575	128059	16412	1685313	440465

2	4.17204	1.54341	16	0.057	-	-
3	0.928456	0.34812	154	1.0	-	-
4	0.757758	0.117525	492	5.35	-	-
5	0.300651	0.100064	1691	33.9	-	-
6	0.0732887	0.0115867	3855	107.5	-	-
7	0.0351403	0.0121343	8946	310	-	-
8	0.0111547	0.00393551	18985	820.4	-	-
9	0.00382783	0.00121391	40081	3012	462160	146564

Tabelle 3.8: 3d Put Option, log-transformierte Koordinaten: Relative Fehler, (mittlere) Gittergrößen und Laufzeiten für reguläre Dünne Gitter für die Level 2-12 (oben). Ergebnisse für adaptive Gitter stehen in der unteren Tabelle für Startlevel 2-9; normale Adaptivität (Mitte), gewichtete Adaptivität (unten).

drei-dimensionalen Fall (wegen des deutlich erhöhten Rechenbedarfs) keine Parameterstudie durchgeführt worden ist. Die langwierige Durchführung einer solchen könnte unter Umständen noch bessere Ergebnisse bei der Verwendung von adaptiven Dünne Gittern erlauben. Da die jetzigen Resultate bereits eine enorme Steigerung bedeuten und der Raum unter der Verwendung der aktuellen Einstellungen schon recht genau diskretisiert wird, liegt die Annahme nahe, dass andere Einstellungen nur noch minimale Verbesserungen zulassen können und es ist somit fraglich, ob der enorme Aufwand einer Parameterstudie rentabel ist. Zusätzlich zu

dem hier durchgeführten, recht systematischen Abklopfen der Adaptivitätsparameter, sind zahlreiche Tests „ins Blaue“ gemacht worden. Mit einem dieser Ergebnisse war es z.B. möglich, in etwas geringerer Zeit auf gut 30000 Gitterpunkten den Put mit gleichem relativen Fehler wie auf dem regulären Gitter mit 12 Level zu bewerten ($6 \cdot 10^{-3}$). Allerdings besaß das Gitter aufgrund der größeren Fehlernorm ein schlechteres Q -Maß. Im Falle dieser Ausführung wurden leicht abgeänderte Einstellungen gewählt: Das max. Level war um eins größer als das Startlevel gewählt und es wurden 5 initiale Verfeinerungen durchgeführt. Durch dieses Beispiel wird nochmals deutlich, wie schwierig die Wahl der „optimalen“ Parameter ist, da es zum Einen mehrere Größen gibt, für die optimiert werden kann und zum Anderen alle Möglichkeiten im drei-dimensionalen Raum nicht schnell durchgerechnet werden können.

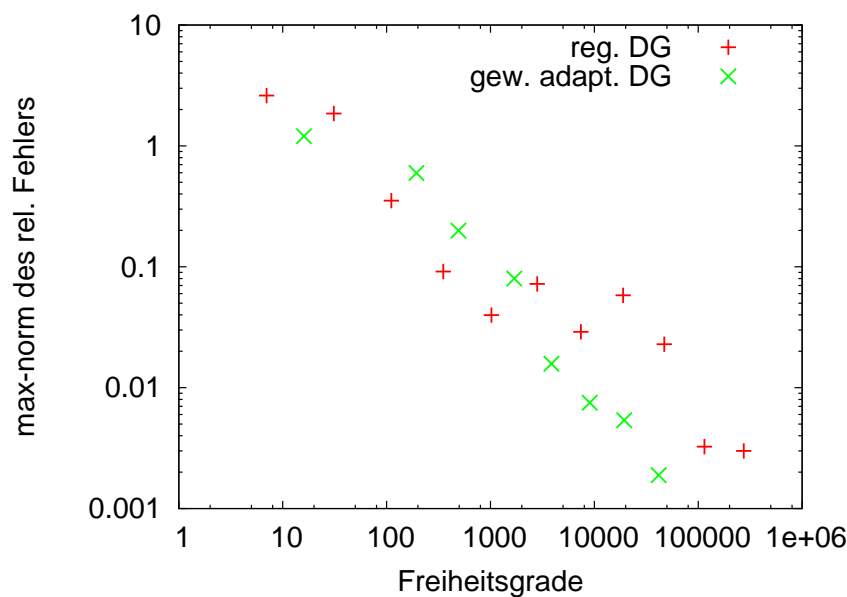


Abbildung 3.14: Konvergenz der 3d Call Option: Vergleich von regulären und gewichteten adaptiven Gitterstrukturen mit log-transformierten Koordinaten.

3.3.4 Höher-dimensionale Baskets

Nach der ausführlichen Behandlung der zwei und drei Asset Optionen wird nun auf Optionen mit mehr als drei Assets eingegangen. Diese sind z.B. interessant, für global importierende oder exportierende Unternehmen, wie in der Automobil- oder Unterhaltungselektronikbranche. Für ein gutes Geschäftsergebnis benötigen sie optimale Wechselkurse und können zum Hedging dieser eine Basket-Option auf mehrere verschiedene Fremdwährungen abschließen.

Es hat sich in den Abbildungen 3.14 und 3.15 bereits gezeigt, dass die Konvergenz für reguläre Gitter bei drei-Asset Optionen spürbar abnimmt. So bestätigen die

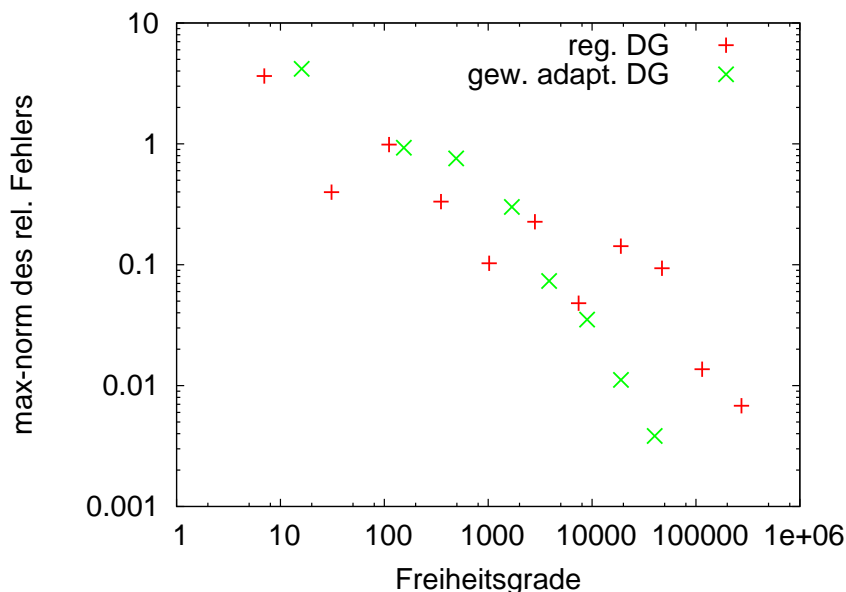


Abbildung 3.15: Konvergenz der 3d Put Option: Vergleich von regulären und gewichteten adaptiven Gitterstrukturen mit log-transformierten Koordinaten.

Konvergenzbetrachtungen für vier und fünf Underlyings in den Abbildungen 3.16 und 3.17 diesen Eindruck. Es ist klar, dass solche Ergebnisse nicht mehr als Referenzwerte für die Bewertung der adaptiven Gitterstrukturen herangezogen werden können. Aus diesem Grund wird zur Bewertung der Lösung mittels adaptiven dünnen Gittern auf den punktuellen Vergleich mit einer Monte Carlo Simulation mit 10^{10} Pfaden ausgewichen, wie sie in Anhang A angegeben ist.

Für die Konvergenzanalysen aus Abbildungen 3.16 und 3.17 und den noch zu erläuternden adaptiven Tests wurden im Fall vier Asset Optionen folgende Parameter der stochastischen Prozesse gewählt:

$$\mu_i = \begin{pmatrix} 0.05 \\ 0.05 \\ 0.05 \\ 0.05 \end{pmatrix}, \quad \sigma_i = \begin{pmatrix} 0.4 \\ 0.25 \\ 0.3 \\ 0.4 \end{pmatrix}, \quad \rho_{i,j} = \begin{pmatrix} 1.0 & 0.1 & -0.4 & 0.2 \\ 0.1 & 1.0 & 0.3 & -0.1 \\ -0.4 & 0.3 & 1.0 & 0.0 \\ 0.2 & -0.1 & 0.0 & 1.0 \end{pmatrix}.$$

Für die fünf Asset Optionen fiel die Wahl auf:

$$\mu_i = \begin{pmatrix} 0.05 \\ 0.05 \\ 0.05 \\ 0.05 \\ 0.05 \end{pmatrix}, \quad \sigma_i = \begin{pmatrix} 0.4 \\ 0.25 \\ 0.3 \\ 0.4 \\ 0.35 \end{pmatrix}, \quad \rho_{i,j} = \begin{pmatrix} 1.0 & 0.1 & -0.4 & 0.2 & 0.1 \\ 0.1 & 1.0 & 0.3 & -0.1 & 0.0 \\ -0.4 & 0.3 & 1.0 & 0.0 & 0.2 \\ 0.2 & -0.1 & 0.0 & 1.0 & -0.7 \\ 0.1 & 0.0 & 0.2 & -0.7 & 1.0 \end{pmatrix}.$$

Wie zuvor wird für alle Bewertungen ein risikoloser Zinssatz von 5% angenommen.

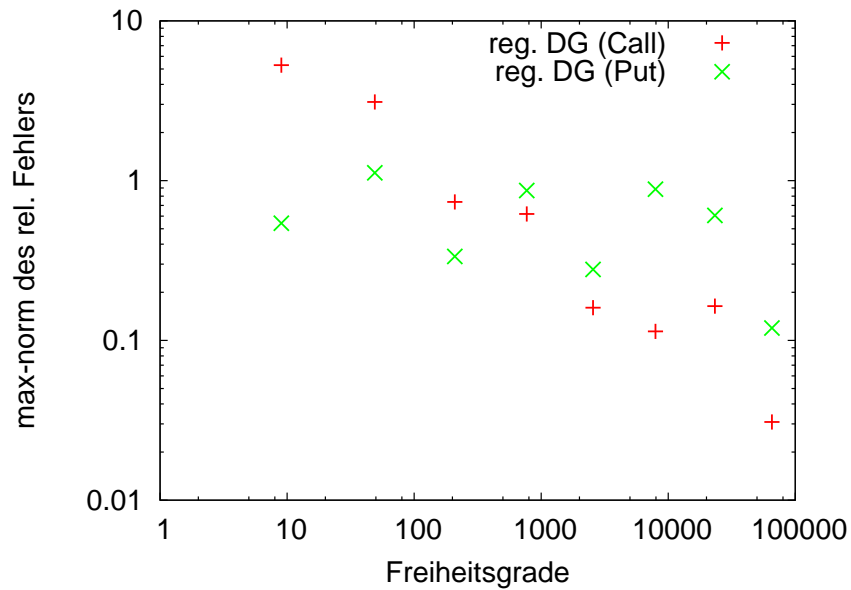


Abbildung 3.16: Konvergenzverhalten von 4d Call und Put Optionen auf regulären Dünnen Gittern. Es ist keine nennenswerte Konvergenz bis Level 9 zu erkennen.

