TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMARBEIT

Über den Arbenz-Embrechts-Puccetti-Algorithmus und eine adaptive Variante zur Anwendung im Operational Risk

Ausgeführt am Institut für
Wirtschaftsmathematik
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Friedrich Hubalek

durch
Christina Kyriakopoulos, B.Sc.
Kolonitzgasse 10/11
1030 Wien

Datum

Unterschrift

Danksagung

An erster Stelle möchte ich mich bei meinem Betreuer Herrn Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Friedrich Hubalek bedanken für die Bereitstellung des interessanten Themas und die aufbauenden und richtungsweisenden Ratschläge und Denkansätze, ohne die diese Arbeit nicht hätte entstehen können.

Besonderer Dank gilt meiner Mutter Dr. Elfrieda Kyriakopoulos für den starken emotionalen wie finanziellen Rückhalt. Darüber hinaus möchte ich meinen Brüdern Fragiskos und Anastasios Kyriakopoulos danken, die mich zu diesem Studium inspirierten und mich immer anregten mehr anzustreben.

Speziell möchte ich mich bei meinem Freund, Markus Draskovits bedanken, der mir während der Zeit des Schreibens mit Liebe und Geduld zur Seite stand und mir bei Zweifeln und in schwierigen Phasen weiterhalf. Meinen Freunden und Studienkollegen danke ich für die aufregende Zeit während meines Studiums.

Zuletzt noch herzlichen Dank an meinen Vorgesetzten und meine Arbeitskollegen bei der Wiener Städtische Versicherung für die große Flexibilität und Hilfe, die sie mir während meines Studienabschlusses entgegen brachten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Berechnung von $\mathbb{P}[X_1 + \ldots + X_d \leq s]$ ist von großer Bedeutung bei der Bestimmung des Eigenkapitalerfordernisses im operationellen Risikomanagement, da die Bestimmung eines Value-at-Risk basierten Eigenkapitalerfordernisses der Ermittlung der Verteilung einer Risikoposition $S_d = X_1 + \ldots + X_d$ entspricht. Ziel dieser Arbeit ist die Betrachtung verschiedener Methoden zur Berechnung der Verteilungsfunktion $\mathbb{P}[X_1 + \ldots + X_d \leq s]$, wenn die Zufallsvariablen $X_i, i = 1, \ldots, d$ voneinander abhängig sind.

In Kapitel 2 wird ein kurzer Überblick über operationelles Risiko und verschiedene Methoden zur Berechnung des Eigenkapitalerfordernisses gegeben.

Die für die Arbeit nötigen Grundlagen wie die Modellierung stochastischer Abhängigkeiten, die betrachteten Verteilungen und die Definition von Hyperwürfel und Simplex werden in Kapitel 3 eingeführt.

Da die Bestimmung der gemeinsamen Dichte abhängiger Zufallsvariablen sehr aufwendig sein kann, wird als Alternative zur numerischen Integration ein Algorithmus zur numerischen Berechnung der Verteilungsfunktion abhängiger, nicht-negativer Zufallsvariablen aus dem Artikel [1] von P. Arbenz, P. Embrechts und G. Puccetti präsentiert, der ausführlich im Kapitel 4 erläutert wird.

Die Ergebnisse des AEP-Algorithmus werden in Kapitel 5 vorgestellt und mit den Ergebnissen von numerischen Integrationsmethoden und Rekursion verglichen.

Im Kapitel 6 wird nach dem Vorbild adaptiver Integrationsmethoden ein adaptiver Algorithmus konstruiert. Anschließend werden die Ergebnisse der Methoden analysiert und Rückschlüsse auf die Anzahl der Simplizes gezogen, die von den Algorithmen benötigt wird, um eine vorgegebene Genauigkeit zu erreichen.

Die selbst erstellten R-Codes zu den vorgestellten Methoden befinden sich im Kapitel 7 der Arbeit.

Kapitel 2

Operationelles Risiko

In diesem Kapitel werden wir einen Überblick über verschiedene Ansätze zur Messung des operationellen Risikos schaffen. Die unten beschriebenen Methoden orientieren sich an [2], [3] und [5]. Als Erstes betrachten wir die Basel II Definition für operationelles Risiko wie sie in der Rahmenvereinbarung des Basler Ausschusses für Bankenaufsicht steht.

Definition 2.1. *Operationelles Risiko (OR) ist das Risiko des Verlustes infolge der Unangemessenheit oder des Versagens interner Prozesse, Menschen und Systeme oder aufgrund externer Ereignisse. Diese Definition schließt Rechtsrisiken ein, beinhaltet aber nicht strategische Risiken oder Reputationsrisiken.*

Nach der neuen Basler Eigenkapitalvereinbarung (Basel II) müssen international aktive Banken Kapital vorsehen, um verschiedene Arten von Risiken auszugleichen, wie zum Beispiel Marktrisiko, Kreditrisiko und operationelles Risiko. Wie in der Rahmenvereinbarung beschrieben wird, gibt es drei Methoden zur Berechnung der Eigenkapitalanforderung für operationelles Risiko in ein Kontinuum von wachsender Komplexität und Risikosensitivität.

Banken werden aufgefordert, sich entlang des Spektrums verfügbarer Ansätze zu bewegen, wenn sie komplexere Systeme und Verfahren zur Messung des operationellen Risikos entwickeln. Von international aktiven Banken und Banken mit signifikanter operationeller Risikoexposition wird erwartet, dass sie einen anspruchsvolleren Ansatz wählen als den *Basisindikatoransatz* (BIA), der für das Risikoprofil des Instituts geeignet ist. Eine Bank kann den Basisindikatoransatz oder den *Standardansatz* (SA) für gewisse Teile des operativen Geschäfts anwenden und den *Advanced Measurement Approach* (AMA) für andere Teile, sofern bestimmte Mindestkriterien erfüllt sind.

Der Basisindikatoransatz und der Standardansatz werden häufig als Top-down-Ansätze bezeichnet, in dem Sinn, dass das Eigenkapitalerfordernis als ein fixer Anteil des Einkommens berechnet wird. Der Advanced Measurement Approach wir auch Bottom-up-Ansatz genannt, im Sinn, dass das Eigenkapitalerfordernis durch aktuelle interne Verlustdaten geschätzt wird. Als Nächstes werden wir einen Überblick über diese drei Methoden geben, die in [5] genau beschrieben sind.

2.1 Basisindikatoransatz

Der *Basisindikatoransatz* ist der einfachste Ansatz. Bei diesem Ansatz wird das Bruttoeinkommen als Indikator für das Ausmaß des operationellen Risikos der Bank angesehen. Seien t die Jahre, für welche die Risikokapitalanforderung berechnet wird, $\mathbf{GI}_{t-k}$ das jährlich Bruttoeinkommen im Jahr $t-k$, $N_t = \sum_{k=1}^{3} \mathbb{1}_{\mathbf{GI}_{t-k}>0}$ die Anzahl der vorherigen drei Jahre, für welche das Bruttoeinkommen positiv ist, und $\alpha = 0.15$. Gemäß dem Basisindikatoransatz wird die Eigenkapitalanforderung für operationelles Risiko $\mathbf{RC}_t^{BIA}$ zum Zeitpunkt t berechnet durch

$$\mathbf{RC}_t^{BIA} = \frac{\alpha}{N_t} \sum_{k=1}^{3} \max\{\mathbf{GI}_{t-k}, 0\}.$$

Das bedeutet, dass das operationelle Risikokapital für das Jahr t ein α-Anteil des Durchschnitts des positiven jährlichen Bruttoeinkommens über die vergangenen drei Jahre ist. Es ist nicht definiert, wenn das jährliche Bruttoeinkommen in den drei Jahren vor t nicht-positiv ist.

2.2 Standardansatz

Beim Standardansatz werden die Tätigkeiten der Banken in die folgenden acht *Geschäftsbereiche* unterteilt: Unternehmensfinanzierung, Handel und Verkauf, Retail Banking, Commercial Banking, Zahlungsregulierung, Vermittlungsdienste, Vermögensverwaltung und Wertpapierprovisionsgeschäft. Das zu hinterlegende Eigenkapital für das operationelle Risiko wird gemäß dem *Standardansatz* berechnet durch

$$\mathbf{RC}_t^{SA} = \frac{1}{3} \sum_{k=1}^{3} \max\left\{ \sum_{b=1}^{8} \beta_b \mathbf{GI}_{t-k}^b, 0 \right\}, \tag{2.1}$$

wobei $\mathbf{GI}_{t-k}^b$ das jährliche Bruttoeinkommen im Jahr $t-k$ für den Geschäftsbereich b ist und $(\beta_1, \ldots, \beta_8) = (0.18, 0.18, 0.12, 0.15, 0.18, 0.15, 0.12, 0.12)$ die Gewichte der verschiedenen Geschäftsbereiche. Weiter ist zu beachten, dass $\sum_{b=1}^{8} \beta_b = 1.2 = 8\alpha$. Die Idee dieser Methode ist, dass das Bruttoeinkommen stellvertretend für den Umfang der Unternehmenstätigkeit ist und damit für den voraussichtlichen Umfang von operationellem Risiko in jedem einzelnen Geschäftsbereich. Die Eigenkapitalanforderung für jeden Geschäftsbereich $b \in \{1, \ldots, 8\}$ wird als β_b-Anteil des Bruttoeinkommens in diesem Geschäftsbereich bestimmt. Die Gleichung (2.1) setzt voraus, dass in jedem Jahr negatives Eigenkapital in einem Geschäftsbereich positives Eigenkapital in einem anderen Geschäftsbereich ausgleichen kann.

2.3 Advanced Measurement Approach

Mithilfe des *Advanced Measurement Approach* wird das Mindesteigenkapital durch das interne OR Messsystem der Bank bestimmt. Im Advanced Measurement Approach soll-

ten operationelle Verluste gemäß folgender acht Geschäftsbereiche sowie der folgenden sieben Verlust-Ereignistypen kategorisiert werden.

b	Geschäftsbereich	l	Ereignistyp
1	Unternehmensfinanzierung	1	Interner Betrug
2	Handel und Verkauf	2	Externer Betrug
3	Retail Banking	3	Beschäftigungspraktiken und
4	Commercial Banking		Sicherheit am Arbeitsplatz
5	Zahlungsregulierung	4	Kunden, Produkte und Geschäftspraktiken
6	Vermittlungsdienste	5	Sachschäden
7	Vermögensverwaltung	6	Betriebsstörung und Systemausfall
8	Wertpapierprovisionsgeschäft	7	Ausführung, Lieferung und Prozessmanagement

Tabelle 2.1: Geschäftsbereiche und Verlust-Ereignistypen

Wir werden nun ein Grundgerüst des Advanced Measurement Approach für die Berechnung des zu hinterlegenden Kapitals für operationelle Risiken vorstellen. Die Anzahl der Geschäftsbereiche und Ereignistypen können für unterschiedliche Banken variieren und sind oft von den verfügbaren Daten abhängig. Wir nehmen an es existieren historische Verlustdaten von folgender Form

$$\{X_{t-k,i}^{b,l} : k \in \{1,\dots,T\}, b \in \{1,\dots,8\}, l \in \{1,\dots,7\}, i \in \{1,\dots,N_{t-k}^{b,l}\}\},$$

wobei t das Jahr ist, für welches wir das zu hinterlegende Risikokapital bestimmen wollen, T die Anzahl der Jahre, für die historische Daten betrachtet werden, und $X_{t-k,i}^{b,l}$ den i-ten der $N_{t-k}^{b,l}$ Verluste bezeichnet, die den Geschäftsbereich b und den Ereignistyp l im Jahr $t-k$ beeinflussen. Die totale historische Schadensumme für den Geschäftsbereich b und den Ereignistyp l im Jahr $t-k$ kann durch diese Summe modelliert werden

$$L_{t-k}^{b,l} = \sum_{i=1}^{N_{t-k}^{b,l}} X_{t-k,i}^{b,l}.$$

Die totale historische Schadensumme im Jahr $t-k$ beträgt daher

$$L_{t-k} = \sum_{b=1}^{8} \sum_{l=1}^{7} \sum_{i=1}^{N_{t-k}^{b,l}} X_{t-k,i}^{b,l}, \qquad k \in \{1,\dots,T\}.$$

Unser Ziel ist es die Verteilung des totalen Verlustes L_t im Jahr t zu schätzen und das gewählte Risikomaß ρ_α für die geschätzte Verteilung zum α-Quantil zu berechnen, d.h.

$$\mathbf{RC}_t^{AMA} = \rho_\alpha(L_t)$$

zu berechnen. Da die Verteilung von L_t im Allgemeinen unbekannt ist, ist dieses Problem in der Regel schwer zu lösen. Daher sammelt man Risikomaße berechnet für

Verluste, die gewisse Geschäftsbereiche und Ereignistypen betreffen, für die genug Information über die zugrundeliegende Verteilung vorhanden ist. Im Besonderen vereinigt man Risikomaße, die für Verluste über Ereignistypen für fixe Geschäftsbereiche vereinigt wurden, d.h.

$$\mathbf{RC}_t^{AMA} = \sum_{b=1}^{8} \rho_\alpha(L_t^b),$$

wobei L_t^b der Verlust im Geschäftsbereich b im Jahr t ist, geschätzt durch $L_{t-k}^b = \sum_{l=1}^{7} \sum_{i=1}^{N_{t-k}^{b,l}} X_{t-k,i}^{b,l}$.

Kapitel 3

Grundlagen

Probleme wie die Berechnung von $\mathbb{P}[X_1 + \ldots + X_d \leq s]$ treten insbesondere bei der Bestimmung des Eigenkapitalerfordernisses im Versicherungs- und Finanzwesen auf, um die Risikoposition $S_d = X_1 + \ldots + X_d$ auszugleichen, die sich von einem Portfolio von d zufälligen Verlusten für eine gegebene multivariate Verteilung ableiten. Das Mindestkapitalerfordernis, das mit S_d assoziiert wird, wird normalerweise durch den Value-at-Risk der Verteilung von S_d für ein bestimmtes Konfidenzniveau berechnet. Daher ist die Berechnung eines VaR-basierten Eigenkapitalerfordernisses äquivalent zur Berechnung der Verteilung von S_d. Die Berechnung von $\mathbb{P}[X_1 + \ldots + X_d \leq s]$ ist eine schwierige Aufgabe. Es existieren verschiedene Modelle zur Berechnung von $\mathbb{P}[X_1 + \ldots + X_d \leq s]$, wenn die Zufallsvariablen $X_i, i = 1, \ldots, d$ unabhängig sind. Weit weniger ist bekannt für den Fall, dass die Zufallsvariablen abhängig voneinander sind.

3.1 Copula

Da wir uns mit dem Problem der Modellierung der stochastischen Abhängigkeit der Komponenten eines Zufallsvektors von Finanzrisikofaktoren befassen, werden wir in Anlehnung an [11] und [12] das Konzept der *Copula* betrachten und uns anschließend mit der Clayton Copula nach der Definition in [16] auseinandersetzen. Im gewissem Sinne enthält jede multivariate Verteilungsfunktion eines Zufallsvektors von Risikofaktoren sowohl eine Beschreibung des Randverhaltens der individuellen Risikofaktoren als auch eine Beschreibung der Abhängigkeitsstruktur. Zusätzlich ermöglichen Copulas einen Bottom-up-Ansatz zur Erstellung eines multivariaten Modells. Das ist besonders nützlich im Risikomanagement, wo oft mehr Information über das Randverhalten von individuellen Risikofaktoren verfügbar ist als über ihre Abhängigkeitsstruktur.

3.1.1 Definition und Eigenschaften

Definition 3.1. *Eine d-dimensionale Copula ist eine Verteilungsfunktion auf $[0,1]^d$ mit gleichverteilten Rändern auf dem Intervall $[0,1]$.*

Eine Copula kann auch durch folgende Eigenschaften charakterisiert werden:

1. $C(u_1, \ldots, u_d)$ ist wachsend in jeder Komponente u_i.

2. Für alle $u_i \in [0,1], i \in \{1, \ldots, d\}$ gilt, dass die Copula *gleichverteilte Ränder* hat, d.h. $C(1, \ldots, 1, u_i, 1, \ldots, 1) = u_i$ und, dass sie *geerdet* ist, d.h. $C(u_1, \ldots, u_{i-1}, 0, u_{i+1}, \ldots, u_d) = 0$.

3. C ist d-wachsend, d.h. für alle $a = (a_1, \ldots, a_d), b = (b_1, \ldots, b_d) \in [0,1]^d$ mit $a_i \leq b_i, i = 1, \ldots, d$ gilt, dass

$$\sum_{i_1=1}^{2} \ldots \sum_{i_d=1}^{2} (-1)^{i_1+\ldots+i_d} C(u_{1i_1}, \ldots, u_{di_d}) \geq 0, \tag{3.1}$$

   wobei $u_{i1} = a_i$ und $u_{i2} = b_i$ für alle $i \in 1, \ldots, d$

Die erste Eigenschaft ist erforderlich für jede multivariate Verteilungsfunktion. Die zweite Eigenschaft ist die Voraussetzung für die gleichverteilten Ränder und sagt aus, dass, falls bei einer Randverteilung die Nullwahrscheinlichkeit auftritt, dies auch bei der gemeinsamen Verteilung gelten muss. Die letzte Eigenschaft ist nicht so offensichtlich. Die Ungleichung (3.1) stellt sicher, dass wenn C die Verteilungsfunktion des Zufallsvektors $(U_1, \ldots, U_d)$ ist, $\mathbb{P}(a_1 \leq U_1 \leq b_1, \ldots, a_d \leq U_d \leq b_d)$ nicht-negativ ist. Erfüllt eine Funktion C diese Eigenschaften, dann ist sie eine Copula.

Die Bedeutung von Copulas bei der Untersuchung von multivariaten Verteilungsfunktionen wird im *Theorem von Sklar* zusammengefasst. Einerseits wird gezeigt, dass alle multivariaten Verteilungsfunktionen Copulas enthalten, anderseits können Copulas in Verbindung mit eindimensionalen Verteilungsfunktionen verwendet werden, um multivariate Verteilungsfunktionen zu konstruieren.

Theorem 3.2. *Sei H eine multivariate Verteilungsfunktion mit eindimensionalen Randverteilungen $F_1, \ldots, F_d$. Dann existiert eine Copula $C : [0,1]^d \to [0,1]$, sodass für alle $x_1, \ldots, x_d$ in $\overline{\mathbb{R}} = [-\infty, \infty]$*

$$H(x_1, \ldots, x_d) = C(F_1(x_1), \ldots, F_d(x_d)) \tag{3.2}$$

gilt. Sind die Ränder stetig so ist C eindeutig. Andernfalls ist C eindeutig bestimmt auf Ran $F_1 \times$ Ran $F_2 \times \ldots \times$ Ran F_d, *wobei* Ran $F_i = F_i(\overline{\mathbb{R}})$ *für $i = 1, \ldots, d$. Umgekehrt gilt, wenn C eine Copula ist und $F_1, \ldots, F_d$ eindimensionale Verteilungsfunktionen, dann ist die Funktion H, definiert in (3.2), eine multivariate Verteilungsfunktion mit Rändern $F_1, \ldots, F_d$.*

Den Beweis zum Theorem 3.2 kann man in [11, Theorem 5.3] nachlesen.

Satz 3.3. *Sei C eine Copula. Für beliebige $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_d \in [0,1]$ existiert die partielle Ableitung $\partial C / \partial u_i$ für fast alle u_i mit $i \in \{1, \ldots, d\}$, und es gilt*

$$0 \leq \frac{\partial}{\partial u_i} C(u_1, \ldots, u_d) \leq 1.$$

Für den Beweis von Satz 3.3 sei auf [12, Theorem 2.2.7.] verwiesen. Da Copulas als Verteilungsfunktionen definiert sind, lassen sich Ausdrücke für die Copula-Dichte herleiten, falls diese existiert. Die Komonotonie-Copula und die Kontramonotonie-Copula sind Beispiele für Copulas, die nicht absolut stetig sind. Jede Copula C lässt sich in einen absolut stetigen Teil A_C und einen singulären Teil S_C zerlegen

$$C(u_1, \dots, u_d) = A_C(u_1, \dots, u_d) + S_C(u_1, \dots, u_d),$$

wobei

$$A_C(u_1, \dots, u_d) = \int_0^{u_1} \cdots \int_0^{u_d} \frac{\partial^d}{\partial s_1 \cdots \partial s_d} C(s_1, \dots, s_d) ds_1 \cdots ds_d$$

und

$$S_C(u_1, \dots, u_d) = C(u_1, \dots, u_d) - A_C(u_1, \dots, u_d).$$

Infolge von Satz 3.3 existieren die partiellen Ableitungen fast überall in $[0,1]^d$. Im Gegensatz zu multivariaten Verteilungsfunktionen sind die Randverteilungen der Copulas immer stetig, daher haben Copulas keine Atome. Existiert die Dichtefunktion c einer Copula C, so ist sie gegeben durch

$$c(u_1, \dots, u_d) = \frac{\partial^d C(u_1, \dots, u_d)}{\partial u_1 \cdots \partial u_d}. \tag{3.3}$$

Definition 3.4. *Sei H eine absolut stetige, multivariate Verteilungsfunktion des Zufallsvektors $(U_1, \dots, U_d)$ mit absolut stetiger Copula C, d.h. $C = A_C$, und streng monoton wachsenden, stetigen Randverteilungen $F_1, \dots, F_d$. Wir können $C(u_1, \dots, u_d) = H(F_1^{\leftarrow}(u_1), \dots, F_d^{\leftarrow}(u_d))$ differenzieren, um zu sehen, dass die* Copula-Dichte *durch*

$$c(u_1, \dots, u_d) = \frac{h(F_1^{-1}(u_1), \dots, F_d^{-1}(u_d))}{f_1(F_1^{-1}(u_1)) \cdots f_d(F_d^{-1}(u_d))}$$

gegeben ist, wobei h die gemeinsame Dichtefunktion von H ist, $f_1, \dots, f_d$ die Randdichten und $F_1^{-1}, \dots, F_d^{-1}$ die verallgemeinerte Inversen der Randverteilungen sind.

Unter Verwendung von (3.3) und Sklar's Theorem kann die gemeinsame Dichtefunktion h geschrieben werden als

$$\begin{aligned} h(x_1, \dots, x_d) &= \frac{\partial^d H(x_1, \dots, x_d)}{\partial x_1 \cdots \partial x_d} = \frac{\partial^d}{\partial x_1 \cdots \partial x_d} \left(C(F_1(x_1), \dots, F_d(x_d)) \right) \\ &= \frac{\partial^d C}{\partial u_1 \cdots \partial u_d}(F_1(x_1), \dots, F_d(x_d)) \prod_{i=1}^{d} \frac{\partial F_i(x_i)}{\partial x_i}. \end{aligned}$$

Substituieren wir $u_i = F_i(x_i)$ für $i \in \{1, \dots, d\}$, so erhalten wir

$$\begin{aligned} h(x_1, \dots, x_d) &= \frac{\partial^d C(u_1, \dots, u_d)}{\partial u_1 \cdots \partial u_d} \prod_{i=1}^{d} f_i(x_i) \\ &= c(F_1(x_1), \dots, F_d(x_d)) \prod_{i=1}^{d} f_i(x_i). \end{aligned} \tag{3.4}$$

3.1.2 Clayton Copula

Eine *archimedische Copula* ist eine d-dimensionale Copula definiert durch

$$C(u_1, \ldots, u_d) := \varphi^{[-1]}[\varphi(u_1) + \ldots + \varphi(u_d)],$$

wobei $\varphi(u)$ Generator der Copula genannt wird.

Definition 3.5. *Der Generator* $\varphi : [0,1] \to [0,\infty]$ *ist eine streng monoton fallende Funktion mit* $\varphi(1) = 0$. *Die* Pseudo-Inverse *von* φ *ist eine Funktion* $\varphi^{[-1]} : [0,\infty] \to [0,1]$ *gegeben durch*

$$\varphi^{[-1]}(t) = \begin{cases} \varphi^{-1}(t), & \textit{falls } 0 \leq t \leq \varphi(0), \\ 0, & \textit{falls } \varphi(0) \leq t \leq \infty. \end{cases}$$

Demnach ist $\varphi^{[-1]}$ stetig und monoton fallend auf $[0,\infty]$ und streng monoton fallend auf $[0,\varphi(0)]$. Weiter gilt $\varphi^{[-1]}(\varphi(u)) = u$ auf $[0,1]$ und

$$\varphi(\varphi^{[-1]}(t)) = \begin{cases} t, & \text{falls } 0 \leq t \leq \varphi(0), \\ \varphi(0), & \text{falls } \varphi(0) \leq t \leq \infty. \end{cases}$$

Falls $\varphi(0) = \infty$, so gilt $\varphi^{[-1]} = \varphi^{-1}$.

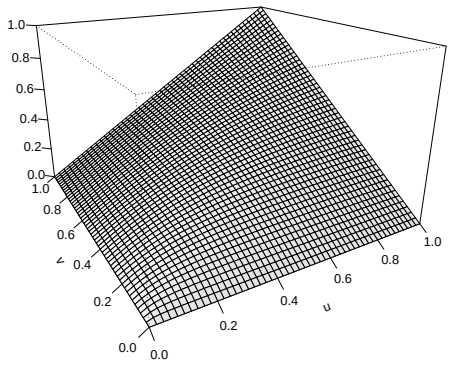

Abbildung 3.1: Graph einer bivariaten Clayton Copula mit Parameter $\delta = 1.2$.

Einer der bekanntesten Vertreter der Familie der archimedischen Copulas ist die *Clayton Copula*. Sie ist gegeben durch

$$C_\delta^{Cl}(u_1, \ldots, u_d) = (u_1^{-\delta} + \ldots + u_d^{-\delta} - d + 1)^{-\frac{1}{\delta}}, \tag{3.5}$$

wobei $0 < \delta < \infty$ der Abhängigkeitsparameter ist. Für $\lim_{\delta \to 0} C_\delta^{Cl}$ erhalten wir die Unabhängigkeits-Copula und für $\lim_{\delta \to \infty} C_\delta^{Cl}$ erhalten wir die Komonotonie-Copula. Der Generator und der inverse Generator der Clayton Copula sind gegeben durch $\varphi(u) = u^{-\delta} - 1$ und $\varphi^{-1}(t) = (1+t)^{-\frac{1}{\delta}}$.

Die Dichtefunktion der Clayton Copula ist gegeben durch

$$c_\delta^{Cl}(u_1, \ldots, u_d) = (u_1^{-\delta} + \ldots + u_d^{-\delta} - d + 1)^{-d - \frac{1}{\delta}} \prod_{i=1}^{d} u_1^{-\delta 1}((i-1)\delta + 1)). \tag{3.6}$$

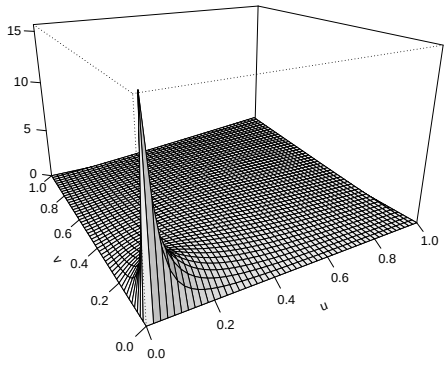

Abbildung 3.2: Graph der Dichte einer bivariaten Clayton Copula mit Parameter $\delta = 1.2$.

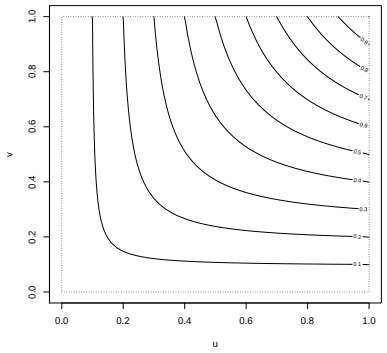

Abbildung 3.3: Konturdiagramm einer bivariaten Clayton Copula mit Parameter $\delta = 1.2$.

Als Nächstes betrachten wir für den zweidimensionalen Fall die Menge der Punkte auf $[0,1]^2$, für die die Clayton Copula jeweils denselben konstanten Wert annimmt, also $C_\delta^{Cl}(u,v) = \alpha$ wobei $\alpha \in (0,1)$ ist. Durch Umformen erhalten wir

$$v_\alpha(u) = \left(\alpha^{-\delta} - u^{-\delta} + 1\right)^{-\frac{1}{\delta}} \qquad \text{für } \alpha \leq u \leq 1.$$

Dadurch können wir mithilfe von Konturlinien auf eine anschaulichere Weise in Abbildung 3.3 die Form und Gestalt einer bivariaten Clayton Copula betrachten.

3.2 Benötigte Verteilungen

Für die im Laufe der Arbeit betrachteten Beispiele werden die Exponentialverteilung und die Pareto-Verteilung benötigt. Die in den nachstehenden Abschnitten beschrieben werden.

3.2.1 Exponentialverteilung

Die Exponentialverteilung einer stetigen Zufallsvariable X ist beschrieben durch die Wahrscheinlichkeitsdichte f und die Verteilungsfunktion F von der Form:

$$f(x) = \lambda e^{-\lambda x}, F(x) = 1 - e^{-\lambda x} \text{ für } x \geq 0$$

Die Verteilung wird durch den positiven Parameter λ charakterisiert und ihr Erwartungswert ist gegeben durch $\mathbb{E}[X] = \frac{1}{\lambda}$.

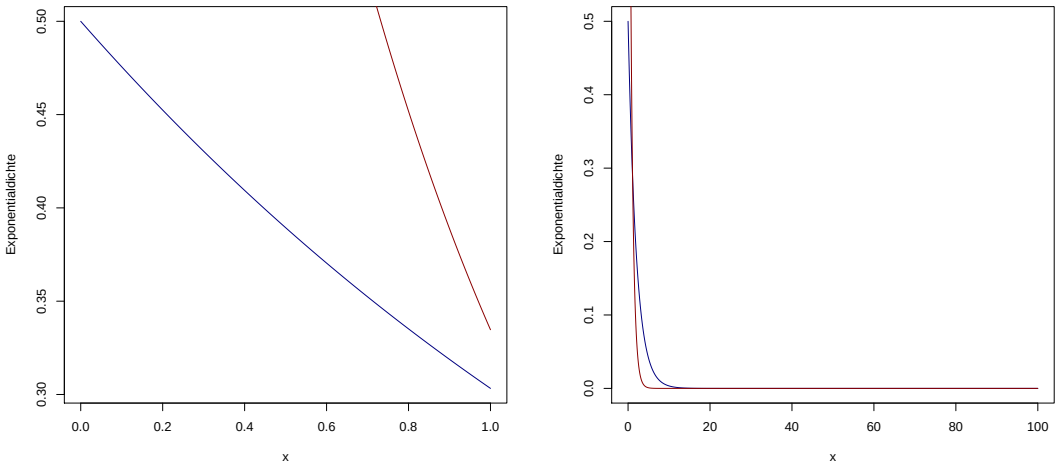

Abbildung 3.4: Dichte einer Exponentialverteilung mit Parameter $\lambda = 0.5, 1.5$.

Weiter lässt sich die Quantilfunktion der Exponentialverteilung angeben durch

$$F^{-1}(p) = \frac{-ln(1-p)}{\lambda}.$$

3.2.2 Pareto-Verteilung

Die Pareto-Verteilung wie in [14] beschrieben ist eine Heavy-tailed-Verteilung, die weitgehend bei der Modellierung von Verlusten Verwendung findet, wenn die Wahrscheinlichkeit für Großschadenereignisse sehr hoch ist. Eine stetige Zufallsvariable X heißt Pareto-verteilt $Pa(\alpha, \theta)$ mit Parametern $\alpha > 0$ und $\theta > 0$, wenn sie die Wahrscheinlichkeitsdichte

$$f(x) = \frac{\alpha\theta^{\alpha}}{(x+\theta)^{\alpha+1}}$$

besitzt. Die Verteilungsfunktion ist gegeben durch

$$F(x) = 1 - \left(\frac{\theta}{x+\theta}\right)^{\alpha}.$$

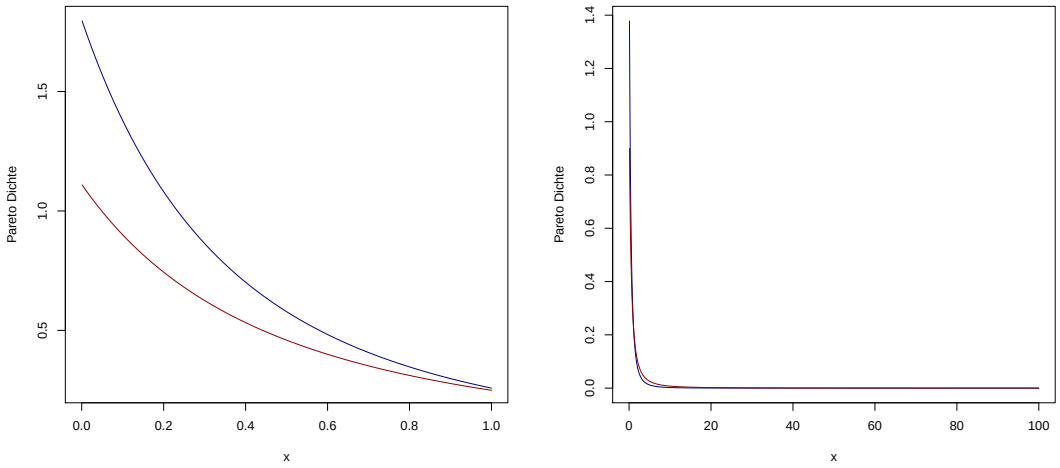

Abbildung 3.5: Dichte einer Pareto-Verteilung mit Parameter $\alpha = 0.9, 1.8$ und $\theta = 1$.

Der Parameter α bestimmt, wie schnell sich die Verteilungsfunktion dem Wert 1 annähert, d.h. je höher α ist, desto schneller läuft die Verteilungsfunktion gegen 1. Der Parameter θ bezeichnet die untere Grenze der Daten. Weiter sei

$$F^{-1}(p) = \frac{\theta}{(1-p)^{\frac{1}{\alpha}}} - \theta$$

die Quantilfunktion der Pareto-Verteilung.

3.3 Hyperwürfel und Simplex

In diesem Abschnitt werden wichtige Grundlagen für die Beschreibung des AEP-Algorithmus definiert. Im weiteren Verlauf der Arbeit werden Vektoren fett gedruckt geschrieben, zum Beispiel $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}^d, d > 1$. Weiters bezeichne $\mathbf{e}_k$ den $k-ten$ Vektor der kanonischen Basis von $\mathbb{R}^d$ und $D = \{1, \ldots, d\}$. Gegeben sei der Vektor $\mathbf{b} = (b_1, \ldots, b_d) \in \mathbb{R}^d$ und eine reelle Zahl h, dann bezeichnet $\mathcal{Q}(\mathbf{b}, h) \subset \mathbb{R}^d$ einen Hyperwürfel definiert durch

$$\mathcal{Q}(\mathbf{b}, h) = \begin{cases} \bigtimes_{k=1}^{d} (b_k, b_k + h], & \text{wenn } h > 0, \\ \bigtimes_{k=1}^{d} (b_k + h, b_k], & \text{wenn } h < 0. \end{cases} \tag{3.7}$$

Zur Vereinfachung der Notation setzen wir $\mathcal{Q}(\mathbf{b}, 0) = \varnothing$. Auf einem Wahrscheinlichkeitsraum $(\Omega, \mathfrak{S}, \mathbb{P})$ haben die Zufallsvariablen $X_1, \ldots, X_d$ eine d-dimensionale multivariate Verteilung H. H induziert das Wahrscheinlichkeitsmaß V_H auf $\mathbb{R}^d$ durch

$$V_H \left[\bigtimes_{i=1}^{d} (-\infty, x_i] \right] = H(x_1, \ldots, x_d).$$

Wir bezeichnen mit $\mathbf{i}_0, \ldots, \mathbf{i}_N$ alle 2^d Vektoren in $\{0, 1\}^d$, wobei $N = 2^d - 1$. Anders als in [1], sind das alle Vektoren der Länge d, die mithilfe einer systematischen Binärdarstellung als Kombination der Elemente $\{0, 1\}$ erzeugt werden, wie $\mathbf{i}_0 = (0, \ldots, 0), \mathbf{i}_1 = (1, 0, \ldots, 0), \mathbf{i}_2 = (0, 1, 0 \ldots, 0), \ldots, \mathbf{i}_N = \mathbf{1} = (1, \ldots, 1)$. Als Nächstes sei $\#\mathbf{i} = \sum_{k=1}^{d} i_k$ die Anzahl der Einser im Vektor $\mathbf{i}$, zum Beispiel, $\#\mathbf{i}_0 = 0$ und $\#\mathbf{i}_N = d$. Das V_H-Maß eines Hyperwürfels $\mathcal{Q}(\mathbf{b}, h), h > 0$, kann ausgedrückt werden durch

$$V_H[\mathcal{Q}(\mathbf{b}, h)] = \mathbb{P}[X_k \in (b_k, b_k + h], k \in D] = \sum_{j=0}^{N} (-1)^{d - \#\mathbf{i}_j} H(\mathbf{b} + h\mathbf{i}_j). \tag{3.8}$$

Wir definieren das d-dimensionale Simplex $\mathcal{S}(\mathbf{b}, h) \subset \mathbb{R}^d$ durch

$$\mathcal{S}(\mathbf{b}, h) = \begin{cases} \left\{ \mathbf{x} \in \mathbb{R}^d : x_k - b_k > 0, k \in D \text{ und } \sum_{k=1}^{d} (x_k - b_k) \leq h \right\}, & \text{wenn } h > 0, \\ \left\{ \mathbf{x} \in \mathbb{R}^d : x_k - b_k \leq 0, k \in D \text{ und } \sum_{k=1}^{d} (x_k - b_k) > h \right\}, & \text{wenn } h < 0. \end{cases} \tag{3.9}$$

Wieder soll $\mathcal{S}(\mathbf{b}, 0) = \varnothing$ gelten. Letztlich bezeichnen wir mit λ_d das Lebesgue-Maß auf $\mathbb{R}^d$. Das Lebesgue-Maß auf dem Simplex $\mathcal{S}(\mathbf{b}, h)$ ist nach [1] gegeben durch

$$\lambda_d[\mathcal{S}(\mathbf{b}, h)] = \frac{|h|^d}{d!}. \tag{3.10}$$

Kapitel 4

Der AEP-Algorithmus zur schnellen Berechnung der Verteilung von Summen abhängiger Zufallsvariablen

In einem 2011 im Bernoulli Journal veröffentlichten Artikel stellen Arbenz, Embrechts und Puccetti einen Algorithmus zur numerischen Berechnung der Verteilungsfunktion von d abhängigen, nicht-negativen Zufallsvariablen für eine gegebene absolut stetige multivariate Verteilungsfunktion vor. Für eine gegebene multivariate Verteilungsfunktion H approximiert der Algorithmus das H-Maß auf einem Simplex durch eine endliche Summe der H-Maße von Hyperwürfeln [6].

4.1 Beschreibung des AEP-Algorithmus

Für den weiteren Verlauf der Arbeit nehmen wir an, dass die Zufallsvariablen $X_1, \ldots, X_d$ nicht-negativ seien, d.h. $\mathbb{P}[X_k < 0] = 0, k \in D$. Weiters nehmen wir an, dass die multivariate Verteilung H des Zufallsvektors $(X_1, \ldots, X_d)$ bekannt ist und definieren $S_d = X_1 + \ldots + X_d$. Unser Ziel ist die nummerische Berechnung von

$$\mathbb{P}[S_d \leq s] = V_H[\mathcal{S}(\mathbf{0}, s)]$$

für einen fixen positiven Wert s. Wegen (3.8) ist es sehr leicht das V_H-Maß für Hyperwürfel auf $\mathbb{R}^d$ zu berechnen. Die Idee hinter dem AEP-Algorithmus ist es das Simplex $\mathcal{S}(\mathbf{0}, s)$ durch Hyperwürfel zu approximieren.

Zu Beginn der n-ten Iteration $n \in \mathbb{N}$ (der Algorithmus startet bei $n = 1$), erhält der Algorithmus als Input N^{n-1} Simplizes, die wir mit $\mathcal{S}_n^k = \mathcal{S}(\mathbf{b}_n^k, h_n^k)$ bezeichnen, für $k = 1, \ldots, N^{n-1}$. Zu jedem Simplex wird der Wert $s_n^k \in \{-1, 1\}$ assoziiert, der anzeigt, ob das Maß des Simplex addiert ($s_n^k = 1$) oder subtrahiert wird ($s_n^k = -1$), um eine

Approximation von $V_H[\mathcal{S}(\mathbf{0}, s)]$ zu berechnen. Jedes Simplex wird durch einen Hyperwürfel $\mathcal{Q}_n^k = \mathcal{Q}(\mathbf{b}_n^k, \alpha h_n^k)$ mit $\alpha \in [1/d, 1)$ und N Simplizes $\mathcal{S}_{n+1}^k = \mathcal{S}(\mathbf{b}_{n+1}^k, h_{n+1}^k)$ zerlegt. Das V_H-Maß für jedes Simplex $\mathcal{S}_n^k$ kann mithilfe des folgenden Theorems aus [1] berechnet werden.

Theorem 4.1. *Für alle $\mathbf{b} \in \mathbb{R}^d, h \in \mathbb{R}$ und $\alpha \in [1/d, 1)$ gilt, dass*

$$V_H[\mathcal{S}_n^k] = V_H[\mathcal{Q}_n^k] + \sum_{j=1}^{N} m^j V_H[\mathcal{S}_{n+1}^{Nk-N+j}], \tag{4.1}$$

wobei die Folgen $\boldsymbol{b}_n^k, h_n^k$ und m^j durch ihre Anfangswerte $\boldsymbol{b}_1^1 = \boldsymbol{0}$, $h_1^1 = s$ und

$$\begin{gathered} \boldsymbol{b}_{n+1}^{Nk-N+j} = \boldsymbol{b}_n^k + \alpha h_n^k \boldsymbol{i}_j, \qquad h_{n+1}^{Nk-N+j} = (1 - \#\boldsymbol{i}_j \alpha) h_n^k, \\ m^j = \begin{cases} (-1)^{1+\#\boldsymbol{i}_j}, & \text{wenn } \#\boldsymbol{i}_j < 1/\alpha, \\ 0, & \text{wenn } \#\boldsymbol{i}_j = 1/\alpha, \\ (-1)^{d+1-\#\boldsymbol{i}_j}, & \text{wenn } \#\boldsymbol{i}_j > 1/\alpha \end{cases} \end{gathered} \tag{4.2}$$

definiert sind für alle $j = 1, \ldots, N$ und $k = 1, \ldots, N^{n-1}$.