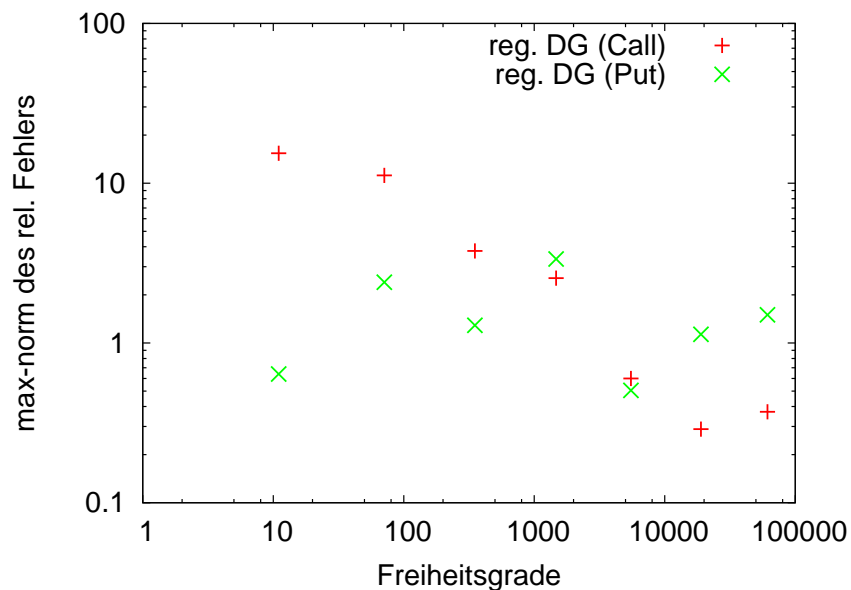


Abbildung 3.17: Konvergenzverhalten von 5d Call und Put Optionen auf regulären Dünnen Gittern. Es ist keine nennenswerte Konvergenz bis Level 8 zu erkennen.

Auch für die hier behandelten höher-dimensionalen Fälle müssen die Adaptivitätseinstellungen angepasst werden. So bleibt der Schwellwert der Verfeinerung bei 10^{-6} und der Schwellwert für die Vergrößerung bei 10^{-7} . Jedoch ändern sich die Start- und max. Level sowie die Anzahl der initialen Gitterverfeinerungen. Da durch den Vergleich mit Monte Carlo Simulationen nicht mehr die Norm auf ei-

nem Testgebiet Ω_T berechnet werden kann, ist es hilfreich, das Verfeinerungsgebiet der initialen Gitterverfeinerung enger zu wählen. Da als Vergleichsstelle der Punkt $K = 1, \dots, 1 \in \mathbb{R}^d, d \in \{4, 5\}$ gewählt wird, bleibt der Erwartungswert $\mu^{refine} = 1$ der Verfeinerung-Normalverteilung unverändert. Das verkleinerte Verfeinerungsgebiet wird durch eine geringere Standardabweichung von $\sigma^{refine} = 0.15$ realisiert. In Tests hat sich gezeigt, dass eine noch kleinere Standardabweichung zu schlechteren Ergebnissen führt. Dies hängt mit der abermals gewählten Lösungsgebietsgröße von $\Omega = [0, 7]^d, d \in \{4, 5\}$ und der Dünngitterstruktur auf den doch recht niedrigen Startleveln zusammen: Wird die Verfeinerungs-Standardabweichung hier zu klein gewählt, so sind nicht genügend Gitterpunkte im Betrachtungsbereich der Verfeinerung-Normalverteilung vorhanden, um eine adäquate Verfeinerung des Gitters zu realisieren. Unter Umständen ist es nötig, für die höher-dimensionalen Fälle von den getroffenen Annahmen abzuweichen und z.B. eine initiale Gitterverfeinerung mit einer veränderlichen Verfeinerung-Normalverteilung zu implementieren. Da dies allerdings sehr ausführliche und langwierige Tests zur Folge hätte und somit den Rahmen dieser Arbeit sprengen würde, wird eine solche Implementierung an dieser Stelle nicht betrachtet.

Für die vier-Asset Option muss im Fall der normalen Adaptivität der Start- und max. Level auf 6 gesenkt werden, damit die entsprechende Black-Scholes Gleichung in noch angemessener Zeit gelöst werden kann. Wird hingegen die gewichtete Adaptivität verwendet, so können wieder höhere Start- und max. Level verwendet werden (bis max. 7). Im Gegenzug wurde für beide Varianten die Anzahl der initialen Verfeinerungen auf 5 angehoben, um eine bessere Auffüllung dieser niedrigeren Level zu erreichen. Die erzielten Ergebnisse mit diesem Vorgehen sind in Tabelle 3.9 gezeigt. Hier zeigt sich für beide Arten der Adaptivität Konvergenz. Allerdings wachsen die Gitter mit Standard-Adaptivität rasant an. Dies hängt erneut mit der Beschaffenheit der Payoff-Funktion zusammen. Hier liegt ein 3-dimensionaler Knick vor, der auf dem gesamten Lösungsgebiet Ω verfeinert wird. Interessant ist auch die Entwicklung der gewichteten adaptiven Gitterstrukturen. So zeigen sich die Werte für Startlevel 6 als Ausreißer: Es wird nur eine geringe Anzahl zusätzlicher Punkte erstellt. Der Grund hierfür sind vermutlich bereits beschriebene Dünngittereffekte auf Level 6. Allgemein lässt sich an allen adaptiven Berechnungen für die vier Asset Optionen ablesen, dass alle Berechnungen im Bereich 10^{-7} ablaufen, da keine spürbare Gitterverkleinerung während des Lösens erfolgt.

Bei der fünf-Asset Option werden sofort die Auswirkungen von dieser Beobachtung deutlich: Das Lösen der Optionen auf Gittern mit Standard-Adaptivität würde mehrere Tage (bei paralleler Ausführung auf heutigen High-End Workstations mit zwei Hexa-Core CPUs) in Anspruch nehmen. Aus diesem Grund wird hier nur die gewichtete Adaptivität behandelt. Da selbst diese bei den bis jetzt verwendeten Einstellungen zu viel Rechenzeit benötigen würde, werden Anpassungen vorgenommen, die bereits bei der drei-Asset Option am Rande angeklungen sind. So wird hier das max. Level ein Level höher als das Startlevel gewählt und es werden 7 initiale Gitterverfeinerungen durchgeführt, um ein hinreichend fein-maschiges

S.-Level	gew. Adap.	MC_{diff}	$\#GP$	t in [s]
Call				
2	N	0.0973468	80	1.04
3	N	0.0336638	1970	81.5
4	N	0.0093738	14330	711.7
5	N	0.0024121	53873	4901
6	N	0.0023109	160712	22149
2	J	0.0361128	27	0.23
3	J	0.0615028	97	1.19
4	J	0.0053298	749	25.29
5	J	0.0017485	3262	181.7
6	J	0.0049143	8719	522.1
7	J	0.0003443	24538	1852
Put				
2	N	0.0274410	80	1.19
3	N	0.0065009	1970	98.17
4	N	0.0015213	14321	708
5	N	0.0018154	53819	4920
6	N	0.0019739	160587	22422
2	J	0.028147	27	0.25
3	J	0.0105549	97	1.25
4	J	0.0043438	749	30.2
5	J	0.0032023	3263	190.3
6	J	0.0052798	8720	546
7	J	0.0002715	24495	1813

Tabelle 3.9: 4d Optionen, log-transformierte Koordinaten: Ergebnisse im Vergleich zu einer Monte Carlo Simulation derselben Option. N: normale Adaptivität, J: gewichtete Adaptivität.

Gitter, bei doch recht niedrigem Startlevel, zu erhalten. Es ist das erwähnte Phänomen, dass falls die Gitter zu grob sind, nicht hinreichend viele Punkte bei einem fixen Schwellwert für die Verfeinerung erzeugt werden, feststellbar. So sind für beide Optionen die Abweichung zur Monte Carlo mit einem Startlevel kleiner 5 nicht zufriedenstellend klein. Tabelle 3.10 zeigt die berechneten und sehr guten Ergebnisse.

Aus den beiden Ergebnistabellen wird deutlich, dass die (gewichteten) adaptiven Dünneren Gitter in puncto Genauigkeit zu Monte Carlo Simulationen vergleichbare Werte liefern können und zusätzlich noch die bereits beschriebenen Vorteile aufweisen. Allerdings wird dies mit sehr hohen Lösungszeiten „erkauft“. Diese steigen aufgrund der Rekursion des benötigten Up/Down Schemas im höher-Dimensionalen extrem an. Da ein Einsparen von weiteren Gitterpunkten zu schlechteren Ergebnis-

S.-Level	MC_{diff}	$\#\overline{GP}$	t in [s]
Call			
2	0.0703552	91	4.80
3	0.0966638	291	23.91
4	0.0151622	700	74.75
5	0.0015479	39041	14693
6	0.0003137	89856	46090
Put			
2	0.0180346	91	5.89
3	0.0221329	291	25.52
4	0.0213167	700	80.16
5	0.0027057	39039	14540
6	0.0003485	89872	46870

Tabelle 3.10: 5d Optionen, log-transformierte Koordinaten: Ergebnisse im Vergleich zu einer Monte Carlo Simulation derselben Option.

sen führt, muss man, um die Lösungsqualität zu erhalten, das Problem von seiner algorithmischen Seite aus beleuchten. Hierzu werden bereits bei den hier gezeigten Ergebnissen Parallelisierungsansätze verwendet, die in Kapitel 4 ausführlich erläutert werden. Im letzten Unterabschnitt dieses Kapitels, Abschnitt 3.4, wird ein Ansatz zur Lösungszeitsenkung basierend auf dem verwendeten iterativen Löser (BiCGStab) vorgestellt. Dieses Verfahren fand ebenfalls bei der Bewertung von 4d und 5d Optionen Anwendung. So wurden im Fall der vier-Asset Optionen max. 160 Iterationen und max. 180 Iterationen im Falle der fünf-Asset Optionen pro Zeitschritt ausgeführt.