Für den Beweis des Theorems benötigen wir einige Lemmata. Wir bezeichnen mit δ_{ij} das Kronecker-Delta, das definiert ist als

$$\delta_{ij} = \begin{cases} 0, & \text{falls } i \neq j, \\ 1, & \text{falls } i = j. \end{cases}$$

Lemma 4.2. *Seien $i, j \in D$ fix mit $i \neq j$. Dann gilt für alle $h, s \in \mathbb{R}$ mit $hs \geq 0$ und $\mathbf{b} \in \mathbb{R}^d$, dass*

$$\mathcal{S}(\mathbf{b} + h\mathbf{e}_i, s) \cap \mathcal{S}(\mathbf{b} + h\mathbf{e}_j, s) = \begin{cases} \mathcal{S}(\mathbf{b} + h\mathbf{e}_i + h\mathbf{e}_j, s - h), & \textit{falls } |h| < |s|, \\ \varnothing, & \textit{falls } |h| \geq |s|. \end{cases}$$

Beweis. Als Erstes zeigen wir die Inklusion $\subseteq$. Angenommen $0 < s \leq h$. Nach der Definition (3.9) gilt für einen Vektor $\mathbf{x} \in \mathcal{S}(\mathbf{b} + h\mathbf{e}_i, s)$, dass

$$x_k > b_k + \delta_{ik} h, k \in D \text{ und } \sum_{k=1}^{d} (x_k - b_k - \delta_{ik} h) \leq s.$$

Daraus folgt, dass

$$x_j \leq b_j + s - \sum_{k \neq j} (x_k - b_k - \delta_{ik} h) < b_j + s \leq b_j + h$$

gilt, das heißt $\mathbf{x} \notin \mathcal{S}(\mathbf{b}+h\mathbf{e}_j, s)$. Wir nehmen nun an, dass $0 < h < s$. Für einen Vektor $\mathbf{x} \in \mathcal{S}(\mathbf{b} + h\mathbf{e}_i, s) \cap \mathcal{S}(\mathbf{b} + h\mathbf{e}_j, s)$ gilt, dass

$$x_k - b_k > 0, k \in D \qquad \text{mit } x_i > b_i + h \text{ und } x_j > b_j + h. \tag{4.3}$$

Sei wieder $\mathbf{x} \in \mathcal{S}(\mathbf{b} + h\mathbf{e}_i, s)$, folglich erhalten wir $\sum_{k=1}^{d}(x_k - (b_k + h\delta_{ik})) \leq s$. Subtrahieren wir h von beiden Seiten der letzten Ungleichung, so erhalten wir

$$\sum_{k=1}^{d}(x_k - (b_k + h\delta_{ik} + h\delta_{jk})) \leq s - h. \tag{4.4}$$

Die Gleichungen (4.3) und (4.4) zeigen, dass $\mathbf{x} \in \mathcal{S}(\mathbf{b} + h\mathbf{e}_j + h\mathbf{e}_i, s - h)$. Der Fall $h, s < 0$ folgt analog.

Nun zeigen wir die Inklusion $\supseteq$. Im Fall $0 < s \leq h$, gibt es nichts zu zeigen. Daher nehmen wir an, dass $0 < h < s$. Für jedes fixe $\mathbf{x} \in \mathcal{S}(\mathbf{b} + h\mathbf{e}_j + h\mathbf{e}_i, s - h)$, gilt (4.4) für $x_k - (b_k + h\delta_{ik} + h\delta_{jk}) > 0, k \in D$. Addieren wir $h\delta_{jk}$ zur Summe auf der linken Seite und h zur rechten Seite von (4.4), so erhalten wir

$$\sum_{k=1}^{d}(x_k - (b_k + h\delta_{ik})) \leq s. \tag{4.5}$$

Da $(x_k - (b_k + h\delta_{ik}))$ positiv ist für alle $k \in D$, folgt aus (4.5), dass $\mathbf{x} \in \mathcal{S}(\mathbf{b} + h\mathbf{e}_i, s)$. Durch ähnliche Überlegungen erhalten wir, dass $\mathbf{x} \in \mathcal{S}(\mathbf{b} + h\mathbf{e}_j, s)$. Der Fall $h, s < 0$ folgt analog und der Fall $hs = 0$ ist trivial. □

Lemma 4.3. *Für alle* $\mathbf{b} \in \mathbb{R}^d, h \in \mathbb{R}$ *und* $\alpha \in (0, 1)$ *gilt, dass*

$$\mathcal{S}(\mathbf{b}, h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h) = \bigcup_{k=1}^{d} \mathcal{S}(\mathbf{b} + \alpha h\mathbf{e}_k, h - \alpha h).$$

Beweis. Zunächst zeigen wir die Inklusion $\subseteq$. Sei als Erstes $h > 0$. Wenn $\mathbf{x} \in \mathcal{S}(\mathbf{b}, h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)$, dann ist $x_k > b_k, k \in D$ und $\sum_{k=1}^{d}(x_k - b_k) \leq h$, solange nach Definition (3.7) ein $j \in D$ existiert, sodass $x_j - b_j > \alpha h$. Für dieses j können wir folgendes schreiben:

$$\sum_{k=1}^{d}(x_k - (b_k + \delta_{jk}\alpha h)) \leq h - \alpha h \qquad \text{mit } x_k - (b_k + \delta_{jk}\alpha h) > 0, k \in D. \tag{4.6}$$

Daraus erhalten wir $\mathbf{x} \in \mathcal{S}(\mathbf{b} + \alpha h\mathbf{e}_j, h - \alpha h) \subset \bigcup_{k=1}^{d} \mathcal{S}(\mathbf{b} + \alpha h\mathbf{e}_k, h - \alpha h)$.

Zum Beweis der Inklusion $\supseteq$ sei $\mathbf{x} \in \bigcup_{k=1}^{d} \mathcal{S}(\mathbf{b} + \alpha h\mathbf{e}_k, h - \alpha h)$, das heißt es existiert ein $j \in D$, für welches $\mathbf{x}$ die Ungleichung (4.6) erfüllt. Es folgt, dass $x_j \geq b_j + \alpha h$

(daher ist $\mathbf{x} \notin \mathcal{Q}(\mathbf{b}, \alpha h)$) und $\sum_{k=1}^{d}(x_k - b_k) \leq h - \alpha h + \alpha h = h$. Weiter nehmen wir zur Kenntnis, dass (4.6) $x_k > b_k, k \in D$ impliziert, und wir erhalten schließlich $\mathbf{x} \in \mathcal{S}(\mathbf{b}, h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)$. Der Fall $h < 0$ ist analog, während der Fall $h = 0$ trivial ist. □

Lemma 4.4. *Für alle* $\mathbf{b} \in \mathbb{R}^d, h \in \mathbb{R}$ *und* $\alpha \in (0, 1)$ *gilt, dass*

$$\mathcal{Q}(\mathbf{b}, \alpha h) \setminus \mathcal{S}(\mathbf{b}, h) = \mathcal{S}(\mathbf{b} + \alpha h\mathbf{1}, h - \alpha dh) \cap \mathcal{Q}(\mathbf{b}, \alpha h).$$

Beweis. Als Erstes zeigen wir die Inklusion $\subseteq$. Für $\alpha = 1/d$ folgt das Lemma direkt. Wir wählen $\alpha \in (1/d, 1)$ und nehmen $h > 0$ an. Wenn $\mathbf{x} \in \mathcal{Q}(\mathbf{b}, \alpha h) \setminus \mathcal{S}(\mathbf{b}, h)$, dann gilt $x_k > b_k$ für alle $k \in D$. Da $\mathbf{x} \notin \mathcal{S}(\mathbf{b}, h)$, erhalten wir $\sum_{i=1}^{d}(x_i - b_i) > h$. Da $x_k \leq b_k + \alpha h$ gilt für alle $k \in D$, können wir

$$\sum_{k=1}^{d}(x_k - b_k - \alpha h) > h - \alpha dh \qquad \text{mit } x_k - (b_k + \alpha h) \leq 0 \text{ für alle } k \in D \tag{4.7}$$

schreiben. Da $h - d\alpha h = h(1 - d\alpha) < 0$, stellen wir fest, dass $\mathbf{x} \in \mathcal{S}(\mathbf{b} + \alpha h\mathbf{1}, h - \alpha dh)$ und somit erhalten wir durch unsere Annahme, dass $\mathbf{x} \in \mathcal{S}(\mathbf{b}+\alpha h\mathbf{1}, h-\alpha dh) \cap \mathcal{Q}(\mathbf{b}, \alpha h)$.

Wir kommen zum Beweis der verbleibenden Inklusion $\supseteq$. Sei $\mathbf{x} \in \mathcal{S}(\mathbf{b} + \alpha h\mathbf{1}, h - \alpha dh) \cap \mathcal{Q}(\mathbf{b}, \alpha h)$. Wegen $h - \alpha dh < 0$ folgt, dass (4.7) gilt. Das impliziert, dass $\sum_{k=1}^{d}(x_k - b_k) > h$ gilt und daraus folgt, dass $\mathbf{x} \notin \mathcal{S}(\mathbf{b}, h)$. Der Fall $h < 0$ ist analog, während der Fall $h = 0$ trivial ist. □

Für den Beweis von Theorem 4.1 verwenden wir eine Notation, die unabhängig von dem Iterationsschritt ist. Es sei

$$V_H[\mathcal{S}(\mathbf{b}, h)] = V_H[\mathcal{Q}(\mathbf{b}, \alpha h)] + \sum_{j=1}^{N} m^j V_H[\mathcal{S}(\mathbf{b}^j, h^j)],$$

für alle $j = 1, \ldots, N$.

Beweis von Theorem 4.1. Der Fall $h = 0$ ist trivial. Wir nehmen nun an, dass $h \neq 0$. Unter Verwendung der folgenden allgemeinen Eigenschaften zweier Mengen A, B:

1. $B = (A \cup (B \setminus A)) \setminus (A \setminus B)$
2. $(A \setminus B) \subset A \cup (B \setminus A)$
3. $A \cap (B \setminus A) = \varnothing$

erhalten wir, dass

$$V_H[\mathcal{S}(\mathbf{b}, h)] = V_H[\mathcal{Q}(\mathbf{b}, \alpha h)] + V_H[\mathcal{S}(\mathbf{b}, h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)] - V_H[\mathcal{Q}(\mathbf{b}, \alpha h) \setminus \mathcal{S}(\mathbf{b}, h)]. \tag{4.8}$$

Unter Verwendung der Notation $\mathcal{S}^k = \mathcal{S}(\mathbf{b} + \alpha h \mathbf{e}_k, h - \alpha h)$ impliziert das Lemma 4.3 für den zweiten Summanden in (4.8), dass

$$V_H[\mathcal{S}(\mathbf{b}, h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)] = V_H\left[\bigcup_{k=1}^{d} \mathcal{S}^k\right] = \sum_{k=1}^{d} (-1)^{k+1} \sum_{I \subset D, |I|=k} V_H\left[\bigcap_{i \in I} \mathcal{S}^i\right]. \quad (4.9)$$

Wir fixieren $I \subset D$ mit $I = \{n_1, \ldots, n_k\}$. Die wiederholte Verwendung von Lemma 4.2 führt zu

$$\bigcap_{i \in I} \mathcal{S}(\mathbf{b} + \alpha h \mathbf{e}_{n_i}, h - \alpha h) = \begin{cases} \mathcal{S}\left(\mathbf{b} + \alpha h \sum_{j=1}^{k} \mathbf{e}_{n_j}, h(1 - k\alpha)\right), & \text{falls } k\alpha < 1, \\ \varnothing, & \text{falls } k\alpha \geq 1. \end{cases}$$

Durch Substitution des letzten Ausdrucks in (4.9) erhalten wir, dass

$$\begin{aligned} V_H[\mathcal{S}(\mathbf{b}, h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)] &= \sum_{\substack{k \in D, \\ k\alpha < 1}} (-1)^{k+1} \sum_{\substack{\mathbf{i}_r \in \{0,1\}^d, \\ \#\mathbf{i}_r = k}} V_H[\mathcal{S}(\mathbf{b} + \alpha h \mathbf{i}_r, h(1 - k\alpha))] \\ &= \sum_{\substack{\mathbf{i} \in \{0,1\}^d, \\ 0 < \#\mathbf{i} < 1/\alpha}} (-1)^{\#\mathbf{i}+1} V_H[\mathcal{S}(\mathbf{b} + \alpha h \mathbf{i}, h(1 - \#\mathbf{i}\alpha))]. \end{aligned} \quad (4.10)$$

Wenden wir Lemma 4.4 auf den dritten Summanden in (4.8) an, so erhalten wir

$$\begin{aligned} &V_H[\mathcal{Q}(\mathbf{b}, \alpha h) \setminus \mathcal{S}(\mathbf{b}, h)] \\ &\quad = V_H[\mathcal{S}(\mathbf{b} + \alpha h \mathbf{1}, h - \alpha d h) \cap \mathcal{Q}(\mathbf{b}, \alpha h)] \\ &\quad = V_H[\mathcal{S}(\mathbf{b} + \alpha h \mathbf{1}, h - \alpha d h)] - V_H[\mathcal{S}(\mathbf{b} + \alpha h \mathbf{1}, h - \alpha d h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)]. \end{aligned} \quad (4.11)$$

Beachte, dass wenn $\alpha = 1/d$ gilt, dass das Maß in (4.11) Null ist. Daher können wir annehmen, dass $\alpha \neq 1/d$ gilt. Verwenden wir, dass $\mathcal{Q}(\mathbf{b}, \alpha h) = \mathcal{Q}(\mathbf{b} + \alpha h \mathbf{1}, -\alpha h)$ gilt, und definieren $\hat{\mathbf{b}} = \mathbf{b} + \alpha h \mathbf{1}, \hat{\alpha} = -\alpha/(1 - \alpha d) > 1/d$ und $\hat{h} = h(1 - \alpha d)$, so können wir

$$V_H[\mathcal{S}(\mathbf{b} + \alpha h \mathbf{1}, h - \alpha d h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)] = V_H[\mathcal{S}(\hat{\mathbf{b}}, \hat{h}) \setminus \mathcal{Q}(\hat{\mathbf{b}}, \hat{\alpha}\hat{h})]$$

schreiben. Die rechte Seite der obigen Gleichung ist leer, wenn $\hat{a} \geq 1$, das heißt, dass $\alpha \in (1/d, 1/(d-1)]$. An dieser Stelle liefert die Gleichung (4.10)

$$\begin{aligned} &V_H[\mathcal{S}(\mathbf{b} + \alpha h \mathbf{1}, h - \alpha d h) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)] \\ &\quad = \sum_{\substack{\mathbf{i} \in \{0,1\}^d, \\ 0 < \#\mathbf{i} < 1/\widehat{\alpha}}} (-1)^{\#\mathbf{i}+1} V_H[\mathcal{S}(\hat{\mathbf{b}} + \hat{\alpha}\hat{h}\mathbf{i}, \hat{h}(1 - \#\mathbf{i}\hat{\alpha}))] \\ &\quad = \sum_{\substack{\mathbf{i} \in \{0,1\}^d, \\ 0 < \#\mathbf{i} < d - 1/\alpha}} (-1)^{\#\mathbf{i}+1} V_H[\mathcal{S}(\mathbf{b} + \alpha h(\mathbf{1} - \mathbf{i}), h(1 - \alpha(d - \#\mathbf{i})))]. \end{aligned}$$

Substituieren wir $\hat{\mathbf{i}} = \mathbf{1} - \mathbf{i}$ ($\#\hat{\mathbf{i}} = d - \#\mathbf{i}$) in der obigen Gleichung, so können wir

$$\begin{aligned} &V_H[\mathcal{S}(\mathbf{b} + \alpha h\mathbf{1}, h - \alpha dh) \setminus \mathcal{Q}(\mathbf{b}, \alpha h)] \\ &\quad = \sum_{\substack{\hat{\mathbf{i}} \in \{0,1\}^d, \\ 1/\alpha < \#\hat{\mathbf{i}} < d}} (-1)^{d - \#\hat{\mathbf{i}} + 1} V_H[\mathcal{S}(\mathbf{b} + \alpha h\hat{\mathbf{i}}, h(1 - \#\hat{\mathbf{i}}\alpha))] \end{aligned} \tag{4.12}$$

schreiben. In Übereinstimmung mit dem, was oben festgestellt wurde, ist die letzte Gleichung Null im zuvor genannten Fall, dass $\hat{\alpha} \geq 1$ ist. Unter Verwendung von (4.11) und

$$\mathcal{S}(\mathbf{b} + \alpha h\mathbf{1}, h - \alpha dh) = \mathcal{S}(\mathbf{b} + \alpha h\mathbf{i}_N, h(1 - \#\mathbf{i}_N\alpha)),$$

erhalten wir

$$\begin{aligned} &V_H[\mathcal{Q}(\mathbf{b}, h) \setminus \mathcal{S}(\mathbf{b}, \alpha h)] \\ &\quad = V_H[\mathcal{S}(\mathbf{b} + \alpha h\mathbf{i}_N, h(1 - \#\mathbf{i}_N\alpha))] \\ &\qquad - \sum_{\substack{\hat{\mathbf{i}} \in \{0,1\}^d, \\ 1/\alpha < \#\hat{\mathbf{i}} < d}} (-1)^{d - \#\hat{\mathbf{i}} + 1} V_H[\mathcal{S}(\mathbf{b} + \alpha h\hat{\mathbf{i}}, h(1 - \#\hat{\mathbf{i}}\alpha))] \\ &\quad = \sum_{\substack{\hat{\mathbf{i}} \in \{0,1\}^d, \\ 1/\alpha < \#\hat{\mathbf{i}} < d}} (-1)^{d - \#\hat{\mathbf{i}}} V_H[\mathcal{S}(\mathbf{b} + \alpha h\hat{\mathbf{i}}, h(1 - \#\hat{\mathbf{i}}\alpha))]. \end{aligned} \tag{4.13}$$

Rufen wir uns die Definitionen in (4.2) in Erinnerung und substituieren (4.10) und (4.13) in (4.8), so erhalten wir

$$\begin{aligned} V_H[\mathcal{S}(\mathbf{b}, h)] &= V_H[\mathcal{Q}(\mathbf{b}, \alpha h)] + \sum_{\substack{\mathbf{i} \in \{0,1\}^d, \\ 0 < \#\mathbf{i} < 1/\alpha}} (-1)^{\#\mathbf{i} + 1} V_H[\mathcal{S}(\mathbf{b} + \alpha h\mathbf{i}, h(1 - \#\mathbf{i}\alpha))] \\ &\quad - \sum_{\substack{\hat{\mathbf{i}} \in \{0,1\}^d, \\ 1/\alpha < \#\hat{\mathbf{i}} < d}} (-1)^{d - \#\hat{\mathbf{i}}} V_H[\mathcal{S}(\mathbf{b} + \alpha h\hat{\mathbf{i}}, h(1 - \#\hat{\mathbf{i}}\alpha))] \\ &= V_H[\mathcal{Q}[\mathbf{b}, \alpha h)] + \sum_{j=1}^{N} m^j V_H[\mathcal{S}(\mathbf{b}^j, h^j)]. \end{aligned}$$

□

Als Nächstes definieren wir die Folge $P_n(s)$ als die Summe der V_H-Maße der $\mathcal{Q}_n^k$ multipliziert mit den entsprechenden s_n^k,

$$P_n(s) = P_{n-1} + \sum_{k=1}^{N^{n-1}} s_n^k V_H[\mathcal{Q}_n^k] = \sum_{i=1}^{n} \sum_{k=1}^{N^{i-1}} s_i^k V_H[\mathcal{Q}], \tag{4.14}$$

wobei $P_0(s) = 0$ und die s_n^k definiert sind durch $s_1^1 = 1$ und

$$s_{n+1}^{Nk-N+j} = s_n^k m^j \qquad \text{für alle } j = 1, \dots, N \text{ und } k = 1, \dots, N^{n-1}. \tag{4.15}$$

Wir werden zeigen, dass unter schwachen Annahmen auf H, die Folge $P_n(s)$ gegen $V_H[\mathcal{S}(\mathbf{0}, s)]$ konvergiert. Aus der Gleichung (3.8) können wir $P_n(s)$ direkt berechnen. Die $(N^{n-1}) \times N = N^n$ Simplizes $\mathcal{S}_{n+1}^k$, die durch (4.1) generiert werden, werden in der $(n+1)$-ten Iteration verwendet, um ihre V_H-Maße durch die Maße der Hyperwürfel $\mathcal{Q}_{n+1}^k$ zu approximieren. Um zu zeigen, dass $P_n(s)$ gegen $V_H[\mathcal{S}(\mathbf{0}, s)]$ konvergiert, berechnen wir als Erstes den Approximationsfehler, der entsteht, wenn man $P_n(s)$ statt $V_H[\mathcal{S}(\mathbf{0}, s)]$ verwendet. Für die Theoreme 4.5, 4.6 und 4.7 sei auf [1] verwiesen.

Theorem 4.5. *Mit der oben eingeführten Notation gilt*

$$V_H[\mathcal{S}(\mathbf{0}, s)] - P_n(s) = \sum_{k=1}^{N^n} s_{n+1}^k V_H[\mathcal{S}_{n+1}^k] \qquad \textit{für alle } n \in \mathbb{N}. \tag{4.16}$$

Beweis. Wir beweisen das Theorem durch Induktion nach n. Für $n = 1$ entspricht die Gleichung (4.16) der Gleichung (4.1). Es gilt

$$\begin{aligned} V_H[S_1^1] &= V_H[\mathcal{Q}_1^1] + \sum_{j=1}^{N} m^j V_H[\mathcal{S}_2^{Nk-N+j}] \\ &= P_1(s) + \sum_{k=1}^{N^0} \sum_{j=1}^{N} m^j V_H[\mathcal{S}_2^{Nk-N+j}] \\ &= P_1(s) + \sum_{k=1}^{N} m^j V_H[\mathcal{S}_2^k] = P_1(s) + \sum_{k=1}^{N} s_2^k V_H[\mathcal{S}_2^k], \end{aligned}$$

da $s_2^k = s_1^1 m^j = m^j$. Nun nehmen wir durch Induktion an, dass

$$V_H[\mathcal{S}(\mathbf{0}, s)] = P_{n-1}(s) + \sum_{k=1}^{N^{n-1}} s_n^k V_H[\mathcal{S}_n^k]$$

gilt. Unter Verwendung von (4.1),(4.14) und (4.15) erhalten wir

$$\begin{aligned} V_H[\mathcal{S}(\mathbf{0}, s)] &\overset{(4.1)}{=} P_{n-1}(s) + \sum_{k=1}^{N^{n-1}} s_n^k V_H[\mathcal{Q}_n^k] + \sum_{k=1}^{N^{n-1}} s_n^k \left(\sum_{j=1}^{N} m^j V_H[\mathcal{S}_{n+1}^{Nk-N+j}] \right) \\ &\overset{(4.14)}{=} P_n(s) + \sum_{k=1}^{N^{n-1}} \sum_{j=1}^{N} s_n^k m^j V_H[\mathcal{S}_{n+1}^{Nk-N+j}] \\ &\overset{(4.15)}{=} P_n(s) + \sum_{k=1}^{N^{n-1}} \sum_{j=1}^{N} s_{n+1}^{Nk-N+j} V_H[\mathcal{S}_{n+1}^{Nk-N+j}] \\ &= P_n(s) + \sum_{k=1}^{N^n} s_{n+1}^k V_H[\mathcal{S}_{n+1}^k]. \end{aligned}$$

□

Jetzt sind wir bereit eine hinreichende Bedingung für die Konvergenz der Folge $P_n(s)$ nach $V_H[\mathcal{S}(\mathbf{0}, s)]$ zu geben. Ist das totale Lebesgue-Maß der N neuen Simplizes $\mathcal{S}_{n+1}^{Nk-N+j}$, $j = 1, \ldots, N$, die durch das Simplex $\mathcal{S}_n^k$ generiert werden, kleiner als das Lebesgue-Maß von $\mathcal{S}_n^k$, dann konvergiert der Approximationsfehler (4.16) unter der Annahme, dass H stetig ist, gegen Null. Wir definieren $e_n = \sum_{k=1}^{N^n} \lambda_d[\mathcal{S}_{n+1}^k]$ als die Summe der Lebesgue-Maße der Simplizes, die in der Iteration $n+1$ verwendet werden. Außerdem definieren wir den *Volumenfaktor* $f(\alpha)$ als das Verhältnis der Lebesgue-Maße der Simplizes in zwei aufeinanderfolgenden Iterationen, das ist $f(\alpha) = e_n/e_{n-1}$. Mithilfe der Formel (3.10) für das λ_d-Maß eines Simplex erhalten wir

$$\sum_{j=1}^{N} \lambda_d[\mathcal{S}_{n+1}^{Nk-N+j}] = \sum_{j=1}^{N} \frac{|(1 - \#\mathbf{i}_j\alpha)h_n^k|^d}{d!} = \sum_{j=1}^{d} \binom{d}{j} \frac{|1 - j\alpha|^d |h_n^k|^d}{d!}.$$

Unter Verwendung der obigen Gleichung und mit der Beobachtung, dass die N Simplizes $\mathcal{S}_{n+1}^{Nk-N+j}, j = 1, \ldots, N$ durch das Simplex S_n^k erzeugt werden, erhalten wir

$$\begin{aligned} f(\alpha) = \frac{e_n}{e_{n-1}} &= \frac{\sum_{k=1}^{N^n} \lambda_d[\mathcal{S}_{n+1}^k]}{\sum_{k=1}^{N^{n-1}} \lambda_d[\mathcal{S}_n^k]} \\ &= \frac{\sum_{k=1}^{N^{n-1}} \sum_{j=1}^{N} \lambda_d[\mathcal{S}_{n+1}^{Nk-N+j}]}{\sum_{k=1}^{N^{n-1}} \lambda_d[\mathcal{S}_n^k]} = \frac{\sum_{k=1}^{N^{n-1}} \sum_{j=1}^{d} \binom{d}{j} (|1 - j\alpha|^d |h_n^k|^d / d!)}{\sum_{k=1}^{N^{n-1}} \lambda_d[\mathcal{S}_n^k]} \\ &= \frac{(1/d!) \sum_{k=1}^{N^{n-1}} |h_n^k|^d \sum_{j=1}^{d} \binom{d}{j} |1 - j\alpha|^d}{(1/d!) \sum_{k=1}^{N^{n-1}} |h_n^k|^d} = \sum_{j=1}^{d} \binom{d}{j} |1 - j\alpha|^d. \end{aligned}$$

Eine hinreichende Bedingung für die Konvergenz des AEP-Algorithmus kann mithilfe des Volumenfaktors $f(\alpha)$ ausgedrückt werden. Als Erstes nehmen wir an, dass H absolut stetig ist mit beschränkter Dichte.

Theorem 4.6. *Angenommen V_H hat eine beschränkte Dichte v_H. Wenn der Volumenfaktor $f(\alpha) < 1$ erfüllt, dann gilt*

$$\lim_{n \to \infty} P_n(s) = V_H[\mathcal{S}(\mathbf{0}, s)]. \tag{4.17}$$

Beweis. Da die Dichte v_H von V_H durch eine Konstante $c > 0$ beschränkt ist und unter

Verwendung von (4.16), gilt

$$\begin{aligned}|V_H[\mathcal{S}(\mathbf{0},s)] - P_n(s)| &= \left|\sum_{k=1}^{N^n} s_{n+1}^k V_H[\mathcal{S}_{n+1}^k]\right| = \left|\sum_{k+1}^{N^n}\int_{\mathcal{S}_{n+1}^k} s_{n+1}^k dH\right| \\ &\le \left|\sum_{k+1}^{N^n}\int_{\mathcal{S}_{n+1}^k} s_{n+1}^k c d\lambda_d\right| \le c\sum_{k+1}^{N^n}\int_{\mathcal{S}_{n+1}^k}\left|s_{n+1}^k\right| d\lambda_d \\ &= c\sum_{k+1}^{N^n}\int_{\mathcal{S}_{n+1}^k} d\lambda_d = c\sum_{k=1}^{N^n}\lambda_d[\mathcal{S}_{n+1}^k] = ce_n.\end{aligned}$$

Wir folgern unter den Annahmen, dass $e_n > 0$ und $e_n/e_{n-1} = f(\alpha) < 1$ gelten, dass e_n exponentiell gegen Null konvergiert für n gegen unendlich. □

Damit die Gleichung (4.17) gilt, genügt es, dass v_H auf $\bigcup_{k=1}^{N^n}\mathcal{S}_{n+1}^k$ beschränkt ist für hinreichend großes n. Wir definieren die Hyperebene Γ_s durch

$$\Gamma_s = \left\{(x_1,\ldots,x_d)\in\mathbb{R}^d : \sum_{k=1}^d x_k = s\right\}. \tag{4.18}$$

Das folgende Theorem besagt, dass der L^1-Abstand von der Hyperebene Γ_s zu jedem Punkt in $\bigcup_{k=1}^{N^n}\mathcal{S}_{n+1}^k$ beschränkt durch einen Faktor $\gamma^n s$ ist, wobei $\gamma = \max\{1-\alpha, |1-d\alpha|\}$. Wenn $\alpha\in(0,2/d)$, gilt $\gamma<1$ und der Abstand konvergiert gegen Null für $n\to\infty$. Damit das Theorem 4.6 gilt, wenn $\alpha\in(0,2/d)$, ist es hinreichend, zu verlangen, dass H nur in der Umgebung von Γ_s eine beschränkte Dichte hat.

Theorem 4.7. *Ist $x\in\bigcup_{k=1}^{N^n}\mathcal{S}_{n+1}^k$, dann ist sein L^1-Abstand zur Hyperebene Γ_s beschränkt durch $\gamma^n s$, mit $\gamma = \max\{1-\alpha,|1-d\alpha|\}$.*

Beweis. Wir bezeichnen mit $b_n^{k,r}$ (bzw. i_j^r) für $r\in D$ die d Komponenten des Vektors $\mathbf{b}_n^k$ (bzw. $\mathbf{i}_j$). Wir wollen durch Induktion nach n beweisen, dass

$$\sum_{r=1}^d b_n^{k,r} + h_n^k = s \qquad \text{für alle } k=1,\ldots,N^{n-1} \text{ und } n\ge 1. \tag{4.19}$$

Für $n=1$ ist die Aussage wahr, da nur ein Simplex existiert mit $\mathbf{b}_1^1 = \mathbf{0}$ und $h_1^1 = s$. Angenommen die Aussage gilt für $n>1$. Wegen (4.2) erhalten wir für alle $j=1,\ldots,N$ und $k=1,\ldots,N^{n-1}$, dass

$$\begin{aligned}&\sum_{r=1}^d b_{n+1}^{Nk-N+j,r} + h_{n+1}^{Nk-N+j} \\ &\quad= \sum_{r=1}^d (b_n^{k,r} + \alpha h_n^k i_j^r) + (1-\#\mathbf{i}_j\alpha)h_n^k = \sum_{r=1}^d b_n^{k,r} + \alpha h_n^k\sum_{r=1}^d i_r^j + h_n^k - \alpha h_n^k \#\mathbf{i}_j\end{aligned}$$

$$= \sum_{r=1}^{d} b_n^{k,r} + \alpha h_n^k \# \mathbf{i}_j + h_n^k - \alpha h_n^k \# \mathbf{i}_j = \sum_{r=1}^{d} b_n^{k,r} + h_n^k = s,$$

wobei die letzte Gleichung die Induktionsannahme ist. Wegen (4.19) liegt die diagonale Seitenfläche jedes Simplex $\mathcal{S}_{n+1}^k$, der durch den AEP-Algorithmus generiert wird, auf der Hyperebene Γ_s. Als Konsequenz ist der L^1-Abstand von Γ_s von jedem Punkt in $\mathcal{S}_{n+1}^k$ strikt kleiner als der Abstand vom Vektor $\mathbf{b}_{n+1}^k$, was $|h_{n+1}^k|$ ist. Für ein fixes n und $k = 1, \ldots, N^{n-1}$, erhalten wir, dass $|h_{n+1}^{Nk-N+j}| \leq \gamma |h_n^k|$ für alle $j = 1, \ldots, N$. Daher gilt

$$\max_{k=1,\ldots,N^n} |h_{n+1}^k| = \gamma^n h_1^1 = \gamma^n s, \tag{4.20}$$

wobei die Gleichung für jedes $n \geq 1$ gilt, da $|h_{n+1}^{Nk-N+j}| = \gamma |h_n^k|$ für $j = 1$ oder $j = N$. □

4.2 Beispiel zur Erläuterung des AEP-Algorithmus

In diesem Abschnitt soll der AEP-Algorithmus anhand eines einfachen Beispiels illustriert werden. Die multivariate Verteilungsfunktion H wird mithilfe der Randverteilungen F_{X_i} und einer Copula C dargestellt. Wir betrachten ein zweidimensionales Portfolio ($d = 2$) mit exponentialverteilten Rändern, d.h.

$$F_{X_i}(x) = \mathbb{P}[X_i \leq x] = 1 - e^{-\lambda_i x}, \qquad x \geq 0, i = 1, 2,$$

mit Parametern $\lambda_1 = 1.5$ und $\lambda_2 = 0.5$. Wir verbinden diese exponentialverteilte Ränder durch eine bivariate Clayton Copula $C = C_\delta^{Cl}$ mit

$$C_\delta^{Cl} = (u_1^{-\delta} + u_2^{-\delta} + 1)^{-1/\delta}, \qquad u_k \in [0, 1], k = 1, 2.$$

Als Parameter wählen wir $\delta = 1.2, \alpha = \frac{3}{4}$ und $s = 10$. Mithilfe des Satzes von Sklar können wir die multivariate Verteilungsfunktion H berechnen:

$$\begin{aligned} H(x_1, x_2) &= C_\delta^{Cl}(F_{X_1}(x_1), F_{X_2}(x_2)) \\ &= ((1 - e^{-1.5x_1})^{-1.2} + (1 - e^{-0.5x_2})^{-1.2} - 1)^{-1/1.2}. \end{aligned}$$

In der ersten Iteration, d.h. für $n = 1$, wird das V_H-Maß des Simplex $\mathcal{S}(\mathbf{0}, s)$ durch das V_H-Maß des Hyperwürfels $\mathcal{Q}_1^1 = \mathcal{Q}(\mathbf{0}, \alpha s)$ approximiert, wie in Abbildung 4.1 illustriert wird.

Formal gilt

$$\mathcal{S}(\mathbf{0}, s) = (\mathcal{Q}_1^1 \cup \mathcal{S}_2^1 \cup \mathcal{S}_2^2) \setminus \mathcal{S}_2^3 \qquad \text{für alle } \alpha \in [1/2, 1). \tag{4.21}$$

Da $\alpha \in [1/2, 1)$, sind die Mengen $\mathcal{S}_2^1, \mathcal{S}_2^2$ und $\mathcal{Q}_1^1$ paarweise disjunkt und außerdem gilt $\mathcal{S}_2^3 \subset \mathcal{Q}_1^1$. Das V_H-Maß auf $\mathcal{S}(\mathbf{0}, s)$ kann daher in dieser Weise geschrieben werden

$$V_H[\mathcal{S}(\mathbf{0}, s)] = V_H[\mathcal{Q}_1^1] + V_H[\mathcal{S}_2^1] + V_H[\mathcal{S}_2^2] - V_H[\mathcal{S}_2^3].$$

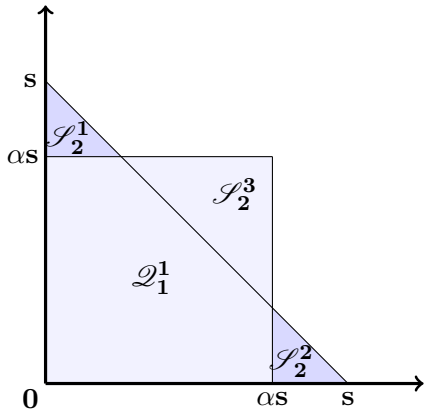

Abbildung 4.1: Zerlegung des zweidimensionalen Simplex $\mathcal{S}(\mathbf{0}, s)$.

Mit der Notation $s_2^1 = s_2^2 = 1$ und $s_2^3 = -1$ kann man die Gleichung oben wie folgt umschreiben

$$V_H[\mathcal{S}(\mathbf{0}, s)] = V_H[\mathcal{Q}_1^1] + \sum_{k=1}^{3} s_2^k V_H[\mathcal{S}_2^k]. \tag{4.22}$$

Unter Verwendung von (3.8) ist die erste Approximation von $V_H[\mathcal{S}(\mathbf{0}, 10)]$ gegeben durch den Wert

$$\begin{aligned} P_1(10) = V_H[\mathcal{Q}_1^1] &= \sum_{j=0}^{3} (-1)^{2-\#\mathbf{i}_j} H(\mathbf{0} + 0.75\mathbf{i}_j) \\ &= H(0,0) - H(7.5,0) - H(0,7.5) + H(7.5,7.5) \\ &= 0 - 0 - 0 + 0.97647 = 0.97647 \end{aligned}$$

mit $\mathbf{i}_0 = (0,0), \mathbf{i}_1 = (1,0), \mathbf{i}_2 = (0,1)$ und $\mathbf{i}_3 = (1,1)$.

Als nächstes berechnen wir den Input für die zweite Iteration. Es gilt $b_2^j = b_1^1 + \alpha \mathbf{i}_j$ und $h_2^j = (1 - \#\mathbf{i}_j \alpha) h_1^1$ für $j = 1, 2, 3$. Daraus ergeben sich folgende Werte

$$\begin{aligned} &\mathbf{b}_2^1 = (0,0) + 0.75 \cdot 10 \cdot (1,0) = (7.5, 0), \qquad \mathbf{b}_2^2 = (0,0) + 0.75 \cdot 10 \cdot (0,1) = (0, 7.5), \\ &\mathbf{b}_2^3 = (0,0) + 0.75 \cdot 10 \cdot (1,1) = (7.5, 7.5), \\ &h_2^1 = h_2^2 = (1 - 1 \cdot 0.75) \cdot 10 = 2.5 \quad \text{und} \quad h_2^3 = (1 - 2 \cdot 0.75) \cdot 10 = -5 \end{aligned}$$

sowie die drei Simplizes

$$\begin{aligned} &\mathcal{S}_2^1 = \mathcal{S}((7.5, 0), 2.5), \qquad \mathcal{S}_2^2 = \mathcal{S}((0, 7.5), 2.5) \quad \text{und} \\ &\mathcal{S}_2^3 = \mathcal{S}((7.5, 7.5), -5). \end{aligned}$$

Die Simplizes werden durch diese drei Hyperwürfel zerlegt

$$\mathcal{Q}_2^1 = \mathcal{Q}((7.5, 0), 1.875), \qquad \mathcal{Q}_2^2 = \mathcal{Q}((0, 7.5), 1.875) \quad \text{und}$$
$$\mathcal{Q}_2^3 = \mathcal{Q}((7.5, 7.5), -3.75).$$

Bei der zweiten Iteration erhalten wir für das V_H-Maß des Simplex $V_H[\mathcal{S}(\mathbf{0}, 10)]$ folgende Approximation

$$\begin{aligned} P_2(10) &= P_1(10) + \sum_{k=1}^{3} s_2^k V_H[\mathcal{Q}_2^k] \\ &= P_1(10) + V_H[\mathcal{Q}_2^1] + V_H[\mathcal{Q}_2^2] - V_H[\mathcal{Q}_2^3] \end{aligned}$$

mit $s_2^1 = s_2^2 = 1$ und $s_2^3 = -1$, wobei die V_H-Maße der Hyperwürfel wie folgt berechnet werden

$$\begin{aligned} V_H[\mathcal{Q}_2^1] &= H(7.5, 0) - H(9.375, 0) - H(7.5, 1.875) + H(9.375, 1.875) \\ &= 0 - 0 - 0.608390 + 0.608394 = 4.09732e\text{-}06, \end{aligned}$$

$$\begin{aligned} V_H[\mathcal{Q}_2^2] &= H(0, 7.5) - H(1.875, 7.5) - H(0, 9.375) + H(1.875, 9.375) \\ &= 0 - 0,919385 - 0 + 0,931903 = 0,012518 \qquad \text{und} \end{aligned}$$

$$\begin{aligned} V_H[\mathcal{Q}_2^3] &= H(7.5, 7.5) - H(3.75, 7.5) - H(7.5, 3.75) + H(3.75, 3.75) \\ &= 0.97647 - 0.97306 - 0.846636 + 0,84414 = 0.000917. \end{aligned}$$

Daraus folgt

$$P_2(10) = 0.97647 + 4.09732e\text{-}06 + 0,012518 - 0.000917 = 0.988074$$

Für die dritte Iteration gilt $\mathbf{b}_3^{Nk-N+j} = \mathbf{b}_2^k + \alpha h_2^k \mathbf{i}_j$ und $h_3^{Nk-N+j} = (1 - \#\mathbf{i}\alpha) h_2^k$ für $N = 3$, $j = 1, 2, 3$ und $k = 1, \ldots, 9$. Daraus erhalten wir

$$\begin{aligned} \mathbf{b}_3^1 &= (7.5, 0) + 0.75 \cdot 2.5 \cdot (1, 0) = (9.375, 0), \\ \mathbf{b}_3^2 &= (7.5, 0) + 0.75 \cdot 2.5 \cdot (0, 1) = (7.5, 1.875), \\ \mathbf{b}_3^3 &= (7.5, 0) + 0.75 \cdot 2.5 \cdot (1, 1) = (9.375, 1.875), \\ \mathbf{b}_3^4 &= (0, 7.5) + 0.75 \cdot 2.5 \cdot (1, 0) = (1.875, 7.5), \\ \mathbf{b}_3^5 &= (0, 7.5) + 0.75 \cdot 2.5 \cdot (0, 1) = (0, 9.375), \\ \mathbf{b}_3^6 &= (0, 7.5) + 0.75 \cdot 2.5 \cdot (1, 1) = (1.875, 9.375), \\ \mathbf{b}_3^7 &= (7.5, 7.5) - 0.75 \cdot 5 \cdot (1, 0) = (3.75, 7.5), \\ \mathbf{b}_3^8 &= (7.5, 7.5) - 0.75 \cdot 5 \cdot (0, 1) = (7.5, 3.75), \\ \mathbf{b}_3^9 &= (7.5, 7.5) - 0.75 \cdot 5 \cdot (1, 1) = (3.75, 3.75), \end{aligned}$$

$$
\begin{aligned}
&h_3^1 = h_3^2 = h_3^4 = h_3^5 = (1 - 1 \cdot 0.75) \cdot 2.5 = 0.625,\\
&h_3^3 = h_3^6 = (1 - 2 \cdot 0.75) \cdot 2.5 = -1.25\\
&h_3^7 = h_3^8 = (1 - 1 \cdot 0.75) \cdot (-5) = -1.25, \quad \text{und}\\
&h_3^9 = (1 - 2 \cdot 0.75) \cdot (-5) = 2.5
\end{aligned}
$$

sowie die neun Simplizes

$$
\begin{aligned}
&\mathcal{S}_3^1 = \mathcal{S}((9.375, 0), 0.625), && \mathcal{S}_3^2 = \mathcal{S}((7.5, 1.875), 0.625),\\
&\mathcal{S}_3^3 = \mathcal{S}((9.375, 1.875), -1.25), && \mathcal{S}_3^4 = \mathcal{S}((1.875, 7.5), 0.625),\\
&\mathcal{S}_3^5 = \mathcal{S}((0, 9.375), 0.625), && \mathcal{S}_3^6 = \mathcal{S}((1.875, 9.375), -1.25),\\
&\mathcal{S}_3^7 = \mathcal{S}((3.75, 7.5), -1.25), && \mathcal{S}_3^8 = \mathcal{S}((7.5, 3.75), -1.25) \quad \text{und}\\
&\mathcal{S}_3^9 = \mathcal{S}((3.75, 3.75), 2.5).
\end{aligned}
$$

Diese neun Simplizes zerlegen wir wiederum durch neun Hyperwürfel

$$
\begin{aligned}
&\mathcal{Q}_3^1 = \mathcal{Q}((9.375, 0), 0.46875), && \mathcal{Q}_3^2 = \mathcal{Q}((7.5, 1.875), 0.46875),\\
&\mathcal{Q}_3^3 = \mathcal{Q}((9.375, 1.875), -0.9375), && \mathcal{Q}_3^4 = \mathcal{Q}((1.875, 7.5), 0.46875),\\
&\mathcal{Q}_3^5 = \mathcal{Q}((0, 9.375), 0.46875), && \mathcal{Q}_3^6 = \mathcal{Q}((1.875, 9.375), -0.9375),\\
&\mathcal{Q}_3^7 = \mathcal{Q}((3.75, 7.5), -0.9375), && \mathcal{Q}_3^8 = \mathcal{Q}((7 - 5, 3.75), -0.9375) \quad \text{und}\\
&\mathcal{Q}_3^9 = \mathcal{Q}((3.75, 3.75), 1.875)
\end{aligned}
$$

und erhalten dadurch folgende Approximation für das V_H-Maß des Simplex $V_H[\mathcal{S}(\mathbf{0}, 10)]$:

$$
\begin{aligned}
P_3(10) &= P_2(10) + \sum_{k=1}^{9} s_3^k V_H[\mathcal{Q}_3^k]\\
&= P_2(10) + V_H[\mathcal{Q}_3^1] + V_H[\mathcal{Q}_3^2] - V_H[\mathcal{Q}_3^3] + V_H[\mathcal{Q}_3^4] +\\
&\quad V_H[\mathcal{Q}_3^5] - V_H[\mathcal{Q}_3^6] - V_H[\mathcal{Q}_3^7] - V_H[\mathcal{Q}_3^8] + V_H[\mathcal{Q}_3^9]\\
&= 0.988074 + 1.25898e\text{-}08 + 7.04268e\text{-}07 - 5.29607e\text{-}07 + 3.03579e\text{-}04 +\\
&\quad 4.32361e\text{-}04 - 1.82521e\text{-}03 - 3.27927e\text{-}04 - 6.19326e\text{-}06 + 6.06876e\text{-}04\\
&= 0.987258
\end{aligned}
$$

mit $s_3^1 = s_3^2 = s_3^4 = s_3^5 = s_3^9 = 1$ und $s_3^3 = s_3^6 = s_3^7 = s_3^8 = -1$.