Analog zu den anderen betrachteten Baskets wird der aktuelle Abschnitt mit Konvergenzabbildungen für das vier Asset (Abbildung 3.18) und fünf Asset Basket (Abbildung 3.19) geschlossen. Hierbei werden in beiden Fällen die Lösungen der gewichteten adaptiven Gitter verwendet. Allerdings wird nun nicht der relative Fehler auf dem Testgebiet Ω_T sondern die Abweichung vom Wert der Monte Carlo Simulation als Fehlermaß auf der y -Achse aufgetragen. Beide Plots zeigen ein zu den vorherigen Abbildungen vergleichbares Bild und verdeutlichen nochmals im direkten Vergleich zu den Abbildungen 3.16 und 3.17, dass mit Hilfe von adaptiven Dünngitterstrukturen hoch-dimensionale PDEs mit einer guten Genauigkeit gelöst werden können. Eine Angabe der Q -Maße ist an dieser Stelle nicht mehr sinnvoll, da keine Vergleichsmessungen auf regulären Gittern existieren.

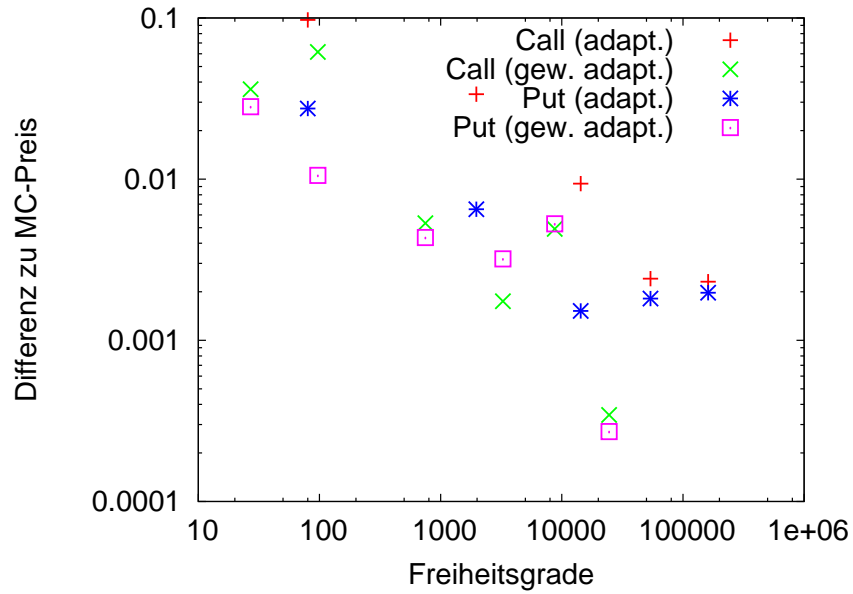


Abbildung 3.18: Konvergenzverhalten von 4d Call und Put Optionen auf gewichteten adaptiven Dünnen Gittern. Es sind deutliche Vorteile für die Gitter mit gewichteter Adaptivität erkennbar.

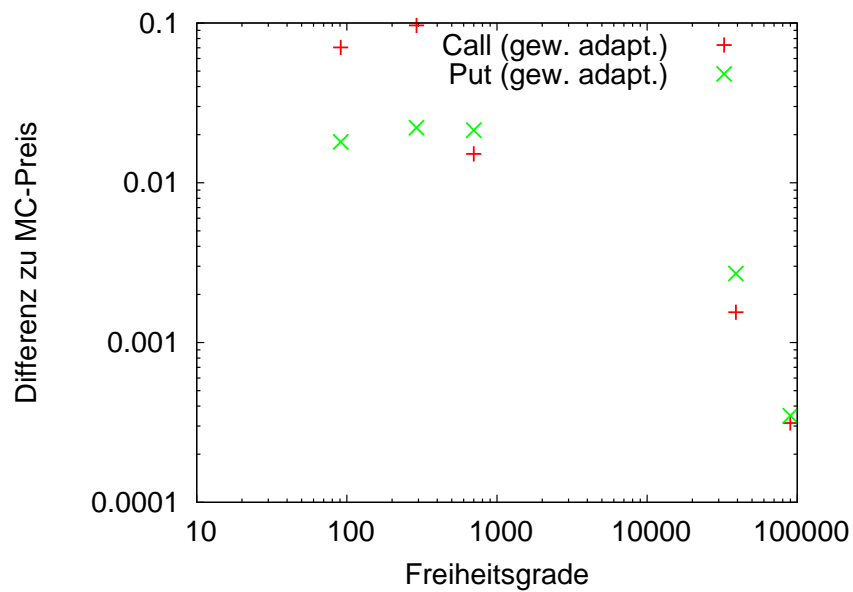


Abbildung 3.19: Konvergenzverhalten von 5d Call und Put Optionen auf gewichteten adaptiven Dünnen Gittern. Erst ab mehreren zehntausend Freiheitsgraden ergeben sich kleine Fehler.

3.4 Analyse des iterativen Lölers

Aus den Lösungstabellen der vier- und fünf-Asset Baskets ist ersichtlich geworden, dass sehr viel Zeit für die Lösung dieser höher-dimensionalen Optionen benötigt wird. Diese kann zum Einen, wie auch in Kapitel 4 beschrieben, durch eine Parallelisierung des Lölers verkürzt werden. Zum Anderen besteht eine weitere Möglichkeit, die sich aus dem verwendeten iterativen Löler, das BiCGStab-Verfahren ergibt. Bei dieser Methode handelt es sich um eine Abwandlung des klassischen CG Verfahrens, die das Lösen von unsymmetrischen linearen Gleichungssystemen unterstützt. Im Allgemeinen werden beim CG Verfahren in den ersten Iterationen die größten Fehler in der Startlösung geglättet, gefolgt von viel Feinarbeit. Es wird solange iteriert, bis Gleichung (3.3) erfüllt ist:

$$r \leq \|r_0\|^2 \cdot \epsilon^2, \quad (3.3)$$

hierbei bezeichnet r das Residuum der aktuellen Iteration und r_0 des Startresiduum und ϵ einen Schwellwertparameter des CG-Verfahrens. Ein weiteres Abbruchkriterium ist meistens eine maximale Anzahl von Iterationen i_{max} .

Da in jedem Zeitschritt beim Lösen der Black-Scholes Gleichung ein lineares Gleichungssystem gelöst werden muss, hat sich die Anpassung des ϵ 's für alle Lösungsvorgänge als nachteilhaft erwiesen. In späteren Zeitschritten ist die Lösung bereits sehr glatt und eine Multiplikation mit einem hohen ϵ^2 hat dann zu wenige Iterationen zur Folge. Diesen reichen nicht aus, um einen Zeitschritt mit hinreichender Genauigkeit zu lösen. Als bessere Steuerung hat sich die Anzahl der maximal durchzuführenden Iterationen erwiesen. Hier kann nach einer ausreichenden Anzahl einfach abgebrochen werden. Eine mathematische Rechtfertigung für ein solches Vorgehen zu finden ist sehr schwierig, da man den Lösungsvorgang beendet, bevor die Fehlerabschätzungen erfüllt sind. Es lassen sich allerdings aus der Anwendung und dem Design des Lölers qualitative Begründungen herleiten:

Parallelisierung: Bei der genauen Betrachtung der verwendeten Parallelisierung in Kapitel 4 wird sich zeigen, dass für eine fehlerfreie Verwendung Barrieren notwendig sind. Dadurch werden die Ergebnisse in-deterministisch, was die max. erreichbare Lösungsgenauigkeit beeinflusst.

Interpolationsfehler in der Startlösung: Wie in Abbildung 3.13 gezeigt, ist es schwer, den Interpolationsfehler der Startlösung am Knick unter 10^{-5} zu drücken.

Parameterschätzung der stoch. Prozesse: Die Parameter der stochastischen Prozesse zur Abbildung der Aktienkurse $(\mu_i, \sigma_i, \rho_{i,j})$ werden aus Finanzmarktdaten geschätzt. Hierbei sind Genauigkeiten nahe der Maschinengenauigkeit undenkbar.

Am Beispiel des drei Asset Call Baskets mit log-transformierten Koordinaten soll für reguläre und gewichtete adaptive Gitter gezeigt werden, dass dieser Ansatz einen Geschwindigkeitsvorteil bei quasi gleicher Genauigkeit liefert.

Level	max #It. pro δt	#It (ges.)	$\ e\ _\infty$	$\ e\ _{L_2}$
11	20	400	0.145717	0.0550243
11	40	800	0.0550849	0.0157473
11	60	1200	0.0236545	0.00608795
11	80	1600	0.013547	0.00264031
11	100	2000	0.0075917	0.00139808
11	120	2400	0.00489869	0.00122131
11	140	2784	0.0040214	0.00128224
11	160	3066	0.00368111	0.00133714
11	180	3205	0.00338309	0.00130551
11	200	3331	0.00331379	0.00130608
11	-	3994	0.00324796	0.00129236

S.-Level	max #It. pro δt	#It (ges.)	$\ e\ _\infty$	$\ e\ _{L_2}$
9	20	400	0.144379	0.0620653
9	40	800	0.0657135	0.0183765
9	60	1200	0.036158	0.00769492
9	80	1600	0.0178514	0.00334498
9	100	2000	0.00893818	0.0016015
9	120	2400	0.00437965	0.000920999
9	140	2800	0.00270033	0.000679258
9	160	3096	0.00223762	0.000662388
9	180	3216	0.00236089	0.000571309
9	200	3365	0.00242756	0.000551581
9	-	3966	0.00188739	0.000418312

Tabelle 3.11: 3d Call Option, log-transformierte Koordinaten: Analyse, wie die Anzahl der max. Iterationen die Normen der relativen Fehler beeinflusst.