Als Vergleich berechnen wir $V_H[\mathcal{S}(\mathbf{0}, 10)]$ durch Integration der gemeinsamen Dichtefunktion

$$
V_H[\mathcal{S}(\mathbf{0}, 10)] = \int_0^{10} \int_0^{10-x_1} h(x_1, x_2) dx_1 dx_2 = 0.98761245, \tag{4.23}
$$

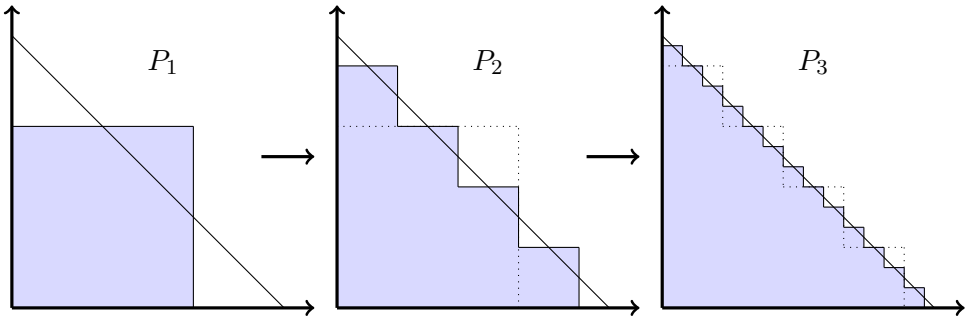

Abbildung 4.2: Graphische Darstellung von P_1, P_2 und P_3.

wobei die Dichtefunktion gegeben ist durch

$$h(x_1, x_2) = (1.2 + 1)\left(\left(1 - e^{-1.5x_1}\right)^{-1.2} + \left(1 - e^{-0.5x_2}\right)^{-1.2} - 1\right)^{-2-1/1.2} \left(1 - e^{-1.5x_1}\right)^{-2.2} 1.5e^{-1.5x_1} \left(1 - e^{-0.5x_2}\right)^{-2.2} 0.5e^{-0.5x_2}.$$

Für die Berechnung des Integrals (4.23) wurde die Funktion `adaptIntegrate` aus dem R-Package `cubature` verwendet. Diese wird ausführlich in Kapitel 5.2 beschrieben.

Wir können sehen, dass die Approximation $P_3(10) = 0.987258$ von $V_H[\mathcal{S}(\mathbf{0}, 10)]$ durch den AEP-Algorithmus bereits nach drei Iterationen bis zur dritten Nachkommastelle genau ist.

4.3 Wahl des Teilungsparameters α

Der AEP-Algorithmus hängt von der Wahl des Parameters α ab. Im Allgemeinen würde die optimale Wahl von α vom Maß V_H abhängen. Im Beweis von Theorem 3.2 haben wir gezeigt, dass

$$|P_n(s) - V_H[\mathcal{S}(\mathbf{0}, s)]| \leq cf(\alpha)^n,$$

gilt, wobei c eine positive Konstante ist. Da der Algorithmus unabhängig von der Wahl der Verteilung H sein soll, verwenden wir ein α^*, welches $f(\alpha)$ minimiert, d.h.

$$\alpha^* = \underset{\alpha \in [1/d, 1)}{\arg\min} f(\alpha) = \frac{2}{d+1}.$$

Wir werden zeigen, dass die Verwendung von α^* mehrere wünschenswerte Konsequenzen hat. Ist $\alpha = \alpha^*$ und die Dimension d ungerade, dann ist bei der Aufspaltung (4.1) für eine Anzahl von $\binom{d}{(d+1)/2}$ Simplizes der entsprechende Koeffizient m^j gleich Null und diese Simplizes können vernachlässigt werden. Dadurch wird der AEP-Algorithmus effizienter. Zum Beispiel generiert der Algorithmus bei der Aufspaltung eines dreidimensionalen Simplex nur vier neue Simplizes bei jeder Iteration mit $\alpha = \alpha^*$, anstatt

von $2^d - 1 = 7$ generierten Simplizes bei jedem anderen möglichen α. Für $\alpha = \alpha^*$ ist die Anzahl der neuen Simplizes, die bei jedem Schritt generiert werden, gegeben durch die Funktion

$$f_S(d) = \begin{cases} 2^d - 1, & \text{wenn } d \text{ gerade ist,} \\ 2^d - 1 - \binom{d}{(d+1)/2}, & \text{wenn } d \text{ ungerade ist.} \end{cases} \tag{4.24}$$

Da

$$(0, +\infty)^d \cap \left(\bigcup_{k=1}^{N^{n-1}} \mathcal{S}_n^k \right) \subset \mathcal{S}(\mathbf{0}, (1+\gamma^n)s) \backslash \mathcal{S}(\mathbf{0}, (1-\gamma^n)s) \tag{4.25}$$

gilt, ist die Wahl $\alpha = \alpha^*$ günstig. Gilt $\alpha = \alpha^* \in (0, 2/d)$, so erhalten wir, dass $\gamma < 1$ und $\gamma^n s$ konvergiert gegen Null für $n \to \infty$.

d	α^*	$f(\alpha^*)$	d	α^*	$f(\alpha^*)$
2	$\frac{2}{3}$	$\frac{1}{3}$	5	$\frac{1}{3}$	$\frac{23}{27}$
3	$\frac{1}{2}$	$\frac{1}{2}$	6	$\frac{2}{7}$	> 1
4	$\frac{2}{5}$	$\frac{83}{125}$	7	$\frac{1}{4}$	> 1

Tabelle 4.1: Werte von α^* und $f(\alpha^*)$ für Dimensionen $d \leq 7$.

Um die Konvergenz der Folge P_n zu garantieren, ist es hinreichend vorauszusetzen, dass die Verteilung H nur in der Umgebung von Γ_s eine beschränkte Dichte hat. Unter dieser Annahme muss die Summe $\mathcal{S}_d$ beim Wert s, bei dem die Verteilung berechnet wird, stetig sein, d.h. $\mathbb{P}[\mathcal{S}_d = s] = 0$. Gilt stattdessen $V_H[\Gamma_s] > 0$ wird möglicherweise der Algorithmus nicht konvergieren.

4.3.1 Ein Gegenbeispiel

Seien zum Beispiel X_1 und X_2 zwei Zufallsvariablen mit $\mathbb{P}[X_1 = 1/2] = \mathbb{P}[X_2 = 1/2] = 1$ und ihre Verteilungsfunktion ist gegeben durch $F(x) = \mathbb{1}_{\{x \geqslant \frac{1}{2}\}}(x)$. Wir berechnen mithilfe einer Clayton Copula die multivariate Verteilung $H(x_1, x_2) = (F(x_1)^{-\delta} + F(x_2)^{-\delta} - 1)^{-\frac{1}{\delta}}$ für $0 < \delta < \infty$. Sei $s = 1$ und $\alpha \in [1/d, 1)$. Dann erhalten wir für die erste Iteration:

$$P_1(1) = V_H[\mathcal{Q}_1^1] = H(0,0) - H(\alpha, 0) - H(0, \alpha) + H(\alpha, \alpha) = 1$$

und

$$\begin{aligned} \mathbf{b}_2^1 &= (\alpha, 0), & \mathbf{b}_2^2 &= (0, \alpha), & \mathbf{b}_2^3 &= (\alpha, \alpha) \\ h_2^1 &= 1 - \alpha, & h_2^2 &= 1 - \alpha, & h_2^3 &= 1 - 2\alpha \\ s_2^1 &= 1, & s_2^2 &= 1, & s_2^3 &= -1. \end{aligned}$$

Für $n = 2$ ist die zweite Approximation gegeben durch:

$$P_2(1) = P_1(1) + V_H[\mathcal{Q}_2^1] + V_H[\mathcal{Q}_2^2] - V_H[\mathcal{Q}_2^3]$$

mit

$$\begin{aligned}
V_H[\mathcal{Q}_2^1] &= H(\alpha, 0) - H(\alpha(2-\alpha), 0) - H(\alpha, \alpha(1-\alpha)) + H(\alpha(2-\alpha), \alpha(1-\alpha)) \\
&= 0 - 0 - 0 + 0, \\
V_H[\mathcal{Q}_2^2] &= H(0, \alpha) - H(\alpha(1-\alpha), \alpha) - H(0, \alpha(2-\alpha)) + H(\alpha(1-\alpha), \alpha(2-\alpha)) \\
&= 0 - 0 - 0 + 0, \\
V_H[\mathcal{Q}_2^3] &= H(\alpha, \alpha) - H(2\alpha(1-\alpha), \alpha) - H(\alpha, 2\alpha(1-\alpha)) + H(2\alpha(1-\alpha), 2\alpha(1-\alpha)) \\
&= 1 - 0 - 0 + 0.
\end{aligned}$$

Daraus folgt, dass $P_2(1) = 1 + 0 + 0 - 1 = 0$.

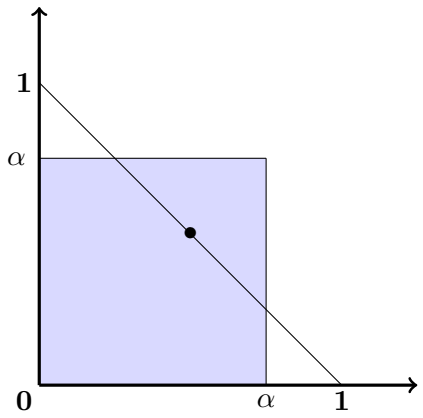

Abbildung 4.3: Gegenbeispiel zur Konvergenz der Folge P_n.

Obwohl $V_H[\mathcal{S}(\mathbf{0}, 1)] = 1$ gilt, alterniert die Folge $P_n(1)$ zwischen 0 und 1. Wenn H zumindest eine beschränkte Dichte in der Nähe von Γ_s hat, dann ist die Konvergenz der Folge $P_n(s)$ gegen den Wert $V_H[\mathcal{S}(\mathbf{0}, s)]$ garantiert.

Eine weitere relevante Frage ist die Tatsache, dass der erforderliche Speicher für die n-te Iteration exponentiell in n anwächst. Bei jeder Iteration des Algorithmus erzeugt jedes Simplex $\mathcal{S}_n^k$ einen Hyperwürfel und eine Anzahl von $f_S(d)$ neuen Simplizes, die an die nächste Iteration weitergegeben werden. Die Wahl von $\alpha = \alpha^*$ ermöglicht es die Genauigkeit des AEP-Algorithmus zu steigern.

4.4 Verbesserung der numerischen Genauigkeit des Algorithmus durch Extrapolation

In diesem Abschnitt führen wir eine Methode zur Erhöhung der Genauigkeit des AEP-Algorithmus basierend auf der Wahl $\alpha = \alpha^*$ ein. Für den weiteren Verlauf machen wir die stärkere Annahme, dass die multivariate Verteilung H eine zweimal stetig differenzierbare Dichte v_H hat, deren Ableitungen beschränkt sind. Das erlaubt uns die Dichte v_H durch ihre lineare Taylorentwicklung zu approximieren. Dies liefert uns eine gute Schätzung für den Näherungsfehler des AEP-Algorithmus. Als Erstes brauchen wir zwei einfache Integrationsergebnisse. Sei S_{d-1} ein Simplex mit Dimension $(d-1)$, für alle $s > 0$ gilt

$$\begin{aligned}\int_{\mathcal{S}(\mathbf{0},s)} x_d d\mathbf{x} &= \int_0^s \int_0^{s-x_d} \dots \int_0^{s-\sum_{k=3}^d x_k} \int_0^{s-\sum_{k=2}^d x_k} x_d d\mathbf{x} \\ &= \int_0^s x_d \int_0^{s-x_d} \dots \int_0^{s-\sum_{k=3}^d x_k} \int_0^{s-\sum_{k=2}^d x_k} d\mathbf{x} \\ &= \int_0^s x_d \lambda_{d-1}[\mathcal{S}_{d-1}(\mathbf{0}, s-x_d)] dx_d = \int_0^s x_d \frac{(s-x_d)^{d-1}}{(d-1)!} dx_d = \frac{s^{d+1}}{(d+1)!}.\end{aligned}$$

Analog gilt für alle $s > 0$

$$\begin{aligned}\int_{\mathcal{Q}(\mathbf{0},\alpha s)} x_d d\mathbf{x} &= \int_0^{\alpha s} \int_0^{\alpha s} \dots \int_0^{\alpha s} x_d d\mathbf{x} \\ &= \int_0^{\alpha s} x_d \int_0^{\alpha s} \dots \int_0^{\alpha s} x_d d\mathbf{x} = (\alpha s)^{d-1} \int_0^{\alpha s} x_d dx_d = 1/2(\alpha s)^{d+1}.\end{aligned}$$

Nun berechnen wir das V_H-Maß eines Hyperwürfels und eines Simplex für den einfachen Fall, dass die Verteilung H eine lineare Dichte hat, d.h. $v_H(\mathbf{b}+\mathbf{x}) = a + \sum_{k=1}^d c_k x_k$ für $\mathbf{x} \in \mathcal{S}(\mathbf{0}, s) \cup \mathcal{Q}(\mathbf{0}, \alpha s)$. Für alle $s > 0$ erhalten wir

$$\begin{aligned}V_H[\mathcal{S}(\mathbf{b}, s)] &= a \int_{\mathcal{S}(\mathbf{0},s)} d\mathbf{x} + \sum_{k=1}^d c_k \int_{\mathcal{S}(\mathbf{0},s)} x_k d\mathbf{x} \\ &= a\frac{s^d}{d!} + \frac{s^{d+1}}{(d+1)!}\left(\sum_{k=1}^d c_k\right) = \frac{s^d}{d!}\left(a + \frac{s}{d+1}\sum_{k=1}^d c_k\right),\end{aligned} \tag{4.26}$$

$$\begin{aligned}V_H[\mathcal{Q}(\mathbf{b}, \alpha s)] &= a \int_{\mathcal{Q}(\mathbf{0}\alpha,s)} d\mathbf{x} + \sum_{k=1}^d c_k \int_{\mathcal{Q}(\mathbf{0},\alpha s)} x_k d\mathbf{x} \\ &= a(\alpha s)^d + \frac{1}{2}\left(\sum_{k=1}^d c_k\right)(\alpha s)^{d+1} = (\alpha s)^d \left(a + \frac{1}{2}\alpha s \sum_{k=1}^d c_k\right).\end{aligned} \tag{4.27}$$

Daher kann für eine lineare Dichte v_H der Bruch $V_H[\mathcal{S}(\mathbf{b}, s)]/V_H[\mathcal{Q}(\mathbf{b}, \alpha s)]$ durch die Wahl $\alpha = \alpha^* = \frac{2}{d+1}$ unabhängig von den Parametern $\mathbf{b}, s, a$ und c_k's gemacht werden. Dadurch erhalten wir

$$V_H[\mathcal{S}(\mathbf{b}, s)] = \frac{(d+1)^d}{2^d d!} V_H[\mathcal{Q}(\mathbf{b}, \alpha^* s)]. \tag{4.28}$$

Das folgende Theorem aus [1] zeigt, dass (4.28) für jede hinreichend glatte Dichte gilt, wenn die Anzahl der Iterationen n des AEP-Algorithmus gegen unendlich geht.

Theorem 4.8. *Angenommen H hat eine zweimal stetig differenzierbare Dichte v_H, wobei alle partielle Ableitungen erster und zweiter Ordnung beschränkt durch eine Konstante D sind. Dann gilt*

$$\sup_{n\in\mathbb{N}} \max_{k=1,\dots,N^{n-1}} \frac{1}{|h_n^k|^{d+2}} \left| V_H[\mathcal{S}(\mathbf{b}_n^k, h_n^k)] - \frac{(d+1)^d}{2^d d!} V_H[\mathcal{Q}(\mathbf{b}_n^k, \alpha^* h_n^k)] \right| \leq A < \infty \tag{4.29}$$

für eine positive Konstante A abhängig nur von der Dimension d und der Verteilung H.

Beweis. Für ein bestimmtes $\mathbf{b}_n^k$ verwenden wir eine Taylorentwicklung, um gewisse Koeffizienten a und $c_k, k = 1, \dots, d$ abhängig von $\mathbf{b}_n^k$ zu finden, so dass

$$v_H(\mathbf{b}_n^k + x) = a + \sum_{k=1}^{d} c_k x_k + \sum_{|\beta|=2} R_\beta(\mathbf{x}) \mathbf{x}^\beta \qquad \text{für alle } \mathbf{x} \in \mathcal{B}(\mathbf{b}_n^k), \tag{4.30}$$

wobei $\mathcal{B}(\mathbf{b}_n^k)$ eine Kugel in $\mathbb{R}^d$ ist mit $\mathbf{b}_n^k$ als Zentrum, so dass $\mathcal{B}(\mathbf{b}_n^k) \supset \mathcal{S}(\mathbf{b}_n^k, h_n^k) \cup \mathcal{Q}(\mathbf{b}_n^k, \alpha^* h_n^k)$ gilt. In (4.30) haben wir eine multi Index Notation verwendet, um zu zeigen, dass sich die Summe in der letzten Gleichung über die multi Indices $\beta \in \mathbb{N}^d$ erstreckt. Unter Verwendung der Annahme über die partiellen Ableitungen von v_H erfüllt das Restglied $R_\beta(x)$ die Ungleichung

$$|R_\beta(\mathbf{x})| \leq \sup_{\mathbf{x}\in\mathcal{B}(\mathbf{b_n^k})} \left| \frac{1}{\beta!} \frac{\partial^\beta v_H(\mathbf{x})}{\partial \mathbf{x}^\beta} \right| \leq D \tag{4.31}$$

für alle β mit $|\beta| = 2$. Unter Verwendung von (4.30), (4.26) und (4.27) für eine lineare Dichte und einem positiven h_n^k erhalten wir

$$\left| V_H[\mathcal{S}(\mathbf{b}_n^k, h_n^k)] - \frac{(d+1)^d}{2^d d!} V_H[\mathcal{Q}(\mathbf{b}_n^k, \alpha h_n^k)] \right|$$

$$= \left| \frac{(h_n^k)^d}{d!} \left(a + \frac{h_n^k}{d+1} \sum_{k=1}^{d} c_k \right) + \int_{\mathcal{S}(\mathbf{0}, h_n^k)} \sum_{|\beta|=2} R_\beta(\mathbf{x}) \mathbf{x}^\beta d\mathbf{x} \right.$$

$$-\frac{(d+1)^d}{2^d d!}\left((\alpha h_n^k)^d\left(a+\frac{1}{2}\alpha h_n^k\sum_{k=1}^{d}c_k\right)+\int_{\mathcal{Q}(\mathbf{0},\alpha h_n^k)}\sum_{|\beta|=2}R_\beta(\mathbf{x})\mathbf{x}^\beta d\mathbf{x}\right)\Bigg|.$$

Durch die Wahl $\alpha=\alpha^*$ vereinfacht sich der vorige Ausdruck zu

$$\left|V_H[\mathcal{S}(\mathbf{b}_n^k,h_n^k)]-\frac{(d+1)^d}{2^d d!}V_H[\mathcal{Q}(\mathbf{b}_n^k,\alpha^* h_n^k)]\right|$$

$$=\left|\int_{\mathcal{S}(\mathbf{0},h_n^k)}\sum_{|\beta|=2}R_\beta(\mathbf{x})\mathbf{x}^\beta d\mathbf{x}-\frac{(d+1)^d}{2^d d!}\int_{\mathcal{Q}(\mathbf{0},\alpha^* h_n^k)}\sum_{|\beta|=2}R_\beta(\mathbf{x})\mathbf{x}^\beta d\mathbf{x}\right|$$

$$\leq\left|\sum_{|\beta|=2}\int_{\mathcal{S}(\mathbf{0},h_n^k)}R_\beta(\mathbf{x})\mathbf{x}^\beta d\mathbf{x}\right|+\frac{(d+1)^d}{2^d d!}\left|\sum_{|\beta|=2}\int_{\mathcal{Q}(\mathbf{0},\alpha^* h_n^k)}R_\beta(\mathbf{x})\mathbf{x}^\beta d\mathbf{x}\right|$$

$$\leq D\left(\left|\sum_{|\beta|=2}\int_{\mathcal{S}(\mathbf{0},h_n^k)}\mathbf{x}^\beta d\mathbf{x}\right|+\frac{(d+1)^d}{2^d d!}\left|\sum_{|\beta|=2}\int_{\mathcal{Q}(\mathbf{0},\alpha^* h_n^k)}\mathbf{x}^\beta d\mathbf{x}\right|\right),$$

wobei die Ungleichung aus (4.31) folgt. Mithilfe der folgenden Gleichungen

$$\begin{aligned}\sum_{|\beta|=2}\int_{\mathcal{S}(\mathbf{0},s)}x^\beta d\mathbf{x}&=\sum_{i=1}^{d}\int_{\mathcal{S}(\mathbf{0},s)}x_i^2 d\mathbf{x}+2\sum_{1\leq i\leq j\leq d}\int_{\mathcal{S}(\mathbf{0},s)}x_i x_j d\mathbf{x}\\&=\frac{2ds^{d+2}}{(d+2)!}+\frac{2d(d-1)s^{d+2}}{(d+2)!}\\&=\frac{2d^2 s^{d+2}}{(d+2)!}\end{aligned}$$

und

$$\begin{aligned}\sum_{|\beta|=2}\int_{\mathcal{Q}(\mathbf{0},\alpha s)}x^\beta d\mathbf{x}&=\sum_{i=1}^{d}\int_{\mathcal{Q}(\mathbf{0},\alpha s)}x_i^2 d\mathbf{x}+2\sum_{1\leq i\leq j\leq d}\int_{\mathcal{Q}(\mathbf{0},\alpha s)}x_i x_j d\mathbf{x}\\&=\frac{d(\alpha s)^{d+2}}{3}+\frac{2d(d-1)(\alpha s)^{d+2}}{4}=\frac{d(3d-1)(\alpha s)^{d+2}}{6}\end{aligned}$$

erhalten wir schließlich

$$\left|V_H[\mathcal{S}(\mathbf{b}_n^k,h_n^k)]-\frac{(d+1)^d}{2^d d!}V_H[\mathcal{Q}(\mathbf{b}_n^k,\alpha^* h_n^k)]\right|\leq A|h_n^k|^{d+2},\tag{4.32}$$

wobei A eine positive Konstante ist, die nur von der Dimension d und der Verteilung H abhängt. In (4.32) verwenden wir den Absolutbetrag von h_n^k, damit wir den analogen Fall betrachten können bei dem h_n^k negativ ist. Daher folgt das Theorem leicht aus (4.32). □

Die Gleichung (4.29) gibt einen lokalen Schätzer für das Volumen des Simplex $\mathcal{S}(\mathbf{b}_n^k, h_n^k)$ in Bezug auf das Volumen des dazugehörigen Hyperwürfels $\mathcal{S}(\mathbf{b}_n^k, h_n^k)$, was einfach zu berechnen ist:

$$V_H[\mathcal{S}(\mathbf{b}_n^k, h_n^k)] \approx \frac{(d+1)^d}{2^d d!} V_H\left[\mathcal{Q}\left(\mathbf{b}_n^k, \frac{2h_n^k}{d+1}\right)\right]. \tag{4.33}$$

Ist die Dichte v_H hinreichend glatt, ist es möglich nach einer Anzahl von Iterationen des AEP-Algorithmus die rechte Seite von (4.16) unter Verwendung der Approximation (4.33) abzuschätzen. Durch diese Vorgehensweise wird der Schätzer $P_n^*(s)$ wie folgt definiert

$$P_n^*(s) = P_{n-1}(s) + \frac{(d+1)^d}{2^d d!} \sum_{k=1}^{N^{n-1}} s_n^k V_H[\mathcal{Q}_n^k]. \tag{4.34}$$

Im Folgenden wird die Verwendung von $P_n^*(s)$, als Approximation von $V_H[\mathcal{S}(\mathbf{0}, s)]$, als *Extrapolationsverfahren* bezeichnet. Das folgende Theorem aus [1] zeigt, dass $P_n^*(s)$ in höheren Dimension schneller gegen $V_H[\mathcal{S}(\mathbf{0}, s)]$ konvergiert als $P_n(s)$.

Theorem 4.9. *Unter den Annahmen von Theorem 4.8 gilt für $d \leq 8$, dass*

$$\lim_{n \to +\infty} P_n^*(s) = V_H[\mathcal{S}(\mathbf{0}, s)].$$

Beweis. Unter Verwendung von (4.16) und (4.32) in der Definition (4.34) von $P_n^*(s)$, erhalten wir

$$\begin{aligned} E^*(n) &= |V_H[\mathcal{S}(\mathbf{0}, s)] - P_n^*(s)| \\ &= \left| V_H[\mathcal{S}(\mathbf{0}, s)] - P_{n-1}(s) - \frac{(d+1)^d}{2^d d!} \sum_{k=1}^{N^{n-1}} s_n^k V_H[\mathcal{Q}_n^k] \right| \\ &= \left| \sum_{k=1}^{N^{n-1}} s_n^k V_H[\mathcal{S}_n^k] - \frac{(d+1)^d}{2^d d!} \sum_{k=1}^{N^{n-1}} s_n^k V_H[\mathcal{Q}_n^k] \right| \\ &\leq \sum_{k=1}^{N^{n-1}} \left| V_H[\mathcal{S}_n^k] - \frac{(d+1)^d}{2^d d!} V_H[\mathcal{Q}_n^k] \right| \leq A \sum_{k=1}^{N^{n-1}} |h_n^k|^{d+2} = A e_{n-1}^*, \end{aligned} \tag{4.35}$$

wobei für die positive Folge $e_n^* = \sum_{k=1}^{N^n} |h_{n+1}^k|^{d+2}$ Folgendes gilt

$$\frac{e_n^*}{e_{n-1}^*} = \frac{\sum_{k=1}^{N^{n-1}} \sum_{j=1}^{N} |h_{n+1}^{Nk-N+j}|^{d+2}}{\sum_{k=1}^{N^{n-1}} |h_n^k|^{d+2}} = \frac{\sum_{k=1}^{N^{n-1}} \sum_{j=1}^{d} \binom{d}{j} |1 - j\alpha^*|^{d+2} |h_n^k|^{d+2}}{\sum_{k=1}^{N^{n-1}} |h_n^k|^{d+2}}$$
$$= \frac{\sum_{k=1}^{N^{n-1}} |h_n^k|^{d+2} \sum_{j=1}^{d} \binom{d}{j} |1 - j\alpha^*|^{d+2}}{\sum_{k=1}^{N^{n-1}} |h_n^k|^{d+2}} = \sum_{j=1}^{d} \binom{d}{j} |1 - j\alpha^*|^{d+2}.$$

Das Theorem folgt daraus, dass $f_*(d)$, definiert durch

$$f_*(d) = \sum_{j=1}^{d} \binom{d}{j} |1 - j\alpha^*|^{d+2}, \tag{4.36}$$

kleiner als 1 ist für $d \leq 8$. In diesen Dimensionen konvergiert e_n^* und daher $E^*(n)$ gegen Null. □

Aufgrund von Theorem 3.3 bleibt Theorem 3.5 auch im Fall gültig, dass H die zusätzlichen Glattheitsbedingungen auf die ersten und zweiten Ableitungen nur in der Umgebung von Γ_s erfüllt. Unter den Annahmen von Theorem 3.4 ist es möglich eine obere Schranke für den Fehler $E^*(n)$ als eine Funktion der Anzahl der Auswertungen des AEP-Algorithmus zu berechnen. In der Tat kann (4.35) als

$$E^*(n) \leq A f_*(d)^n \tag{4.37}$$

umgeschrieben werden. Wir bezeichnen mit $M(n)$ die totale Anzahl der Auswertungen der multivariaten Verteilung H, die durch den AEP-Algorithmus nach der n-ten Iteration durchgeführt wurden. Dann ist $M(n)$ proportional zur Anzahl der Simplizes $f_S(d)^{n-1}$, die für die n-te Iteration verwendet wurden. Für alle $n \geq 2$ gilt

$$\begin{aligned} M(n) &= \sum_{k=0}^{n-1} 2^d f_S(d)^k = \frac{2^d}{f_S(d) - 1} (f_S(d)^n - 1) \\ &\geq \left(\frac{2^d}{f_S(d) - 1} - 1 \right) f_S(d)^n = B f_S(d)^n. \end{aligned} \tag{4.38}$$

In diesem Fall ist B eine positive Konstante, die nur von der Dimension d abhängt. Kombinieren wir (4.37) und (4.38), so erhalten wir

$$E^*(n) \leq A \left(\frac{M(n)}{B} \right)^{\ln f_*(d) / \ln f_S(d)}. \tag{4.39}$$

Dann liefert (4.39) eine obere Schranke für den Approximationsfehler $E^*(n)$ des AEP-Algorithmus als eine Funktion der Anzahl der durchgeführten Auswertungen.

Kapitel 5

Anwendung des AEP-Algorithmus und Vergleich mit anderen Methoden

In diesem Kapitel wird der im Kapitel 4 vorgestellte AEP-Algorithmus auf einen zweidimensionalen Risikovektor (X_1, X_2) angewendet. Anschießend werden wir weitere Methoden zur Berechnung von $\mathbb{P}[X_1 + X_2 \leq s]$ einführen und deren Resultate mit den Ergebnissen des AEP-Algorithmus vergleichen.

5.1 Anwendung des AEP-Algorithmus

Wir betrachten ein zweidimensionales Portfolio mit Pareto-verteilten Rändern, d.h.

$$F_i(x_i) = \mathbb{P}[X_i \leq x_i] = 1 - (1 + x_i)^{-\alpha_i} \qquad x \geq 0, i = 1, 2,$$

mit Paramter $\alpha_1 = 0.9$ und $\alpha_2 = 1.2$. Wir verbinden diese Pareto-verteilten Ränder durch eine bivariate Clayton Copula $C = C_\delta^{Cl}$ mit

$$C_\delta^{Cl} = (u_1^{-\delta} + u_2^{-\delta} + 1)^{-1/\delta}, \qquad u_1, u_2 \in [0, 1].$$

Dieses Beispiel sowie die Resultate des AEP-Algorithmus für $n = 16$ Iterationen stammen aus [1]. Als Parameter δ wählen wir 1.2. Daraus erhalten wir unter Verwendung von Sklar's Theorem die multivariate Verteilungsfunktion H

$$\begin{aligned} H(x_1, x_2) &= C_\delta^{Cl}(F_1(x_1), F_2(x_2)) \\ &= \left((1 - (1 + x_1)^{-0.9})^{-1.2} + (1 - (1 + x_2)^{-1.8})^{-1.2} - 1\right)^{-1/1.2}. \end{aligned}$$

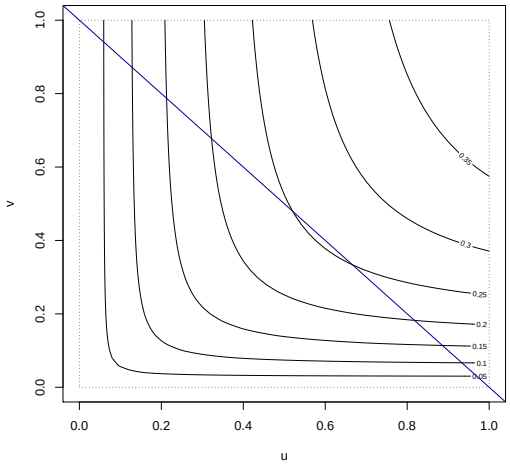

Abbildung 5.1: Konturdiagramm einer bivariaten Clayton Copula mit Parameter $\delta = 1.2$ und Pareto-verteilten Rändern mit Parameter $\alpha_1 = 0.9, \alpha_2 = 1.8$ und $\theta = 1$.

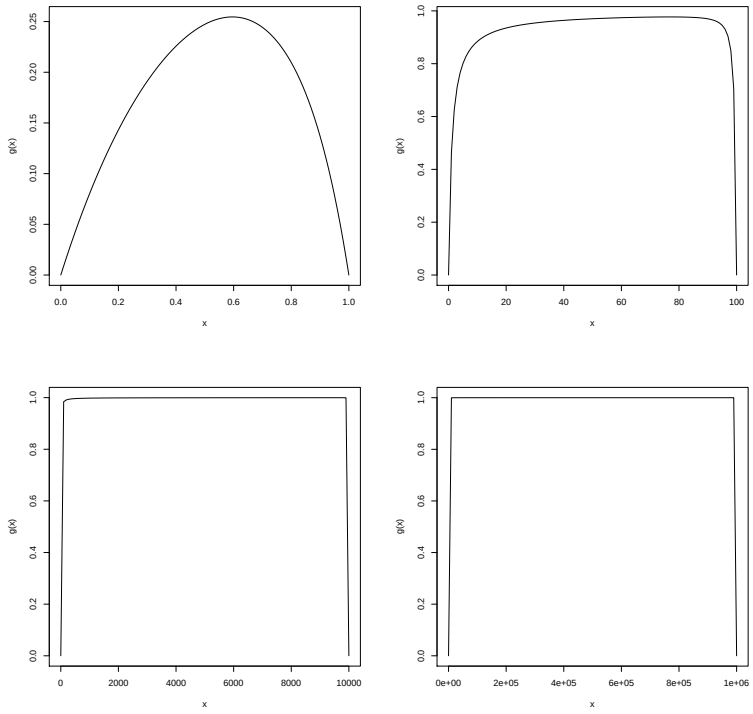

Abbildung 5.2: $g(x)$ für $s = 10^0, 10^2, 10^4, 10^6$

5.1.1 Untersuchung der multivariaten Verteilung

Zunächst untersuchen wir die Verteilungsfunktion H, um ein besseres Verständnis über die Eigenschaften der Verteilung und die Resultate des AEP-Algorithmus sowie der anderen Methoden zu erhalten.

Die gemeinsame Dichtefunktion ist gegeben durch

$$h(x_1, x_2) = (1.2+1)\left(\left(1-(1+x_1)^{-0.9}\right)^{-1.2} + \left(1-(1+x_2)^{-1.8}\right)^{-1.2} - 1\right)^{-2-1/1.2} \left(1-(1+x_1)^{-0.9}\right)^{-2.2} 0.9(1+x_1)^{-1.9} \left(1-(1+x_2)^{-1.8}\right)^{-2.2} 1.8(1+x_2)^{-2.8}. \tag{5.1}$$

Als nächstes betrachten wir folgende Funktion

$$g(x) = H(x, s-x) = \left(\left(1-(1+x)^{-0.9}\right)^{-1.2} + \left(1-(1+s-x)^{-1.8}\right)^{-1.2} - 1\right)^{-1/1.2}$$

für $s = 10^0, 10^2, 10^4, 10^6$. Die Funktion g spiegelt das Verhalten der Verteilungsfunktion entlang der Diagonale $x_1 + x_2 = s$ wieder. Die erste Ableitung von g ist gegeben durch

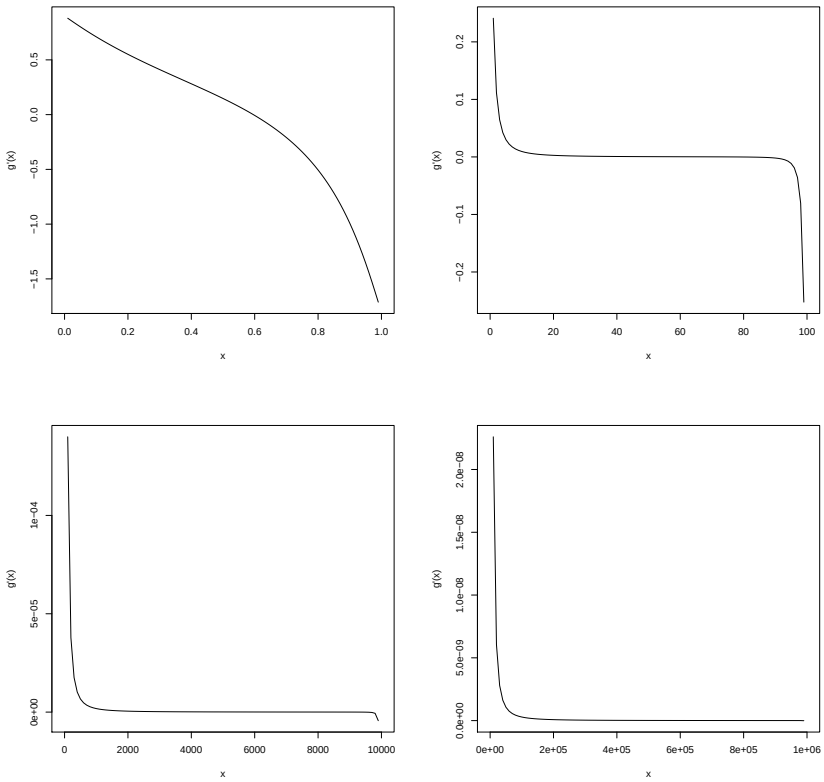

Abbildung 5.3: $g'(x)$ für $s = 10^0, 10^2, 10^4, 10^6$

$$
\begin{aligned}
g'(x) &= H_{x_1}(x, s-x) - H_{x_2}(x, s-x) \\
&= \left(F_1(x)^{-\delta} + F_2(s-x)^{-\delta} - 1\right)^{-1-1/\delta} \cdot \\
&\quad \left(F_1(x)^{-1-\delta} f_1(x) - F_2(s-x)^{-1-\delta} f_2(s-x)\right).
\end{aligned}
$$

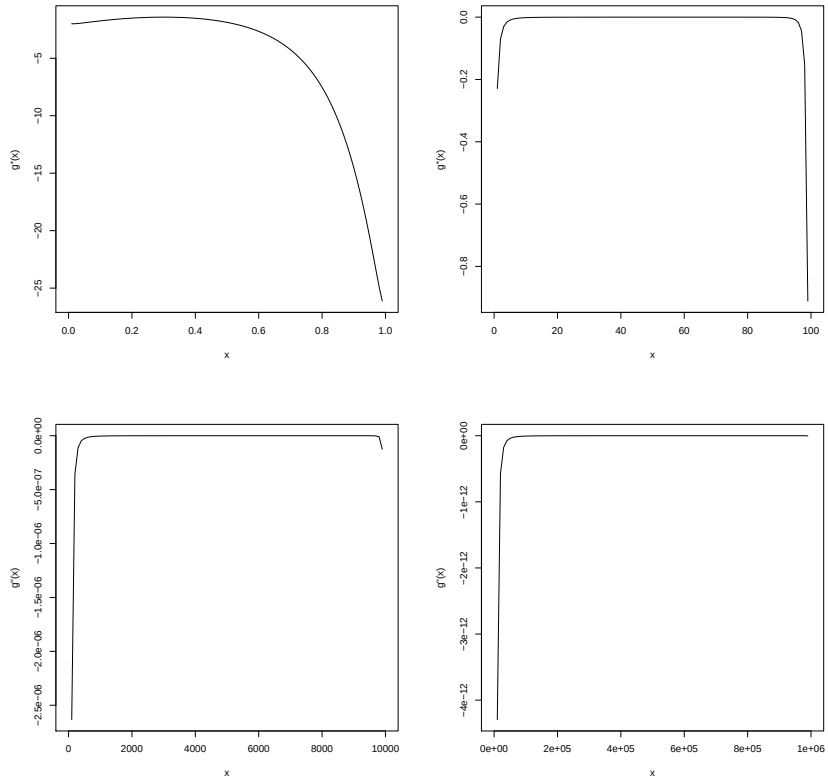

Abbildung 5.4: $g''(x)$ für $s = 10^0, 10^2, 10^4, 10^6$

Die zweite Ableitung von g erhalten wir durch

$$g''(x) = H_{x_1x_1}(x, s-x) - 2H_{x_1x_2}(x, s-x) + H_{x_2x_2}(x, s-x).$$

Die partiellen Ableitungen sind gegeben durch:

$$
\begin{aligned}
H_{x_ix_i}(x_1, x_2) = (1+\delta)\left(F_1(x_1)^{-\delta} + F_2(x_2)^{-\delta} - 1\right)^{-2-\frac{1}{\delta}} \left(F_i(x_i)^{-1-\delta} f_i(x_i)\right)^2 + \\
\left(F_1(x_1)^{-\delta} + F_2(x_2)^{-\delta} - 1\right)^{-1-\frac{1}{\delta}} \cdot \\
\left((-1-\delta)F_i(x_i)^{-2-\delta} f_i(x_i)^2 + F_i(x_i)^{-1-\delta} f_i'(x_i)\right)
\end{aligned}
$$

für $i = 1, 2$ und $\delta = 1.2$ und

$$H_{x_1x_2}(x_1, x_2) = (1+\delta)\left(F_1(x_1)^{-\delta} + F_2(x_2)^{-\delta} - 1\right)^{-2-\frac{1}{\delta}} \cdot F_1(x_1)^{-1-\delta} f_1(x_1) F_2(x_2)^{-1-\delta} f_2(x_2).$$

5.2 Numerische Integration

5.2.1 Adaptive Integrationsmethoden

In diesem Abschnitt werden wir versuchen $V_H[\mathcal{S}(\mathbf{0}, s)]$ durch numerische Integration näherungsweise zu berechnen, dazu verwenden wir die R-Packages `R2Cuba`, `cubature` und `SimplicialCubature`. Einen Sammlung von Programmen zur mehrdimensionalen numerischen Integration findet man in [7]. D.h. im Fall des zweidimensionalen Portfolios aus Abschnitt 5.1 wollen wir folgendes Integral berechnen:

$$V_H[\mathcal{S}(\mathbf{0}, s)] = \int_{\mathcal{S}(\mathbf{0},s)} dH(\mathbf{x}) = \int_0^s \int_0^{s-x_1} h(x_1, x_2) dx_1 dx_2.$$

Als Erstes verwenden wir aus dem Package `R2Cuba` den Algorithmus `cuhre`, das in [8] genau beschrieben wird. `Cuhre` ist ein deterministischer Algorithmus zur mehrdimensionalen numerischen Integration. Der Algorithmus verwendet eine von mehreren Quadraturregeln in einem global adaptiven Unterteilungsschema. `Cuhre` ist sehr mächtig in moderaten Dimensionen und ist in der Regel eine Methode, um hohe Genauigkeit zu erhalten, d.h. mit relativer Genauigkeit weit unter 10^{-3}.