Tabelle 3.11 zeigt die gemessenen Ergebnisse für den 3d Testfall und stellt fest, dass deutlich weniger Iterationen notwendig sind, um die Optionen mit nahezu gleicher Genauigkeit zu lösen. So reichen z.B. im adaptiven Fall knapp 1000 Iterationen weniger, um mit einem Fehler von 10^{-4} (welcher im Bereich des initialen Interpolationsfehlers liegt), im Vergleich zur aus-iterierten Variante, den Optionspreis zu bestimmen. Allerdings könnten noch bessere Ergebnisse durch eine Erweiterung des Lösers erreicht werden. So wäre, wie bereits weiter oben angesprochen, eine Zeitschritt-abhängige Steuerung der CG-Parameter eine lohnende Erweiterung, da so unter Umständen noch bessere Ergebnisse mit weniger Iterationen und somit in einer kürzeren Zeit erreicht werden könnten.

4 Parallelisierung des Black-Scholes PDE Löser

Wie bereits bei der numerischen Analyse angemerkt (und auch angewendet), wird parallele Algorithmik benötigt, um die Black-Scholes Gleichung auf (adaptiven) Dünnen Gittern in realistischer Zeit lösen zu können. Bereits in [11] wurde ein Verfahren vorgestellt, mit dem das hergeleitete Up/Down Schema auf Shared Memory Plattformen parallelisiert werden kann. Allerdings wurde dort auf einige Schwierigkeiten bei diesem Vorgehen hingewiesen, wie z.B., dass „nur“ 2^d parallele Tasks erstellt werden. Damit ist es schwierig, aktuelle Plattformen wie Intel Westmere-EP (12 Kerne) und Intel Nehalem/Westmere-EX (32/40 Kerne) für moderat dimensionale Optionen (3-4 Assets) voll auszureizen. Daher wird im nun folgenden Abschnitt eine Verbesserung dieser Parallelisierung dargestellt.

Ebenfalls wurden in [11] mehrere verschiedene x86 Architekturen untersucht. Hierbei stellte sich heraus, dass die besten Ergebnisse mit Intel's Nehalem Architektur erreicht werden können. Aus diesem Grund wird deren Nachfolger, die Westmere-Architektur, für die Messungen in dieser Arbeit verwendet. Ein Block-Diagramm des verwendeten Systems zeigt Abbildung 4.1. Als CPUs kommen zwei Intel Xeon X5650 Prozessoren mit 6 Kernen, 2.66 GHz Taktfrequenz und 12 MB LLC (Last Level Cache) zum Einsatz. Der Hauptspeicher besteht aus 6GB DDR3 mit 1333MHz und drei Speicherkanälen pro Sockel. Insgesamt verfügt das System also über 12 GB Arbeitsspeicher organisiert als NUMA Plattform. Das System verfügt zudem über mehrere GPUs und besitzt daher zwei Chipsätze, um genügend PCIe Leitungen zur Ansteuerung der Grafikkarten bereitzustellen. Allerdings können die vorhandenen GPUs nicht zur Ausführung der Dünngitteralgorithmik verwendet werden, da diese die komplexen Rekursionen nicht unterstützen. Ein AMD Prozessor wird nicht betrachtet, da die AMD K10 Architektur ebenfalls enorme Schwierigkeiten mit den Rekursionen hat. Nach [11] ist die Assoziativität der L1 Caches zu gering, was zu schlechter Cacheausnutzung führt. Es wird zunächst die Implementierung der Parallelisierung vorgestellt und anschließend ein Vergleich mit der Variante aus [11] durchgeführt.

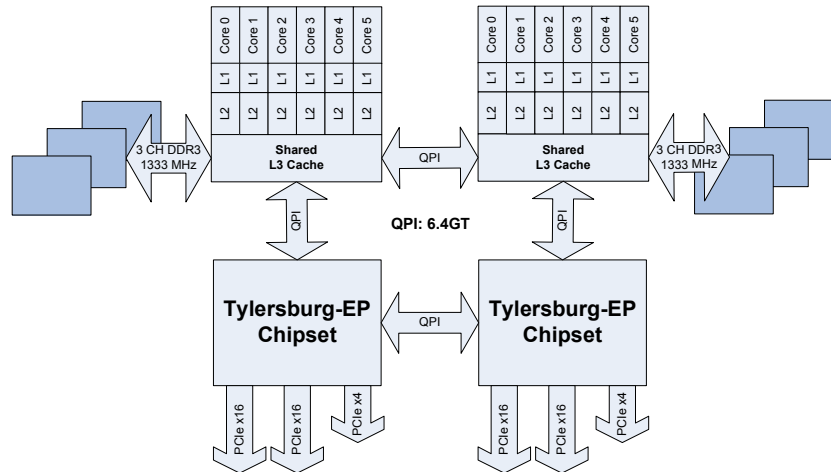


Abbildung 4.1: Intel Tylersburg-EP Plattform mit zwei 6-Kern Intel Westmere Xeon Prozessoren. Die Testplattform in dieser Masterarbeit.

4.1 Paralleles Up/Down Schema

Algorithmus 4.1 zeigt die bereits entwickelte Parallelisierung für das Up/Down Schema. Diese nutzt die Duplikation der Daten aus Abbildung 2.6 aus. Der obere und der untere Ast dieser Abbildung können unabhängig und somit parallel ausgeführt werden. Technisch wird die Parallelisierung mittels OpenMP realisiert. Hierbei werden jedoch nicht die klassischen Direktiven zur Schleifenparallelisierung eingesetzt, sondern das mit OpenMP 3 eingeführte *task* Konstrukt. Mittels diesem lassen sich schwierig parallelisierbare und rekursive Strukturen sehr gut und direkt parallelisieren. Um numerisch korrekte Ergebnisse zu erhalten, müssen die Berechnungen auf jeder Stufe der Rekursion synchronisiert werden. Dies ist nötig, damit die Zwischenergebnisse richtig durch die Dünngitterstruktur während der Berechnung des Up/Downs transportiert werden. Die dazu nötigen Barrieren (`#pragma omp taskwait`) verhindern eine gute Skalierbarkeit. Einzelne Tests, auf die an dieser Stelle nicht weiter eingegangen werden soll, haben gezeigt, dass für kleinere Anzahlen von Dimensionen max. 6 Threads für die parallele Ausführung des Up/Down Algorithmus sinnvoll sind, da sonst zu viel Rechenzeit in den Synchronisationsroutinen verbraucht wird.

Da der OpenMP 3 Standard bereits 2008 verabschiedet wurde, ist er heute in allen wichtigen Compilern zu finden (Intel Compiler für x86, IBM Compiler für PPC und auch im GNU Compiler). Für die Ergebnisse in dieser Arbeit wurde die Applikation

mit dem Intel Compiler in Version 12 (Intel C/C++ Composer XE 2011) übersetzt, um möglichst viele Optimierungen für die Zielplattform anzuwenden.

Algorithmus 4.1 Parallelisierung des Up/Down Schemas, vorgestellt in [11].

```

/**
 * alpha: Koeffizienten der Basisfunktionen, die hierarchischen Überschüsse
 * result: Koeffizienten der Basisfunktionen, Ergebnis des Operators, Rückgabewert
 * dim: Dimension, in der aktuell das Up/Down berechnet wird
 */
void updown_parallel(Vector alpha, Vector result, int dim)
{
    if(dim > 0) // Dimensionen übrig -> rekursive Aufrufe
    {
        int size = alpha.getSize();
        Vector temp(size), result_temp(size), temp_two(size);

        #pragma omp task shared(alpha, result)
        {
            up(alpha, temp, dim);
            updown_parallel(temp, result, dim-1);
        }
        #pragma omp task shared(alpha, result_temp)
        {
            updown_parallel(alpha, temp_two, dim-1);
            down(temp_two, result_temp, dim);
        }
        #pragma omp taskwait

        result.add(result_temp);
    }
    else // Rekursionsende
    {
        Vector temp(alpha.getSize());

        #pragma omp task shared(alpha, result)
        up(alpha, result, dim);

        #pragma omp task shared(alpha, temp)
        down(alpha, temp, dim);

        #pragma omp taskwait

        result.add(temp);
    }
}

```

Wie weiter vorne in der mathematischen Herleitung erläutert worden ist, ist das in Algorithmus 4.1 dargestellte Up/Down Schema die Kernroutine des Black-Scholes Lösers. Allerdings steckt in der zu lösenden Gleichung (nochmals angegeben in Gleichung (4.1)) erheblich mehr Parallelität:

$$A\dot{\vec{u}}(\tau) = \left[\sum_{i=1}^d \nu_i \cdot F - \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \sigma_i \sigma_j \rho_{i,j} \cdot E + r \cdot A \right] \cdot \vec{u}(\tau). \quad (4.1)$$

Die Matrizen A, E, F werden, wie hergeleitet, nicht aufgestellt, sondern deren Anwendung mittels Funktionale durch das Up/Down Schema berechnet. Bisher wurde ausschließlich dieser Teil parallelisiert. In der Summe über die Anwendung von F und in der Doppelsumme über die Anwendung von E liegt sehr viel und sehr

einfach zu synchronisierende Parallelität. Hierbei erhält jeder Prozessor einen privaten Ergebnisvektor welcher nach der Berechnung aller Up/Down Schemas mit allen anderen privaten Ergebnisvektoren zu einem Endergebnis aggregiert wird.