In Tabelle 5.1 vergleichen wir die Resultate des AEP-Algorithmus nach $n = 16$ Iterationen mit den Ergebnissen von `Cuhre`, wobei eine relative Genauigkeit von `rel.tol`=1e-08 gewählt wurde. Wir können sehen, dass für ein wachsendes s die Genauigkeit der Ergebnissen von `Cuhre` sinkt.

s	AEP-Resultate	`cuhre`	Absoluter Fehler
10^0	0.315835041363441	0.31583504325	1.885e-09
10^2	0.983690398913354	0.98369310396	2.705e-06
10^4	0.999748719229367	0.99975021066	1.491e-06
10^6	0.999996018908404	0.99850795736	1.488e-03

Tabelle 5.1: Vergleich des AEP-Resultats (n=16) für $V_H[\mathcal{S}(\mathbf{0}, s)]$ mit dem numerischen Integrationsalgorithmus `cuhre`

Als Nächstes verwenden wir die Funktion `adaptIntegrate` aus dem Package `cubature`. Die Funktion führt mehrdimensionale Integration vektorwertiger Integranden über Hyperwürfel durch. Man kann eine maximale Anzahl an Funktionsauswertungen bestimmen (Standardwert ist 0 für kein Limit). Andernfalls wird die Integration angehalten,

wenn der geschätzte Fehler kleiner als der geforderte absolute Fehler oder kleiner als der Absolutbetrag des relativen Fehlers mal dem Integral ist [10].

s	AEP-Resultate	`adaptIntegrate`	Absoluter Fehler
10^0	0.315835041363441	0.3158350413692	5.759e-12
10^2	0.983690398913354	0.9836903987672	1.461e-10
10^4	0.999748719229367	0.9997487192288	5.397e-13
10^6	0.999996018908404	0.9999960105672	8.341e-09

Tabelle 5.2: Vergleich der AEP-Resultate (n=16) für $V_H[\mathcal{S}(\mathbf{0}, s)]$ mit dem numerischen Integrationsalgorithmus `adaptIntegrate`

Die Ergebnisse der Funktion `adaptIntegrate` werden mit den Resultaten des AEP-Algorithmus nach $n = 16$ Iterationen verglichen. Bei der Verwendung der Funktion `adaptIntegrate` wurde eine relative Genauigkeit von `tol`=1e-08 gewählt. Wie aus Tabelle 5.2 ersichtlich ist sind die Ergebnisse von `adaptIntegrate` für alle s bis zur mindestens achten Nachkommastelle genau.

Bei der Verwendung der Funktionen `cuhre` und `adaptIntegrate` ist zu beachten, dass man nicht mit $(x_2 \leq s - x_1)$ abschneiden darf, da sich die Unstetigkeit längs der Diagonalen verheerend auf adaptive Algorithmen auswirkt. Stattdessen substituieren wir das Simplex durch ein Quadrat und erhalten folgendes Integral:

$$V_H[\mathcal{S}(\mathbf{0}, s)] = \int_0^s \int_0^s h\left(x_1, \left(1 - \frac{x_1}{s}\right) x_2\right) \left(1 - \frac{x_1}{s}\right) dx_1 dx_2.$$

Zuletzt betrachten wir das Package `SimplicialCubature` wie man in [13] nachlesen kann. Dieses Package stellt Methoden zur Auswertung von Integralen von der Form

$$\int_{\mathcal{S}} h(\mathbf{x}) d\mathbf{x}$$

bereit, wobei $\mathcal{S}$ ein Simplex ist und $h(\mathbf{x})$ eine Funktion definiert auf $\mathcal{S}$. Mithilfe der Funktion `adaptIntegrateSimplex` kann man unter Verwendung der Methoden durch Genz und Cools eine allgemeine Funktion über ein Simplex integrieren.

Wie oben vergleichen wir wieder die Ergebnisse des AEP-Algorithmus mit den Resultaten von `adaptIntegrateSimplex` für eine relative Genauigkeit `tol`=1e-08. In Tabelle 5.3 können wir sehen, dass die Berechnung des Integrals für $s = 10^6$ durch die Funktion `adaptIntegrateSimplex` problematisch ist.

Im Graph (a) von Abbildung 5.5 werden die Auswertungen jeder Integrationsmethode dargestellt, die benötigt werden, um die gegebenen relativen Genauigkeiten zu erreichen. Im Graph (b) werden die Ergebnisse dieser Methoden für eine wachsende relative Genauigkeit abgebildet.

s	AEP-Resultate	`adaptIntegrateSimplex`	Absoluter Fehler
10^0	0.315835041363441	0.315835040598	7.652e-10
10^2	0.983690398913354	0.983690397186	1.727e-09
10^4	0.999748719229367	0.999748718006	1.223e-09
10^6	0.999996018908404	0.571790493146	4.282e-01

Tabelle 5.3: Vergleich der AEP-Resultate (n=16) für $V_H[\mathcal{S}(\mathbf{0}, s)]$ mit dem numerischen Integrationsalgorithmus `adaptIntegrateSimplex`

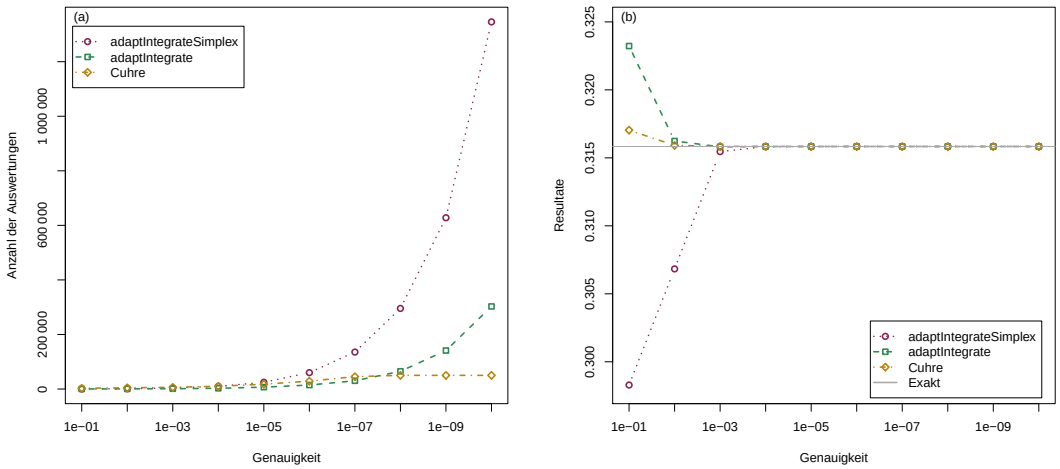

Abbildung 5.5: Vergleich der Integrationsmethoden für $s = 1$ bei steigender Genauigkeit

5.3 Schranken für die adaptive Integration

Um die verschiedenen Algorithmen zur numerischen Integration sinnvoll miteinander vergleichen zu können, werden wir Fehlerschranken für die adaptive Integration ermitteln. Hierfür unterteilen wir das Intervall $[0, s]$ in Teilintervalle

$$I_k = [a_{k-1}, a_k] \text{ mit } a_k = k\frac{s}{n} \text{ für } k = 1, \ldots, n \text{ und } n \in \mathbb{N}.$$

Weiter sei durch

$$x_k = (k - \frac{1}{2})\frac{s}{n}$$

der Mittelpunkt des Teilintervalls I_k mit $k = 1, \ldots, n$ gegeben. Als Erstes betrachten wir die Quadrate $\mathcal{Q}((a_{k-1}, a_{l-1}), \frac{s}{n}) = [a_{k-1}, a_k) \times [a_{l-1}, a_l)$ unter der Diagonale s mit $k, l = 1, \ldots, n-1$. Die Riemann Summe für diese Quadrate ist gegeben durch

$$\mathcal{R}_n = \sum_{k=1}^{n-1} \sum_{l=1}^{n-k} h(x_k, x_l) \left(\frac{s}{n}\right)^2,$$

wobei $h : \mathbb{R}_+^2 \to \mathbb{R}_+$ die gemeinsame Dichte (5.1) bezeichnet. Andererseits können wir das Volumen jedes Quadrats exakt durch

$$V_H[\mathcal{Q}((a_{k-1}, a_{l-1}), \frac{s}{n})] = H(a_k, a_l) - H(a_{k-1}, a_l) - H(a_k, a_{l-1}) + H(a_{k-1}, a_{l-1})$$

berechnen. Mithilfe der Differenz können wir den Fehler für die Quadrate unter der Diagonale schätzen. Diese ist durch

$$\mathcal{L}_n = \sum_{k=1}^{n-1} \sum_{l=1}^{n-k} \left(h(x_k, x_l) \left(\frac{s}{n}\right)^2 - V_H[\mathcal{Q}((a_{k-1}, a_{l-1}), \frac{s}{n})] \right)$$

gegeben. Je feiner die Unterteilung in Teilintervalle ist desto kleiner wird der Fehler $\mathcal{L}_n$.

Als nächstes betrachten wir die Simplizes unter der Diagonale. Diese können wir nicht exakt berechnen, aber wir können das exakte Volumen des Quadrats als obere Schranke verwenden. Es gilt

$$\begin{aligned} \mathcal{U}_{n,k} &= V_H[\mathcal{Q}((a_{k-1}, a_{n-k}), \frac{s}{n})] \\ &= H(a_k, a_{n-k+1}) - H(a_{k-1}, a_{n-k+1}) - H(a_k, a_{n-k}) + H(a_{k-1}, a_{n-k}), \end{aligned}$$

d.h. das Volumen des Simplex $\mathcal{S}((a_{k-1}, a_{n-k}), \frac{s}{n})$ erfüllt die Ungleichung

$$0 \leq V_H[\mathcal{Q}((a_{k-1}, a_{n-k}), \frac{s}{n})] \leq \mathcal{U}_{n,k}.$$

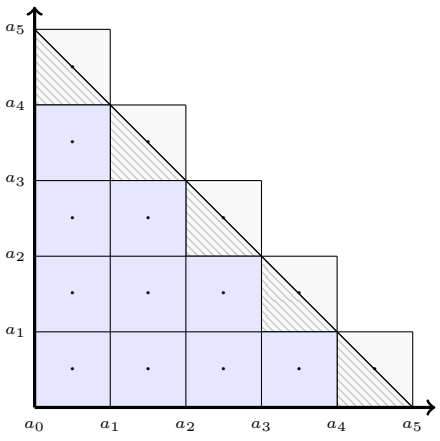

Abbildung 5.6: Zerlegung des zweidimensionalen Simplex $\mathcal{S}(\mathbf{0}, s)$ in Quadrate.

5.4 Rekursion

Die nächste Methode, die wir zur Schätzung von $V_H[\mathcal{S}(\mathbf{0}, s)]$ betrachten, ist die Rekursion. Hierfür werden wir ein paar neue Notationen einführen. Wie bisher betrachten wir ein zweidimensionales Portfolio mit Pareto-verteilten Rändern. Sei $\mathcal{S}(x, y, h)$ ein Simplex, wobei (x, y) der Anfangswert des Simplex ist und h eine reelle Zahl. Weiter sei das Volumen des Hyperwürfels $\mathcal{Q}(x, y, h)$ gegeben durch

$$\mathbf{Q}_H(x, y, h) = H(x + h, y + h) - H(x + h, y) - H(x, y + h) + H(x, y).$$

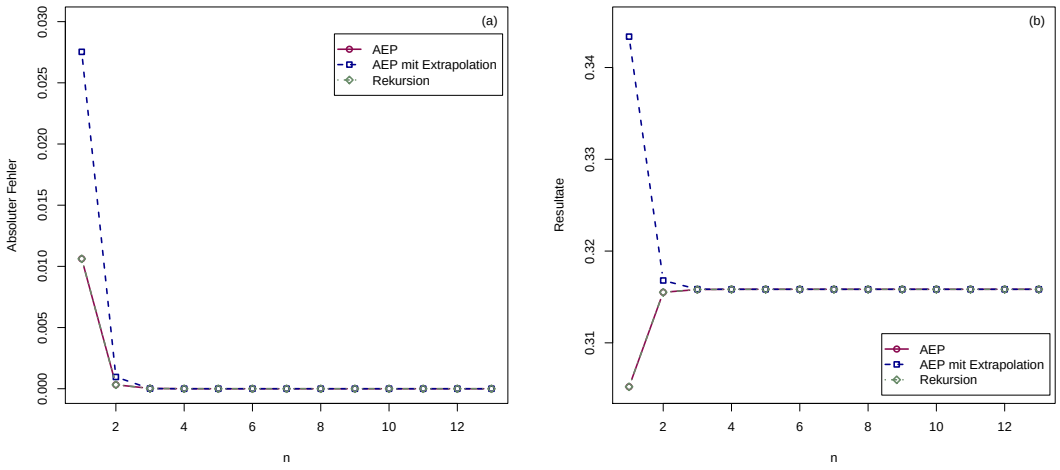

Abbildung 5.7: Absolute Fehler und Resultate des AEP-Algorithmus mit und ohne Extrapolation für $s = 1$

Unter Verwendung der folgenden Rekursion berechnen wir das $\mathbf{V}_H$-Maß des Simplex $\mathcal{S}(x, y, h)$. Der Startwert, d.h. $n = 1$, ist gegeben ist durch

$$\mathbf{V}_H(x, y, h, z, 1) := z\mathbf{Q}_H(x, y, \alpha h),$$

wobei $z \in \{-1, 1\}$ und $\alpha \in [1/2, 1)$. Die nachfolgenden Werte können wir folgendermaßen berechnen:

$$\begin{aligned}\mathbf{V}_H(x, y, h, z, n) := z\mathbf{Q}_H(x, y, \alpha h) &+ \mathbf{V}_H(x, y + \alpha h, (1 - \alpha)h, z, n - 1) \\ &+ \mathbf{V}_H(x + \alpha h, y, (1 - \alpha)h, z, n - 1) \\ &+ \mathbf{V}_H(x + \alpha h, y + \alpha h, (1 - 2\alpha)h, -z, n - 1)\end{aligned}$$

für $n > 1$.

s	AEP-Algorithmus	Fehler	Rekursion	Fehler
10^0	0.31583504135729	6.15585e-12	0.31583504135729	6.15485e-12
10^2	0.98369039891152	1.83009e-12	0.98369039891152	1.83109e-12
10^4	0.99974871922299	6.37568e-12	0.99974871922299	6.37579e-12
10^6	0.99999601885449	5.39142e-11	0.99999601885449	5.39142e-11

Tabelle 5.4: Resultate des AEP-Algorithmus und der Rekursion für $V_H[\mathcal{S}(\mathbf{0}, s)]$ und $n = 10$. Die berechneten absoluten Fehler sind die Abweichungen vom Referenzwert $P_{16}(s)$

Aus Abbildung 5.7 und Tabelle 5.4 ist ersichtlich, dass die Ergebnisse der Rekursion und des AEP-Algorithmus ohne Extrapolation bis zur 14-ten Nachkommastelle übereinstimmen. Für zwei Dimensionen ist die Rekursion eine leicht zu implementierende Methode, die ab einer Rekursionstiefe von $n = 7$ die ersten acht Nachkommastellen unverändert lässt.

Kapitel 6

Adaptiver AEP-Algorithmus für zwei Dimensionen

Im folgenden Kapitel wird eine adaptive Version des AEP-Algorithmus beschrieben. Unser Ziel ist es mithilfe einer adaptiven Variante des AEP-Algorithmus die Geschwindigkeit zu erhöhen und dabei die gleiche Genauigkeit wie durch den Originalalgorithmus zu erzielen. Anschließend werden die Resultate des adaptiven AEP-Algorithmus und des AEP-Algorithmus in Bezug auf Genauigkeit und die Anzahl der benötigten Unterteilungen verglichen. Die Grundstruktur des adaptiven AEP-Algorithmus basiert auf den Aufbau adaptiver Integrationsalgorithmen, wie zum Beispiel dargestellt in [17].

6.1 Adaptiver Integrationsalgorithmus

Adaptive Integrationsalgorithmen beinhalten folgende Komponenten:

1. Ein *Basisintegrationsmodul* erhält einen Input und berechnet einen Schätzwert für das Integral
$$Q(f; B_s) \approx I(f; B_s)$$
als auch eine passende Fehlerabschätzung $E(f; B_s)$, welche in den meisten Fällen die Ungleichung
$$Q(f; B_s) - I(f; B_s) \leq E(f; B_s)$$
erfüllt.

2. Ein *Identifikationsmodul* ist verantwortlich für die Sammlung und Verarbeitung von Daten (Schätzwerte des Integrals und Fehlerabschätzungen in Teilbereichen), die wir vom Basismodul erhalten. Es liefert eine a posteriori Datenergänzung (z.B. die Berechnung der entsprechenden Werte für die Abschätzung des asymptotischen Fehlerverhaltens).

3. Information, die wir auf diese Weise erhalten haben, wird einem *Entscheidungsmodul* zur Verfügung gestellt, das eine Kontrollstrategie für eine adaptive Modifikation der Eingabedaten, die wir durch das Basisintegrationsmodul erhalten,

implementiert. Das Eingreifen, das für die Implementierung der Kontrollstrategie erforderlich ist, wird in ein *Modifikationsmodul* durch die Anpassung der entsprechenden Kontrollparameter durchgeführt. Die Parameter entscheiden in erster Linie, *wo* die Funktion als Nächstes berechnet werden soll.

6.2 Beschreibung des adaptiven AEP-Algorithmus

Die oben beschriebene Herangehensweise nehmen wir uns zum Vorbild, um einen adaptiven Algorithmus zur Bestimmung von $V_H[\mathcal{S}(\mathbf{0}, s)]$ zu entwickeln. Der adaptive AEP-Algorithmus besteht aus einem lokalen Modul, einem Identifikationsmodul und einem Entscheidungsmodul. Diese Module werden unten ausführlich erklärt.

6.2.1 Lokales Modul

Im *lokalen Modul* wird das Volumen eines Simplex $\mathcal{S}(\mathbf{b}, h)$ durch das Volumen des Hyperwürfels $\mathcal{Q}(\mathbf{b}, \alpha h)$ approximiert. Der dabei entstehende Approximationsfehler wird durch obere und untere Schranken geschätzt. Diese Schranken sind definiert durch

$$\mathcal{U}(\mathbf{b}, h) = V_H[\mathcal{Q}(\mathbf{b}, \alpha h)] + V_H[\mathcal{Q}(\mathbf{b} + \alpha h \mathbf{i}_1, (1-\alpha)h)] + V_H[\mathcal{Q}(\mathbf{b} + \alpha h \mathbf{i}_2, (1-\alpha)h)]$$

und

$$\mathcal{L}(\mathbf{b}, h) = V_H[\mathcal{Q}(\mathbf{b}, \alpha h)] - V_H[\mathcal{Q}(\mathbf{b} + \alpha h \mathbf{i}_3, (1-2\alpha)h)]$$

mit $\alpha \in [1/d, 1), \mathbf{b} \in \mathbb{R}^d$ und $h \in \mathbb{R}$. Die Vektoren $\mathbf{i}_1 = (1, 0), \mathbf{i}_2 = (0, 1)$ und $\mathbf{i}_3 = (1, 1)$ sind wie im Abschnitt 3.3 definiert. Die Fehlerabschätzung ist gegeben durch die Differenz der oberen und unteren Schranke, d.h. $\Delta_{\mathcal{S}(\mathbf{b},h)} = \mathcal{U}(\mathbf{b}, h) - \mathcal{L}(\mathbf{b}, h)$.

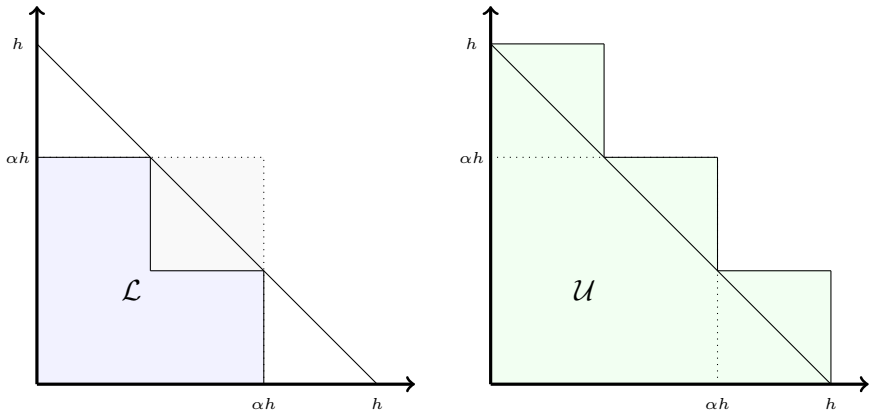

Abbildung 6.1: Untere und obere Schranke für das Simplex $\mathcal{S}(\mathbf{b}, h)$ mit $\alpha = 2/3$

6.2.2 Identifikationsmodul

Im *Identifikationsmodul* werden die Daten aus dem lokalen Modul mithilfe eines Heaps verarbeitet. Eine ausführliche Erklärung zu Heaps kann man in [15] nachlesen. Ein *Heap* ist eine einfache Datenstruktur, die Datensätze in einem Array speichert, sodass jeder Schlüssel stets größer ist als die Schlüssel an zwei anderen konkreten Positionen. Wiederum muss jeder dieser Schlüssel größer als zwei weitere Schlüssel sein usw.

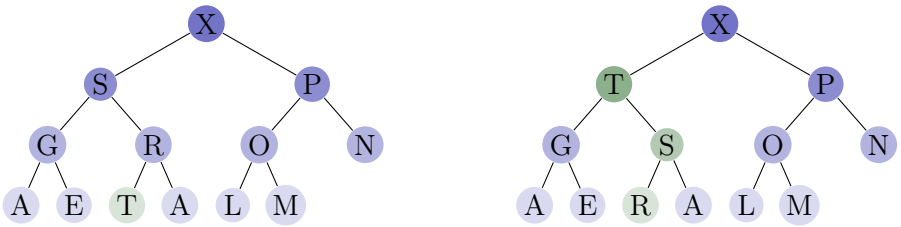

Abbildung 6.2: Wiederherstellen der Heap-Eigenschaft

Der Heap wird als ein Array A implementiert, dessen Größe durch die maximale Anzahl an Simplizes N_{max} beschränkt ist. In diesem Heap wird jedes Simplex mit der dazugehörigen Fehlerabschätzung $\Delta_{\mathcal{S}(\mathbf{b},h)}$ als Schlüssel gespeichert. Kein Simplex in einem Heap-geordneten Baum hat einen Schlüssel, der größer als der Schlüssel in der Wurzel ist. Diese Eigenschaft bezeichnen wir als *Heap-Eigenschaft*.