Da die *tasks* in OpenMP beliebig geschachtelt werden können, ist eine Erweiterung von Algorithmus 4.1 „nach Außen“ aus dem Blickwinkel der Parallelisierung einfach möglich. Allerdings verstößt ein solcher Ansatz gegen das verwendete Klassenkonzept in SG⁺⁺. Hier wird eine Operatoren-Idee im mathematischen Sinne verfolgt. Daher werden die Summe und die Doppelsummen gekapselt und es werden lediglich drei Aufrufe einer abstrakten *mult*-Routine benötigt, um die Anwendung von A, E, F inkl. ihrer Koeffizienten zu realisieren. Eine Parallelisierung der einzelnen Up/Downs macht allerdings nur Sinn, wenn alle Summanden, auch über Operatorenengrenzen hinweg, parallel ausgeführt werden können, da es sonst zu Lastbalancierungsproblemen kommt. Um diese zu meistern, sind die Klassen der Operatoren (Matrix E : *OperationGamma** und Matrix F : *OperationDelta**) um Memberfunktionen erweitert worden, die durch Angabe von i bzw. i, j nur einzelne Summanden berechnen. Für Matrix A ist eine solche Erweiterung nicht nötig, da die Ausführung des kompletten Operators nur ein Up/Down Schema umfasst und sie somit einfach in die Lastbalancierung für E und F eingebaut werden kann. Damit wird jeder Summand der rechten Seite von Gleichung (4.1) zu einem OpenMP-Task, der seinerseits wieder ein mittels OpenMP-Tasks paralleliertes Up/Down Schema aufruft. Dadurch ergeben sich bereits im Falle der zwei Asset Optionen genügend parallele Aufgaben, um auch Systeme mit mehr als vier Prozessorkernen auszulasten. Im höher-dimensionalen Fall steigt die Anzahl der Jobs stark an, so dass auch große Systeme verwendet werden können. Allerdings wurden hierbei massive Skalierungsprobleme auf einem SGI Altix UV (UltraViolet) System mit Nehalem-EX Prozessoren und DSM (Distributed Shared Memory) festgestellt. Es war nicht möglich, einen zusätzlichen Speed-Up zu erreichen, falls Ausführungen auf mehr als einem Blade erfolgten und somit NumaLink Kommunikation unvermeidbar war. Der Grund für dieses Verhalten ist ebenfalls in Algorithmus 4.1 auffindbar. Es handelt sich hierbei um Zugriffe auf die Gitterstruktur (nicht dargestellt im Pseudo-Code) und den Zugriff auf den Koeffizientenvektor *alpha*. Diese sind, im Gegensatz zu allen anderen Daten des Algorithmus, nicht privat und dupliziert für jeden Thread. Somit wird der Speicher, in dem das Gitter und seine Koeffizienten liegen, zum Flaschenhals, da dieser eine solche Flut von Zugriffen nicht mehr abarbeiten kann. Besonders vor dem Hintergrund, dass der Algorithmus rekursiv auf adaptiven Strukturen arbeitet und somit stark Speicher-gebunden ist.

Um diesem Dilemma zu entkommen, existieren zwei Lösungen, welche nun kurz vorgestellt werden. Die erste Lösung würde an der jetzigen Parallelisierung festhalten, jedoch zusätzlich die Gitterdaten und Koeffizienten entsprechend häufig duplizieren, so dass jeder Summand mit komplett privaten Daten berechnet werden könnte. Das würde zahlreiche Anpassungen in verschiedenen Klassen erfordern, allerdings könnte die Parallelisierung mittels OpenMP unverändert weiterverwen-

det werden. Ein solches Vorgehen hat einen großen Nachteil: Es ermöglicht nur die Ausführung auf Systemen mit DSM, wie der Altix UV. Die meisten heutigen Cluster sind aber Maschinen mit verteilten Speichern (meist aus Kostengründen, da „Standard“-Komponenten verwendet werden können). So muss sich der Programmierer selbst um die Verteilung der Daten auf die einzelnen Knoten kümmern, z.B. mittels dem Message-Passing-Interface (MPI). Da dies mit fast identischer Performance auf Systemen mit DSM eingesetzt werden kann, ist es das Verfahren der Wahl. Aus zwei Gründen ist eine Parallelisierung mit MPI nicht in dieser Masterarbeit implementiert worden. Zum Einen war das erklärte Ziel die bessere Ausnutzung von Standard-Workstation Systemen und nicht die Portierung auf Supercomputer. Zum Anderen verlangt eine Integration von MPI in die bestehende Codebasis gewaltige Anpassungen und würde den Rahmen dieser Arbeit sprengen, weswegen an dieser Stelle nur die Idee dargestellt worden ist. Daher wird im Rest des Kapitels auf die beiden implementierten Parallelisierungen für Workstation Systeme mit gemeinsamem Speicher eingegangen.

4.2 Messung der Speed-Ups

Die beiden Parallelisierungen werden nun für die berechneten Optionen auf Gittern mit log-transformierten Koordinaten aus dem letzten Kapitel (Werte von μ, σ, ρ und Strike) bewertet. Da sich keine nennenswerten Unterschiede für Call und Put Optionen herausgestellt haben, werden für die Berechnung der Speed-Ups nur Put Optionen betrachtet. Die so gewonnenen Ergebnisse lassen sich ohne Probleme auf die Fälle der Call Optionen übertragen. Die Auswertung der Speed-Ups erfolgt, falls möglich, auf den regulären Dünnen Gittern und für alle Anzahlen von Underlyings auf gewichteten adaptiven Gittern. Für die adaptiven Gitterstrukturen ist die Betrachtung der Speed-Ups besonders interessant, da diese Gitter zu unbalancierten Rekursionen führen und somit die OpenMP-Tasks unterschiedliche Laufzeiten aufweisen. Hier ist Qualität des Schedulers in der OpenMP Laufzeitumgebung entscheidend für eine effiziente Ausführung. Als Test-Optionen dienen:

- 2 Asset Put auf reg. Gittern mit 13 Leveln
- 2 Asset Put auf gewichtet adap. Gittern (Adap. Einstellungen wie oben)
- 3 Asset Put auf reg. Gittern mit 12 Leveln
- 3 Asset Put auf gewichtet adap. Gittern (Adap. Einstellungen wie oben)
- 4 Asset Put auf gewichtet adap. Gittern (Adap. Einstellungen wie oben)
- 5 Asset Put auf gewichtet adap. Gittern (Adap. Einstellungen wie oben)

Bevor die gemessenen Speed-Ups vorgestellt werden können, müssen noch ein paar Bemerkungen zur technischen Ausführung gemacht werden. Die verwendeten Xeon Prozessoren können pro Kern zwei Threads parallel in Hardware ausführen. Dies

ist unter bestimmten Bedingungen sehr hilfreich (auch bei der Verwendung von adaptiven Dünnen Gittern). Tests mit diesem so genannten Hyperthreading haben sich aber als nachteilig herausgestellt, falls die Parallelisierung über die Summanden verwendet wird. Als Grund für dieses Verhalten wurden die verschiedenen Anzahlen von Up/Down Schemata bei den verschiedenen Operatoren ausgemacht. Dadurch rechnet der gleiche Kern bei verwendetem Hyperthreading unter Umständen an zwei verschiedenen Operatoren, was zu einer schlechteren Ausnutzung des Prozessor-Caches führt. Aus diesem Grund werden die Threads in den hier gezeigten Tests so an die Hardwarethreads der beiden Prozessoren gepinnt, dass immer nur ein Thread pro Kern aktiv ist. Insgesamt werden also bis zu 12 Threads verwendet. Da Intel's Westmere Prozessoren über Energiesparfunktionalität verfügen, die die Taktfrequenz verändert, wurde diese für die durchgeführten Tests abgeschaltet. Gleiches gilt für den mit Westmere (wieder) eingeführten Turbomode der Prozessoren. Durch diesen besitzen die Prozessoren die Möglichkeit, die Taktfrequenz für ausgewählte Kerne anzuheben, falls nicht alle Kerne des Prozessors gleichzeitig aktiv sind. So kann der verwendete Xeon mit bis zu 3.06 GHz arbeiten, falls nur ein Kern genutzt wird. Solche Features sind im Alltagsbetrieb ausgesprochen nützlich, erlauben aber keine Berechnung von Speed-Ups.

Option	Art Parallel.	Anzahl Threads (Kerne)					
		2	4	6	8	10	12
2d reg.	nur Up/Down	1.98	3.97	3.98	3.97	3.99	3.98
2d reg.	Summanden	2.00	3.99	5.78	7.11	8.91	9.78
2d adap.	nur Up/Down	1.96	3.95	3.96	3.95	3.97	3.96
2d adap.	Summanden	1.97	3.95	4.61	5.54	6.85	8.37
3d reg.	nur Up/Down	2.04	3.97	4.61	5.22	5.21	5.22
3d reg.	Summanden	2.13	4.14	5.45	8.14	10.19	11.21
3d adap.	nur Up/Down	2.02	3.33	4.02	4.11	4.09	4.10
3d adap.	Summanden	2.02	3.71	4.99	6.36	7.22	7.35
4d adap.	nur Up/Down	1.99	3.51	4.10	5.10	5.51	5.62
4d adap.	Summanden	2.01	3.77	5.41	7.51	9.31	11.06
5d adap.	nur Up/Down	2.12	4.05	5.32	6.64	8.03	8.30
5d adap.	Summanden	2.14	4.09	5.96	7.75	9.83	11.78

Tabelle 4.1: Erreichte Speed-Ups für beide vorgestellten Parallelisierungsvarianten für die Testoptionen mit den besten numerischen Ergebnissen für zwei bis fünf Underlyings.

Für alle angegebenen Optionen werden die Speed-Ups für die Verwendung von 2,4,6,8,10 und 12 Kernen auf der Tylersburg-EP Plattform in Tabelle 4.1 dargestellt. Zusätzlich verdeutlicht Abbildung 4.2 den Unterschied zwischen der ausschließlichen Parallelisierung des Up/Downs und der Parallelisierung über alle Summanden der rechten Seite. Im Falle der regulären Gitter sind Ergebnisse vergleichbar mit denen aus [11]. Für die Up/Down Parallelisierung sieht man gut

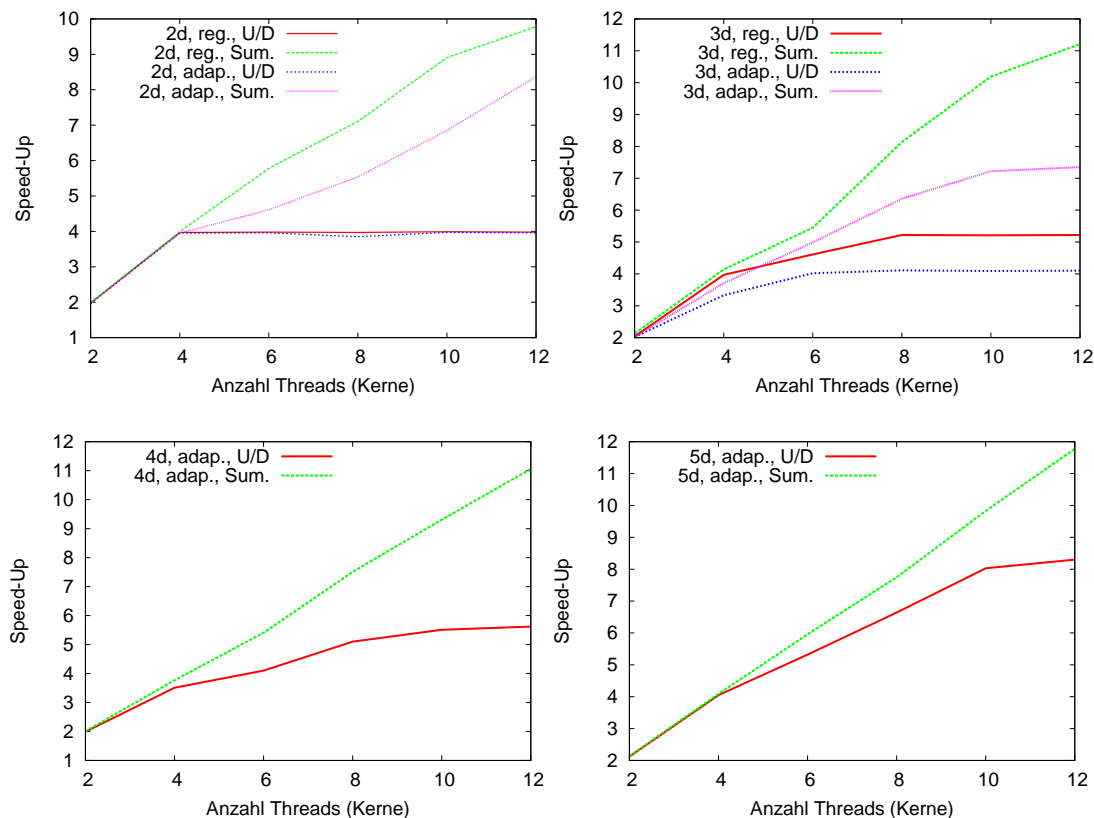


Abbildung 4.2: Vergleich der Speed-Ups für die Parallelisierung der Up/Down Schemata und der Parallelisierung über alle Summanden.

deren Limitierung auf 2^d Tasks. So ist für zwei und drei Asset Baskets die obere Grenze vier bzw. acht. Da allerdings Barrieren benötigt werden, ist es nur für die zwei Asset Option möglich, diese Grenze auch zu erreichen. Erst wenn für Optionen mit mehr als drei Underlyings deutlich mehr parallele Tasks zur Verfügung stehen, können höhere Speed-Ups erreicht werden. Jedoch sind diese weit vom erreichbaren Ideal entfernt. Noch schlechter sieht es mit der Up/Down Parallelisierung für adaptive Gitter aus. Wegen den Barrieren kann die Laufzeitumgebung von OpenMP keine Aufgaben auf freie Prozessoren legen, die wegen des unbalancierten Gitters ihre Berechnungen bereits abgeschlossen haben. Wird hingegen die in dieser Arbeit entwickelte Parallelisierung über alle Summanden verwendet, so können die eben beschriebenen Nachteile überwunden werden. Sowohl bei der Verwendung von regulären als auch von adaptiven Gittern können für zwölf Threads fast immer optimale Speed-Ups erreicht werden. So kann die verwendete Tylersburg Plattform bereits für drei Asset Baskets voll ausgenutzt werden.

Für beide Arten der Parallelisierungen sind super-lineare Speed-Ups messbar. Diese sind auf einen Cache-Effekt, das so genannte *True Sharing* zurückzuführen. Hiermit wird folgendes Szenario beschrieben: Mehrere Threads arbeiten auf denselben Daten auf einer CPU mit gemeinsamen Cache. Holt ein Thread Daten aus

dem Speicher in den gemeinsamen Cache, so kann ein anderer Thread kurz darauf dieselben Daten verwenden, ohne erneut auf den Speicher zugreifen zu müssen. Da die verwendete Up/Down Algorithmik Speicher-gebunden ist, kann so die Ausführungszeit zusätzlich verkürzt werden, was zu den festgestellten super-linearen Speed-Ups führt. Dies funktioniert allerdings nur dann, wenn genügend Platz im gemeinsamen Cache vorhanden ist. So ist für die Parallelisierung über die Summanden erkennbar, dass der super-lineare Speed-Up nur auftritt, falls vier oder zwei Kerne pro CPU aktiv sind. Bei Vollauslastung der CPU wird der gemeinsame Cache zu klein und es kann „nur“ der optimale Speed-Up erzielt werden.

Insgesamt lässt sich zu der Auswertung der parallelen Implementierung festhalten, dass mit der erweiterten Parallelisierung deutlich bessere Speed-Ups erreicht werden können und damit ein Proof-of-Concept für die Parallelisierung mit MPI erbracht werden konnte. Somit sollte es ohne größere Probleme möglich sein, mittels MPI mehr als 12 Kerne bei der Bewertung von Basket-Optionen effizient auszunutzen.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein direkter Löser für die multi-Asset Black-Scholes PDE für die Verwendung von adaptiven Dünngitterstrukturen erweitert. Hierzu wurden zwei Formulierungen der Black-Scholes PDE betrachtet: Eine mit klassischen kartesischen Koordinaten und eine mit logarithmisch-transformierten Koordinaten, die zu einer besseren Kondition des zu lösenden Systems führt. Im Vergleich beider Varianten ergab sich ein klarer Vorteil für die Variante mit log-transformierten Koordinaten, da mit ihnen die Black-Scholes Gleichung in deutlich kürzerer Zeit und mit einer besseren Genauigkeit (kleinerem relativen Fehler) gelöst werden konnte.

Der Schwerpunkt der Arbeit lag auf der Verwendung von adaptiven Dünngitterstrukturen. Mit Hilfe dieses Werkzeugs gelang die Lösung von Basket-Optionen mit bis zu fünf Underlyings mit sehr guten Genauigkeiten. Adaptive Gitter sind bei der Black-Scholes Gleichung aus zweierlei Gründen nützlich. Zum Einen weist die Startlösung (Payoff-Funktion der Option) einen Knick auf, dessen Diskretisierung eine massive Herausforderung darstellt. Es werden an dieser Stelle deutlich höhere Gitterauflösungen als im Rest des Gebietes benötigt, was starke Nachteile für reguläre Gitterstrukturen bedeutet. Zum Anderen ist die Black-Scholes Gleichung eine Verwandte der Wärmeleitungsgleichung und besitzt somit einen Diffusionsanteil. Dadurch wird die Lösung mit fortschreitender Zeit immer glatter und es werden immer weniger Freiheitsgrade benötigt, um die Lösung hinreichend genau auf dem Gitter darzustellen. Die klassischen regulären Dünnen Gitter konnten aufgrund der enorm hohen Rechenzeit (es werden aus oben genannten Gründen sehr viele Gitterpunkte benötigt) mit vertretbarem Einsatz nur für Optionen mit bis zu drei Underlyings verwendet werden. Im Fall von fünf Underlyings reichten selbst normale adaptive Gitterstrukturen nicht aus, um die Optionen mit realistischem Aufwand zu lösen. Aus diesem Grund wurde eine spezielle lokale Adaptivität mittels normalverteilter Gewichten eingeführt.

Tabelle 5.1 zeigt eine Zusammenfassung der besten Ergebnisse für europäische Call- und Put Optionen auf Gittern mit log-transformierten Koordinaten, um die Vorteile der adaptiven Dünngitter zu verdeutlichen: Mit einem Zehntel der Freiheitsgrade (im Vergleich zu regulären Dünnen Gittern) können die Optionen bewertet werden, und da ein rekursiver Algorithmus, das Up/Down Schema, zur Berechnung der benötigten Matrix Vektor Produkte verwendet wird, sinkt die Rechenzeit um deutlich mehr als 90 %.

d	Typ	$\ e\ _\infty$	reg. DG	std. Adaptivität		gew. Adaptivität	
2	Call	0.0001	98305	14017	(14.3%)	11241	(11.4%)
2	Put	0.0002	98305	12507	(12.7%)	9811	(10%)
3	Call	0.003/0.002/0.002	274431	131173	(47.8%)	41588	(15.2%)
3	Put	0.007/0.0008/0.004	274431	128059	(46.7%)	40081	(14.6%)
4	Call	$0.002^{\text{MC}}/0.0003^{\text{MC}}$	-	160712	(-)	24538	(-)
4	Put	$0.002^{\text{MC}}/0.0003^{\text{MC}}$	-	160587	(-)	24495	(-)
5	Call	0.0003^{MC}	-	-	(-)	89856	(-)
5	Put	0.0003^{MC}	-	-	(-)	89872	(-)

Tabelle 5.1: Erreichte Genauigkeiten auf Dünnen Gittern mit log-transformierten Koordinaten für reguläre, standard adaptive und gewichtete adaptive Gitterstrukturen und der Anzahl der jeweils benötigten Freiheitsgrade.

Mit dem Up/Down Schema wurde nun auch der größte algorithmische Flaschenhals der aktuellen Implementierung erwähnt. Es wurde zwar eine verbesserte Parallelisierung in dieser Arbeit entwickelt, allerdings skaliert diese noch nicht optimal und es ist ungewiss, wie sie sich auf in Kürze erscheinenden HPC Sharded-Memory Plattformen (z.B. Intel Westmere-EX (40 Cores/80 Threads), Sandy Bridge-EX) verhalten wird. Daher ist der wichtigste Aspekt bei der Weiterentwicklung des Lösers eine noch effizientere (parallele) Berechnung der benötigten Operatoren, um sich in puncto Ausführungszeit den Monte Carlo Simulationen zu nähern.

Zur Zeit existieren zwei viel versprechende Ansatzpunkte für eine effizientere Implementierung des Up/Downs. Die aktuelle Implementierung nutzt die NUMA Architektur der heutigen mehr-Sockel Systeme nicht aus und bietet keine Unterstützung für Cluster mit einer Distributed-Memory Umgebung. Diese beiden Fliegen können mit einer Klappe geschlagen werden, indem eine hybride MPI und OpenMP Implementierung entwickelt wird. So würde die vorgestellte zwei-stufige Parallelisierung nicht mehr ausschließlich über OpenMP erfolgen, sondern die Summanden der rechten Seite (L) würden mittels MPI parallelisiert und eine Up/Down Ausführung ist ein Paket für einen Sockel. So können auch Features wie SMT (hardware-basiertes Threading) besser genutzt werden, da sich die verschiedenen Threads nicht gegenseitig Daten in Caches überschreiben würden. Der zweite Ansatz besteht in der effizienten Aufstellung der kompletten Systemmatrix. Diese ist weder voll noch dünn besetzt und muss daher für die Verwendung mit Standardtools (z.B. Intel MKL) voll abgespeichert werden. Hier wird meist der verfügbare Hauptspeicher zum Flaschenhals. In [2] wurde ein Cache-effizientes Verfahren zum Speichern und Verwenden von hybriden Matrix-Strukturen vorgestellt, bei dem vollbesetzte Bereiche voll und dünnbesetzte Bereiche dünn abgespeichert werden und sehr gute Performancezahlen in Vergleich zu Standard-Dünnmatrixstrukturen erreicht werden konnten. Mittels einer geeigneten Sortierung der Basisfunktionen des Dünnes Gitters lässt sich eine optimale Struktur für diese hybriden Matrizen realisieren.

A Monte Carlo Simulation für Europäische Basket-Optionen

Wie bereits im Hauptteil dieser Arbeit angeklungen ist, werden in den meisten Fällen die Preise von Basket Optionen mit Hilfe von Monte Carlo Simulationen bestimmt. Dies hat verschiedene Gründe. Der wichtigste ist aber wohl, dass eine Monte Carlo Simulation sehr einfach zu implementieren ist. Es werden nur eine Hand voll Zeilen Programmcode benötigt. In diesem Abschnitt werden die Idee der Monte Carlo Simulation erläutert sowie ein kurzes C++ Programm vorgestellt, das Preise von Europäischen Basket-Optionen bestimmt.

A.1 Idee der Monte Carlo Simulation

Die Monte Carlo Simulation zur Optionsbewertung kann nach [29, 33] folgendermaßen hergeleitet werden: Durch die Berechnung von hinreichend vielen Trajektorien des stochastischen Prozesses, welcher zur Modellierung des Aktienkurses gewählt worden ist. In dieser kurzen Einführung in die Monte Carlo Simulation bei der Optionsbewertung soll nicht weiter auf Beweise für dieses Verfahren eingegangen werden.

Die Monte Carlo Simulation wird durch eine hohe Anzahl von Auswertungen der geschlossenen Lösung der SDE (Stochastische Differentialgleichung) der Geometrischen Brownschen Bewegung (GBM), die stochastische Modellierung des Aktienkurses im Black-Scholes Modell, implementiert. Nach [33] besitzt die SDE der GBM folgende geschlossene Lösung:

$$S(t) = S(0) \cdot \exp [(\mu - 0.5\sigma^2)t + \sigma W(t)], \quad (\text{A.1})$$

wobei $W(t)$ eine Zufallsvariable ist, die einem Wiener Prozess folgt.

Wird allerdings nur die Lösung zum Zeitpunkt $t = T$ benötigt, so wie es bei europäischen Optionen der Fall ist, kann diese ebenfalls nach [33] durch eine direkte Formel ermittelt werden:

$$S(T) :=_v S(0) \cdot \exp \left[(\mu - 0.5\sigma^2)T + \sigma\sqrt{T}Z \right]. \quad (\text{A.2})$$

Hierbei gilt $Z \sim N(0, 1)$, und $:=_v$ bedeutet gleich in der Wahrscheinlichkeitsverteilung.

Damit ist nun der Kern der Monte Carlo Simulation für Europäische (Basket-)Optionen definiert. Nach jeder Auswertung der SDE folgt noch die Bestimmung des Wertes der Payoff-Funktion der Option mit dem Ergebnis der SDE. Dies muss jetzt für eine hinreichend große Anzahl von Pfaden der SDE erfolgen. Richtwerte sind hier, basierend auf empirischen Erfahrungen, $10^6 - 10^{10}$ Pfade, abhängig von der gewünschten Genauigkeit der Lösung. Alle Werte der Payoff-Funktionsauswertungen werden aufsummiert und am Ende durch die Anzahl der verwendeten Pfade geteilt. Dieses Ergebnis ist der Wert der betrachteten Option. Ggf. müssen nun noch die Zinsen berücksichtigt werden und der berechnete Wert der Option abgezinst werden. In Abbildung A.1 werden die einzelnen Schritte nochmals graphisch hervorgehoben. Die grüne Linie zeigt die Payoff-Funktion des betrachteten Puts, die Rote stellt den heutigen Wert der Option dar. Die dünnen blauen Linien sollen schematisch die GBM zeigen und die Auswertung der Payoff-Funktion ist durch die senkrechten türkisen Linien markiert.

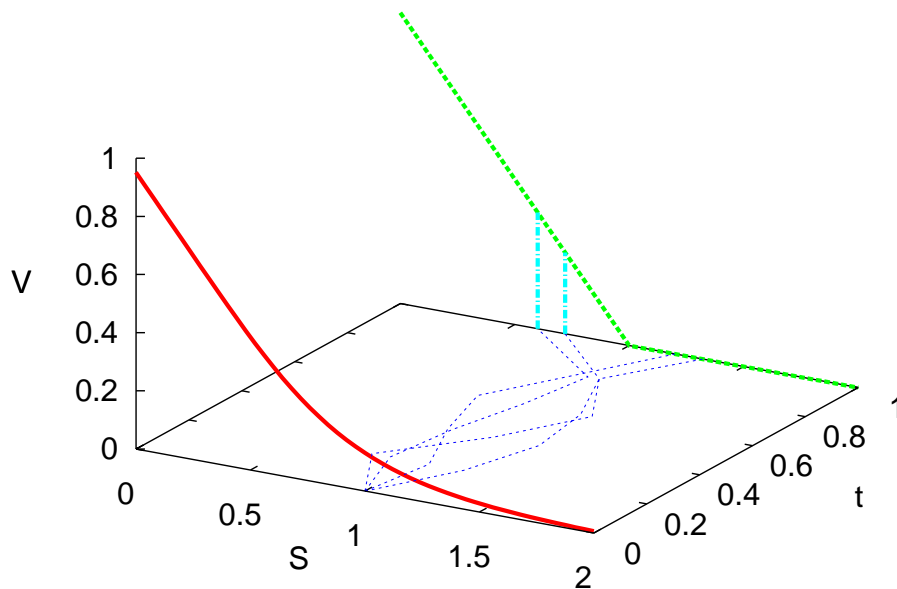


Abbildung A.1: Schematische Verdeutlichung der Funktionsweise der Monte Carlo Simulation beim Bewerten von Optionen. Hier eine Put Option auf ein Asset mit folgenden Parametern: $S(0) = 1.0, T = 1.0, K = 1.0, \sigma = 0.4, \mu = r = 0.05$. Gezeigt werden vier Monte Carlo Pfade. Eigene Darstellung nach [29].

Die bereits verbal gegebene Beschreibung soll anhand von Algorithmus A.1 nochmals in wenigen Zeilen Pseudo-Code hervorgehoben werden. Aus dieser knappen Darstellung wird klar, warum dieser Algorithmus so beliebt, aber auch sehr effizient ist: Es kann mit einfachen Mitteln implementiert werden. Außerdem ist er, was in der heutigen Zeit mehr als wichtig für gute Performance ist, sehr einfach parallelisierbar. Dies gilt sowohl in Hinblick auf Task-Parallelisierung als

auch für die deutlich schwieriger zu realisierende Daten-Parallelisierung (Vektorisierung). Diese ist für eine effiziente Ausführung auf Beschleunigern und/oder Co-Prozessoren wie GPUs sehr wichtig.

Algorithmus A.1 Pseudo-Code einer Monte Carlo Simulation einer europäischen Option mit einem Underlying.

```
1:  $V := 0.0$ 
2: for  $i = 1$  to  $N$  do
3:    $Z := \text{RANDOM}(N(0, 1))$ 
4:    $S = S(0) \cdot \exp \left[ (\mu - 0.5\sigma^2)T + \sigma\sqrt{T}Z \right]$ 
5:    $V = V + \text{Payoff}(S, K)$ 
6: end for
7:  $V = V/N$ 
8:  $V = V \cdot \exp[-(r \cdot T)]$ 
9: return  $V$ 
```

Bis jetzt wurden nur Optionen, die auf ein Underlying gehen, betrachtet, also keine Basket-Optionen. Will man mit Hilfe einer Monte Carlo Simulation den Preis einer Basket-Option bestimmen, so müssen an dem bereits vorgestellten Algorithmus nur kleinere Anpassungen durchgeführt werden. Da die Basket-Option mehrere Assets, die einer GBM folgen, als Underlyings hat, müssen jetzt in jedem Pfad der Monte Carlo Simulation mehrere SDEs gelöst werden. Dies erfolgt unabhängig von einander. Allerdings können die einzelnen Assets miteinander korreliert sein. Diese Korrelation wird über Z modelliert. Hier wird in jedem Pfad für jedes Asset eine unabhängige standard-normalverteilte Zufallszahl gezogen. Mittels eines geeigneten Verfahrens werden aus diesen unabhängigen Zufallszahlen korrelierte Zufallszahlen assembliert. Nach [29] ist hierfür die Cholesky Zerlegung der Korrelations-Matrix der Assets zu bestimmen und anschließend ergibt eine Multiplikation dieser zerlegten Matrix mit dem Vektor der unabhängigen standard-normalverteilten Zufallszahlen genau die Werte, die für Z bei der Berechnung der Lösung der SDE für jedes Asset benötigt werden.

A.2 MC C++ Code für Basket-Optionen

Die am Ende des letzten Abschnitts aufgezeigte Monte Carlo Simulation einer Europäischen Basket-Option soll nun in der Programmiersprache C++ vorgestellt werden. Es werden zwei Algorithmen benötigt, die sich vermutlich nicht im Standard-Repertoire eines Programmierers befinden: Die Generierung von standard-normalverteilten Zufallszahlen und die Cholesky Zerlegung von quadratischen, positiv definiten Matrizen. Allerdings existieren viele Mathematik Bibliotheken, welche entsprechende Implementierungen bereitstellen. Im Falle des Listings A.1 wurde die Intel Math Kernel Library (MKL) verwendet. Mit Hilfe der

Methode *dlarnv* kann ein Vektor mit standard-normalverteilten Zufallszahlen erzeugt werden und *dpotf2* implementiert die Cholesky Zerlegung, beide in doppelter Genauigkeit.

Nun soll der Code genauer erläutert werden. Gelöst wird eine Europäische Put Basket-Option auf zwei Underlyings mit Laufzeit von einem Jahr. Die ersten 30 Zeilen dienen lediglich der Definition benötigter Größen. Bevor die (parallele) Berechnung der einzelnen Pfade in Zeile 40 beginnt, wird die Kovarianzmatrix aufgestellt und deren Cholesky Zerlegung berechnet, um nach oben genannten Verfahren korrelierte Zufallszahlen zu erhalten. In Zeile 47 beginnt die Berechnung eines Pfades. Hierzu werden zunächst mit der MKL unabhängige Zufallszahlen generiert und diese anschließend mit der Cholesky-Zerlegten multipliziert. Danach erfolgt die Lösung der SDE der GBM für alle Underlyings der betrachteten Option. In Zeilen 63-69 erfolgt die Auswertung der Payoff-Funktion. In den noch fehlenden Zeilen wird, wie am Ende des betrachteten Pseudo-Codes, der Schätzer für den Erwartungswert berechnet und anschließend dieser noch mit r diskontiert.

Listing A.1: Implementierung einer Monte Carlo Simulation für Europäische Basket-Optionen mit Hilfe der Intel Math Kernel Library (MKL).

```

1 #include <iostream>
2 #include <cmath>
3 #include <cstdlib>
4 #include <algorithm>
5 #include "omp.h"
6 #include "mkl.h"
7
8 #define MCPATHS 10000000000
9 #define NUMASSESTS 2
10 #define PUT
11
12 double rhoMatrix[NUMASSESTS*NUMASSESTS]= {1.0, -0.5, -0.5, 1.0};
13 const double sigma[NUMASSESTS] = {0.3, 0.4};
14 const double mu[NUMASSESTS] = {0.05, 0.05};
15 const double s_0[NUMASSESTS] = {1.0, 1.0};
16 const double K = 1.0;
17 const double r = 0.05;
18 const double t = 1.0;
19 // dlarnv parameters
20 const int izeed[4] = {0, 0, 0, 1};
21 const int disttype = 3;
22 const int numRNDs = NUMASSESTS;
23 // dpotf2 parameters
24 const char uplo = 'U';
25 const int numAssests = NUMASSESTS;
26 const int info = 0;
27
28 int main(int argc, char* argv[])
29 {
30     double price = 0.0;
31
32     for (size_t i = 0; i < NUMASSESTS; i++) {
33         for (size_t j = 0; j < NUMASSESTS; j++) {
34             rhoMatrix[(i*NUMASSESTS)+j] *= (sigma[i]*sigma[j]);
35         }
36     }
37
38     dpotf2(&uplo, &numAssests, (double*)&rhoMatrix, &numAssests, &info);
39
40     #pragma omp parallel shared(price)
41     {

```

```

42 | double myPrice = 0.0;
43 | double rndNum[NUMASSESTS];
44 | double coRndNum[NUMASSESTS];
45 |
46 | #pragma omp for schedule(static)
47 | for (size_t i = 0; i < MCPATHS; i++) {
48 |     dlarnv(&disttype, (int*)&iseed, &numRNDs, (double*)&rndNum);
49 |
50 |     for (size_t n = 0; n < NUMASSESTS; n++) {
51 |         coRndNum[n] = 0.0;
52 |         for (size_t m = 0; m <= n; m++) {
53 |             coRndNum[n] += rhoMatrix[(n*NUMASSESTS)+m]*rndNum[m];
54 |         }
55 |     }
56 |
57 |     double sim = 0.0;
58 |     for (size_t j = 0; j < NUMASSESTS; j++) {
59 |         sim += s_0[j] * exp(((mu[j] - (0.5*sigma[j]*sigma[j]))*t) + (sqrt(t) *
60 |             coRndNum[j]));
61 |     }
62 |     sim /= static_cast<double>(NUMASSESTS);
63 | #ifdef CALL
64 |     myPrice += std::max<double>(sim - K, 0.0);
65 | #endif
66 | #ifdef PUT
67 |     myPrice += std::max<double>(K - sim, 0.0);
68 | #endif
69 | }
70 |
71 | #pragma omp critical
72 | {
73 |     price += myPrice;
74 | }
75 | }
76 |
77 | price /= static_cast<double>(MCPATHS);
78 | price *= exp((-1.0)*r*t);
79 | std::cout << std::endl << "The option's price is: " << price << std::endl;
80 | return 0;
81 | }

```


Literaturverzeichnis

- [1] Achatz, Stefan: Adaptive finite Dünngitter-Elemente höherer Ordnung für elliptische partielle Differentialgleichungen mit variablen Koeffizienten, Dissertation in Informatik, Technische Universität München, 2003.
- [2] Bader, M.; Heinecke, A.: Cache Oblivious Dense and Sparse Matrix Multiplication Based on Peano Curves, Proceedings of the PARA 08, 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing: Lecture Notes in Computer Science, 2009.
- [3] Bader, Michael; Zimmer, Stefan: Vorlesungsskript zur Vorlesung Algorithmen des wissenschaftlichen Rechnens I im Sommersemester 2008 an der Technischen Universität München.
- [4] Balder, Robert: Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern; Dissertation, Institut für Informatik, Technische Universität München, 1994.
- [5] Barret, R. et. al.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods <http://www.siam.org/books>
- [6] Benk, J.; Bungartz H.-J.; Nagy, A.-E. and Schraufstetter, S.: An Option Pricing Framework Based on Theta-Calculus and Sparse Grids, In Progress in Industrial Mathematics at ECMI 2010, Oktober 2010. submitted.
- [7] Black, Fisher; Scholes, Myron: The Pricing of Options and Corporate Liabilities. The Journal of Political Economy, Vol. 81, No. 3. (May - Jun., 1973), pp. 637-654, 1973.
- [8] Braess, Dietrich: Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie, 3. korrigierte und erweiterte Auflagen, Springer Verlag Berlin Heidelberg, 2009.
- [9] Bungartz, Hans-Joachim: Finite Elements of Higher Order on Sparse Grids, Shaker Verlag Aachen, 1998.
- [10] Bungartz, Hans-Joachim; Griebel, Michael: Sparse Grids; Acta Numerica (2004), Seiten 1-123, Cambridge University Press, DOI:10.1017/S0962492904000182.
- [11] Bungartz, Hans-Joachim; Heinecke, Alexander; Pflüger, Dirk; Schraufstetter, Stefanie: Parallelizing a Black-Scholes Solver based on Finite Elements and Sparse Grids; In IEEE International Parallel & Distributed Processing Symposium, 2010. accepted for publication

- [12] Dornseiger, Thomas: Diskretisierung allgemeiner elliptischer Differentialgleichungen in krummlinigen Koordinatensystemen auf dünnen Gittern, Dissertation in Informatik, Technische Universität München, 1997.
- [13] Feuersänger, Christian: Dünngitterverfahren für hochdimensionale elliptische partielle Differentialgleichungen, Diplomarbeit in Informatik, Rheinische Friedrich-Wilhelms-Universität Bonn, 2005.
- [14] Garcke, Jochen: Sparse Grid Tutorial; Centre for Mathematics and its Applications, Australian National University, Canberra, Australia, August 12, 2006.
- [15] Heinecke, A: Cache Optimised Data Structures and Algorithms for Sparse Matrices, Bachelorarbeit in Informatik, Technische Universität München, 2008.
- [16] Heinecke, A.: Realisierung einer Randbehandlung für adaptive Dünngittermethoden mit Anwendungen im Data Mining und in der Finanzmathematik, Masterarbeit in Informatik, Technische Universität München, 2010.
- [17] Heinecke, A., Bader M.: Parallel matrix multiplication based on space-filling curves on shared memory multicore platforms, Proceedings of the 2008 ACM Research Computing Frontiers Conference and co-located workshops: MAW'08 and WRFT'08, p. 385-392, 2008.
- [18] Heinecke, A., Bader M.: Towards many-core implementation of LU decomposition using Peano Curves, Proceedings of the 2009 ACM Research Computing Frontiers Conference and co-located workshops: Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop, p. 21-30, 2009.
- [19] Hull, J. C.: Options, Futures & Other Derivatives. Prentice-Hall International, 4. Auflage, 2000.
- [20] Johansson, Lennart: Einsatz der Dünngitterklassifikation für Benchmark-Probleme, Diplomarbeit in Informatik, Technische Universität München, 2008.
- [21] Leentvaar, C.C.W; Oosterlee, C.W.: On coordinate transformation and grid stretching for sparse grid pricing of basket options, Journal of Computational and Applied Mathematics, Volume 222 Issue 1, December, 2008.
- [22] Mertens, Thomas: Optionspreisbewertung mit dünnen Gittern, Diplomarbeit in Informatik, Rheinische Friedrich-Wilhelms-Universität Bonn, 2005.
- [23] Reisinger, Christoph: Numerische Methoden für hochdimensionale parabolische Gleichungen am Beispiel von Optionspreisaufgaben, Dissertation, Ruprecht-Karls-Universität Heidelberg, 2004.
- [24] OpenMP.org, The OpenMP API specification for parallel programming: OpenMP 3 Specification, <http://www.openmp.org/mp-documents/spec30.pdf>, Online, abgerufen am 23.12.2009, 2009.
- [25] Pflüger, Dirk: Data Mining mit dünnen Gittern, Diplomarbeit in Informatik, Nummer 2264, Universität Stuttgart, 2005.

- [26] Pflüger, Dirk: Spatially Adaptive Sparse Grids for High-Dimensional Problems, Dissertation in Informatik, Technische Universität München, Verlag Dr. Hut, München, February 2010
- [27] Schraufstetter, S. and Benk, J.: A General Pricing Technique Based on Theta-Calculus and Sparse Grids, In Proceedings of the ENUMATH 2009 Conference, Dezember 2009.
- [28] Schwegler, Kristina: Die Black-Scholes-Gleichung: Analysis und Numerik: Seminar Mathematische Modellbildung und Numerische Simulation, Universität Erlangen-Nürnberg, Wintersemester 2007/2008.
- [29] Seydel, Rüdiger U.: Tools for Computational Finance (Fourth Edition), Springer Verlag Berlin Heidelberg, 2009.
- [30] Wilmott, Paul; Howison, Sam; Dewynne, Jeff: The Mathematics of Financial Derivatives, Press Syndicate of the University of Cambridge, 1995.
- [31] Witten, Ian H.; Frank, Eibe: Data Mining: Practical Machine Learning Tools and Techniques, Second Edition, Elsevier, San Francisco, 2005
- [32] Zagst, Rudi: Vorlesungsskript zur Vorlesung Discrete Time Finance im Elitestudiengang Finanz- und Informationsmanagement an der Universität Augsburg, 2009.
- [33] Zagst, Rudi: Vorlesungsskript zur Vorlesung Continuous Time Finance im Elitestudiengang Finanz- und Informationsmanagement an der Universität Augsburg, 2009.
- [34] Zhang, Peter G.: Exotic Options, World Scientific Publication, Second Edition, 1998.