Für ein besseres Verständnis der Struktur eines Heaps kann die Abbildung 6.2 betrachtet werden.

Fügen wir ein weiteres Simplex am Ende des Heaps ein, so kann es vorkommen, dass die Heap-Eigenschaft verletzt wird, weil der Schlüssel eines Simplex größer ist als der Schlüssel des Vorgängersimplex. Um die Heap-Eigenschaft wiederherzustellen, wenn die Priorität eines Simplex erhöht wurde, durchlaufen wir den Heap nach oben und vertauschen bei Bedarf das Simplex an Position k mit seinem übergeordneten Simplex an Position $\lfloor k/2 \rfloor$. Das setzen wir so lange fort, bis $A[\lfloor k/2 \rfloor] < A[k]$ oder die oberste Ebene des Heaps erreicht ist.

6.2.3 Entscheidungsmodul

Im *Entscheidungsmodul* wird das Simplex mit der größten Fehlerabschätzung vom Heap entfernt. Ist die Fehlerabschätzung $\Delta_{\mathcal{S}(\mathbf{b},h)}$ des Simplex größer als die vorgegebene absolute Fehlerschranke, d.h.

$$\Delta_{\mathcal{S}(\mathbf{b},h)} > \varepsilon \cdot \frac{h^2}{s^2},$$

so wird das Simplex mit durch einen Hyperwürfel approximiert, wobei ε die geforderte Genauigkeit ist. Dabei entstehen drei neue Simplizes, die wie folgt definiert sind:

$$
\begin{aligned}
\mathcal{S}_1 &= \mathcal{S}(\mathbf{b} + \alpha h \mathbf{i}_1, (1-\alpha)h), \\
\mathcal{S}_2 &= \mathcal{S}(\mathbf{b} + \alpha h \mathbf{i}_2, (1-\alpha)h), \\
\mathcal{S}_3 &= \mathcal{S}(\mathbf{b} + \alpha h \mathbf{i}_3, (1-2\alpha)h).
\end{aligned}
$$

Anschließend werden die neuen Simplizes im Heap gespeichert und das lokale Modul wird an jedem der drei Teilsimplices angewandt, um neue Schätzwerte für deren Volumen und Fehler zu erhalten. Dieser Prozess wird so lange durchgeführt bis entweder die Fehlertoleranz erfüllt wird oder bis die Anzahl der erzeugten Simplizes den Eingabeparameter N_{max} überschreitet.

Um die Heap-Eigenschaft wiederherzustellen, wenn die Priorität eines Simplex verringert wurde, durchlaufen wir den Heap nach unten, vertauschen bei Bedarf das Simplex an Position k mit dem größeren seiner beiden Nachfolgersimplices und stoppen, wenn das Simplex an Position k nicht kleiner als seine beiden Nachfolger ist oder die unterste Ebene erreicht wurde. Zu beachten ist, dass das Simplex an Postion k nur einen Nachfolger hat, wenn N gerade und k gleich $N/2$ ist. Die Schleife in diesem Programm hat zwei gesonderte Austrittspunkte; einen für den Fall, dass die unterste Ebene des Heaps erreicht ist, und einen anderen für den Fall, dass die Heap-Eigenschaft bereits in der Mitte des Heaps erfüllt ist.

6.3 Vergleich der Algorithmen

In diesem Abschnitt geben wir einen Überblick über die Ergebnisse des adaptiven Algorithmus für das zweidimensionale Portfolio aus Abschnitt 5.1.

ε	AEP-Algorithmus	Fehler	Adaptiver Algorithmus	Fehler
1e-01	0.30521459866386	1.0620e-02	0.30521459866386	1.0620e-02
1e-02	0.31580954037671	2.5501e-05	0.31580954037671	2.5501e-05
1e-03	0.31583468522434	3.5614e-07	0.31583795992763	-2.9186e-06
1e-04	0.31583504086782	4.9562e-10	0.31583503582694	5.5365e-09
1e-05	0.31583504135729	6.1559e-12	0.31583504209780	-7.3436e-10
1e-06	0.31583504136330	-1.4566e-13	0.31583504136638	-2.9367e-12
1e-07	0.31583504136334	9.9809e-14	0.31583504136367	-2.2626e-13

Tabelle 6.1: Resultate des AEP-Algorithmus und des adaptiven Algorithmus für $V_H[\mathcal{S}(\mathbf{0}, s)]$ für die vorgegebene Genauigkeit ε. Die berechneten Fehler sind die Abweichungen vom Referenzwert $P_{16}(s)$

Die Resultate des adaptiven Algorithmus zur Approximation von $V_H[\mathcal{S}(\mathbf{0}, s)]$ werden in Tabelle 6.1 mit den Ergebnissen des AEP-Algorithmus verglichen.

ε	AEP-Algorithmus	Adaptiver Algorithmus	`adaptIntegrate-Simplex`	`cuhre`
1e-01	1	1	3	22
1e-02	9	9	15	35
1e-03	81	73	74	49
1e-04	2 187	745	185	79
1e-05	19 683	7 791	481	136
1e-06	177 147	79 415	1 155	223
1e-07	1 594 323	795 883	2 601	349

Tabelle 6.2: Anzahl der benötigten Simplizes bzw. Teilbereiche zur Berechnung von $V_H[\mathcal{S}(\mathbf{0}, s)]$ für die vorgegebene Genauigkeit ε. Die berechneten Fehler sind die Abweichungen vom Referenzwert $P_{16}(s)$

Als Nächstes betrachten wir die Anzahl der Simplizes bzw.Teilbereiche, die von den verschiedenen Methoden benötigt werden, um $V_H[\mathcal{S}(\mathbf{0}, s)]$ für die vorgegebene Genauigkeit ε zu approximieren. Wie in Abschnitt 4.1 beschrieben erhält der AEP-

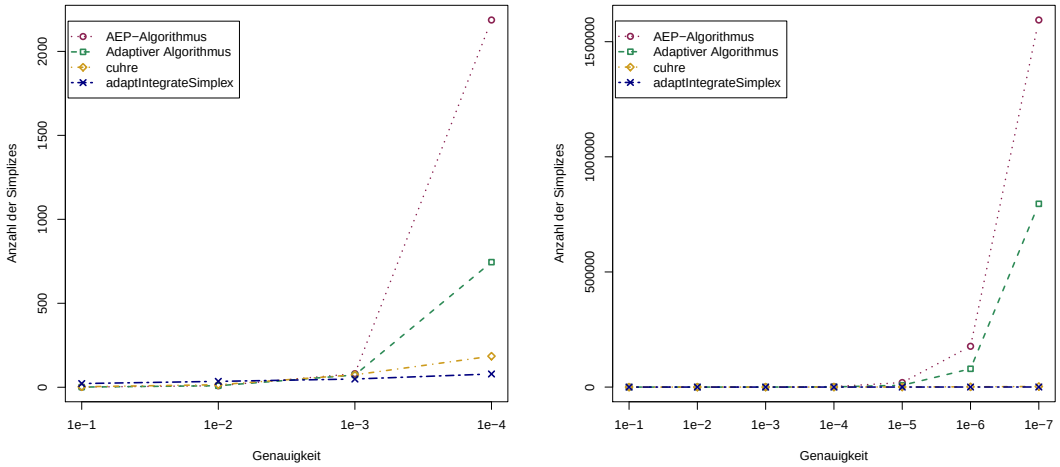

Abbildung 6.3: Anzahl der benötigten Simplizes zur Erzielung der gegebenen Genauigkeiten für ein zweidimensionales Portfolio mit Parteo-veteilten Rändern mit den Parametern $\alpha_1 = 0.9, \alpha_2 = 1.8$ und $\theta = 1$.

Algorithmus pro Iterationsschritt N^{n-1} Simplizes als Input für $N = 2^d - 1$ und $n \geq 1$. Der adaptive Algorithmus erhält hingegen $2(c-1) - 1$ Simplizes als Input, wobei c den Zähler der Funktionsauswertungen bezeichnet. Wie wir sehen können benötigt der adaptive Algorithmus ab einer geforderten Genauigkeit von $\varepsilon =$ 1e-04 ungefähr halb so viel Simplices wie der AEP-Algorithums um vergleichbare Resultate zu erzielen.

6.4 Abschließende Bemerkung

In dieser Arbeit haben wir den AEP-Algorithmus aus dem Artikel [1] von P. Arbenz, P. Embrechts und G. Puccetti zur Berechnung von $\mathbb{P}[X_1 + \ldots + X_d \leq s]$ für abhängige Zufallsvariablen $X_1, \ldots, X_d$ und eine gegebene multivariate Verteilung H beschrieben.

Im Gegensatz zu numerischen Integrationsmethoden müssen wir für die Implementierung des AEP-Algorithmus nicht die gemeinsame Dichtefunktion h berechnen, da diese wie man in (3.4) sieht, sehr komplex sein kann. Obwohl die Integrationsmethoden `adaptIntegrateSimplex` und `cuhre` weit weniger Simplizes bzw. Teilbereiche benötigen, um die vorgegebene Genauigkeit zu erreichen, wie aus Tabelle 6.2 ersichtlich ist, stellt sich der AEP-Algorithmus als effektiver heraus, da für ein wachsendes s die Genauigkeit der Ergebnisse der Integrationsmethoden sinkt (Tabellen 5.1 und 5.3).

Ein Nachteil des AEP-Algorithmus ist, dass mit der Anzahl der Iterationen die Anzahl der Simplizes exponentiell wächst, was zu einer Verlangsamung des Algorithmus führt. Mit Hilfe adaptiver Integrationsmethoden haben wir nach dem Vorbild des AEP-Algorithmus einen adaptiven Algorithmus entwickelt, der pro Auswertung nur drei neue Simplizes erzeugt. Somit wurde die Anzahl der benötigten Simplizes gesenkt ohne Einbußen in Bezug auf Genauigkeit hinnehmen zu müssen. Ein Thema für weitere Untersuchungen wäre das Verhalten des adaptiven Algorithmus für Dimensionen $d > 2$ zu analysieren, da dies für die Berechnung VaR-basierten Eigenkapitalerfordernisses im operationellen Risikomanagement von Bedeutung wäre.

Kapitel 7

Verwendeter R-Code

Der R-Code für die Resultate dieser Arbeit wird unten angeführt. Neben den selbst programmierten Funktionen und Algorithmen haben wir folgende R-Packages verwendet: cubature, copula, actuar, R2Cuba und SimplicialCubature. Diese Packages sind ausführlich in [10], [9], [4], [8] und [13] dokumentiert.

7.1 R-Code zur Berechnung des Integrals 4.23

```
#Remove everything from the workspace to start fresh
rm(list = ls(all = TRUE))
options(digits=15)
##Load package "cubature"
library('cubature')

###Parameters
d = 2  #Dimension of copula
de = 1.2  #Parameter of copula

##Rate parameter of exponential distribution
l1 = 1.5
l2 = 0.5

##P(S_d<=s)
s = 10
##Requested accuracy
e = 1e-10

###Copmute joint density function with exponential margins
h = function(x){
  F1 = 1-exp(-l1*x[1])
```

```
  F2 = 1-exp(-l2*x[2])

  f1 = l1*exp(-l1*x[1])
  f2 = l2*exp(-l2*x[2])

  g = (1+de)*(F1^(-de)+F2^(-de)-1)^(-1/de-2)*(F1*F2)^(-de-1)
  f = g*f1*f2

  return(ifelse(x[1] > 0 && x[2] > 0, f, 0))
}

##Linear transformation of simplex to square
hh = function(x) h(c(x[1],(1-x[1]/s)*x[2]))*(1-x[1]/s)

##Calculation of P(S_d<=s) with adaptIntegrate
Int2 = adaptIntegrate(hh, rep(0,2), rep(s,2), tol = e)
print("Calcuation with adaptIntegrate")
print(Int2)
```

7.2 R-Code für den AEP-Algorithmus

```
##Remove everything from the workspace to start fresh
rm(list = ls(all = TRUE))

options(digits = 15)

##Load packages "copula" and "actuar"
library('copula')
library('actuar')

### Input parameters
d = 2  #Dimension
del = 1.2 #Copula parameter
N = 2^d-1

##Parameters of Pareto distribution
th = 1  #Scale parameter
a1 = 0.9  #Shape parameter
a2 = 1.8  #Shape parameter

##Parameters for the AEP-algorithm
iter = 25 #Number of iterations
x = 10^0  #P(S_d<=x), x=10^0, 10^2, 10^4, 10^6
```

```
al = 2/3  #Alpha is element of the interval (1/d,1]

##Parameters for error estimation
absErr = 1e-3
##Exact value
Exakt = 0.315835041363441

#Initial value of the simplex dependent on the dimension
b = list()
b[[1]] = matrix(rep(0,d),ncol = 1)
#Fixed positive treshold
h = list()
h[[1]] = x
#Indicates wether the measure of the simplex
#has to be added or subtracted
s = list()
s[[1]] = 1
#VH-measure of a hypercube Q(b,al*h)
VH=list()
#Lower bound
LB = list()
#Upper bound
UB = list()
#Absolute error of simplex
AE = list()

###Generate the vectors i1,...,iN in {0,1}^d
y = t(as.matrix(expand.grid(rep(list(0:1),d))))
colnames(y) = NULL

##Compute Clayton copula
clayCop = claytonCopula(del,d)

##Compute multivariate distribution
H = mvdc(clayCop, c("pareto","pareto"),
     list(list(shape = a1, scale = th),
     list(shape = a2, scale = th)))

###Calculate the values of m^j
m = seq(0,0,length.out=N)
for(j in 2:(N+1)){
  if (sum(y[,j])<(1/al)) {
    m[j-1] = (-1)^(1+sum(y[,j]))}
```

```
  else if (sum(y[,j])>(1/al)) {
    m[j-1] = (-1)^(d+1-sum(y[,j]))}
  else {m[j-1] = 0
}}

##Calculation of the VH-measure of a hypercube Q(b,h)
Q = function(b,h){
   b1 = b[1]
   b2 = b[2]
   pMvdc(c(b1+h,b2+h),H)-pMvdc(c(b1+h,b2),H)-
   pMvdc(c(b1,b2+h),H)+pMvdc(c(b1,b2),H)
}


###Main for-loop:
###Calculation of the approximation P of VH[S(0,h)]
P=0
for(n in 1:iter){
##Definition of matrices
 VH[[n]] = rep(0,times=N^(n-1))
 LB[[n]] = rep(0,times=N^(n-1))
 UB[[n]] = rep(0,times=N^(n-1))
 AE[[n]] = rep(0,times=N^(n-1))

##Calculation of the VH-measure of a hypercube Q(b,al*h)
for(k in 1:N^(n-1)){
for(j in 1:(N+1)){
  VH[[n]][k] = VH[[n]][k]+(-1)^(d-sum(y[,j]))*
               pMvdc(b[[n]][,k]+al*h[[n]][k]*y[,j],H)
}}

##Approximation of VH[S(0,h)] with extrapolation technique
G = 0
Pstern = 0
for(k in 1:N^(n-1)){
  G = G+s[[n]][k]*VH[[n]][k]
   Pstern = P+(d+1)^d/(2^d*factorial(d))*G
}
##Print number of iteration
cat("n =",n,"\n")
##Print P*
cat("P* =",Pstern,"\n")
```

```
##Approximation of VH[S(0,h)]
for(k in 1:N^(n-1)){
  P = P+s[[n]][k]*VH[[n]][k]
}
Error = 0
for(k in 1:N^(n-1)){
##Calculation of lower bound
LB[[n]][k] = Q(b[[n]][,k],al*h[[n]][k])-
  Q(b[[n]][,k]+al*h[[n]][k]*y[,4],(1-2*al)*h[[n]][k])
##Calculation of upper bound
UB[[n]][k] = Q(b[[n]][,k],al*h[[n]][k])+
  Q(b[[n]][,k]+al*h[[n]][k]*y[,2],(1-al)*h[[n]][k])+
  Q(b[[n]][,k]+al*h[[n]][k]*y[,3],(1-al)*h[[n]][k])
##Absolute error of simplex
AE[[n]][k] = UB[[n]][k]-LB[[n]][k]
Error = Error + AE[[n]][k]
}
##Print results
cat("P = ",P,"\n")
cat("Error=",Error,"Exakt-P = ",Exakt-P,"\n")
##Stop algorithm if error is smaller as the given accuracy
if(Error <= absErr)break
if(n == iter)break

##Calculation of b,h and s as input for the next iteration
b[[n+1]] = matrix(ncol = N^n,nrow = d)
h[[n+1]] = rep(NA,times = N^n)
s[[n+1]] = rep(NA,times = N^n)

for(k in 1:N^(n-1)){
for(j in 1:N){
  b[[n+1]][,N*k-N+j] = b[[n]][,k]+al*h[[n]][k]*y[,j+1]
  h[[n+1]][N*k-N+j] = (1-sum(y[,j+1])*al)*h[[n]][k]
  s[[n+1]][N*k-N+j] = s[[n]][k]*m[j]
}}}
```

7.3 R-Code zur adaptiven Integration

```
#Remove everything from the workspace to start fresh
rm(list = ls(all = TRUE))
options(digits=15)
##Load packages "cubature", "R2Cuba" and "SimplicialCubature"
library('cubature')
```

```
library('R2Cuba')
library('SimplicialCubature')

###Parameters
d = 2  #Dimension of copula
de = 1.2  #Parameter of copula

##Shape parameter of Pareto distribution
al1 = 0.9
al2 = 1.8

##P(S_d<=s), s=10^0, 10^2, 10^4, 10^6
s = 10^0
##Requested accuracy
e = 1e-8

##Exact values for differend thresholds
exact <- c(0.315835041363441, 0.983690398913354,
           0.999748719229367, 0.999996018908404)

###Copmute joint density function with pareto margins
h = function(x){
  F1 = 1-(1+x[1])^(-al1)
  F2 = 1-(1+x[2])^(-al2)

  f1 = al1*(1+x[1])^(-al1-1)
  f2 = al2*(1+x[2])^(-al2-1)

  g = (1+de)*(F1^(-de)+F2^(-de)-1)^(-1/de-2)*(F1*F2)^(-de-1)
  f = g*f1*f2

  return(ifelse(x[1] > 0 && x[2] > 0, f, 0))
}

##Linear transformation of simplex to square
hh = function(x) h(c(x[1],(1-x[1]/s)*x[2]))*(1-x[1]/s)

##Calculation of P(S_d<=s) with cuhre
Int1 = cuhre(2, 1, hh, lower = rep(0,2), upper = rep(s,2),
       rel.tol = e, flags = list(verbose = 0))
print("Calcuation with cuhre")
print(Int1)
```

```
##Calculation of P(S_d<=s) with adaptIntegrate
Int2 = adaptIntegrate(hh, rep(0,2), rep(s,2), tol = e)
print("Calcuation with adaptIntegrate")
print(Int2)
cat("adaptIntegrate: Result-Exact =",
    Int2$integral-exact[1],"\n")

##Calculation of P(S_d<=s) with adaptIntegrateSimplex
S = CanonicalSimplex(2)*s  #Simplex
Int3 = adaptIntegrateSimplex(h, S, maxEvals = 10^8,
                        tol = e, partitionInfo=TRUE)
print("Calcuation with adaptIntegrateSimplex")
print(Int3)
cat("adaptIntegrateSimplex: Result-Exact =",
    Int3$integral-exact[1],"\n")
```

7.4 R-Code zur Rekursion

```
#remove everything from the workspace to start fresh
rm(list = ls(all = TRUE))
#Time = proc.time()

##Load packages "copula" and "actuar"
library('copula')
library('actuar')

### Input parameters for recursion
n = 10
#Recursion depth
al = 2/3  #Alpha is element of the interval (1/d,1]
s = 10^0  #P(S_d<=s), s=10^0, 10^2, 10^4, 10^6

##Exact values for differend thresholds
Exact = c(0.315835041363441, 0.983690398913354,
          0.999748719229367, 0.999996018908404)

##Copula parameters
d = 2  #Dimension
del = 1.2

##Parameters of Pareto distribution
th = 1  #Scale parameter
a1 = 0.9  #Shape parameter
```

```
a2 = 1.8  #Shape parameter

##Compute Clayton copula
clayCop = claytonCopula(del,d)

##Compute multivariate distribution
H = mvdc(clayCop, c("pareto","pareto"),
     list(list(shape = a1, scale = th),
     list(shape = a2, scale = th)))

##VH-measure of a hypercube Q
Q = function(x,y,h){
   pMvdc(c(x+h,y+h),H)-pMvdc(c(x+h,y),H)-
   pMvdc(c(x,y+h),H)+pMvdc(c(x,y),H)
}
##Recursive AEP-Algorithm
V = function(x,y,h,z,n){
 z*Q(x,y,al*h)+
 (if(n>1){V(x,y+al*h,(1-al)*h,z,n-1)+
 V(x+al*h,y,(1-al)*h,z,n-1)+
 V(x+al*h,y+al*h,(1-2*al)*h,-z,n-1)}
 else{0})
}
##Result
Result = V(0,0,s,1,n)

##Print result
cat("n = ",n,"\n")
cat("V = " ,Result,"\n")
cat("Exact-V = ",Exact[1]-Result)
```

7.5 R-Code für den adaptiven AEP-Algorithmus

```
##Remove everything from the workspace to start fresh
rm(list = ls(all = TRUE))

options(digits=15)

TWOAEP <- function(S,F,AL,NMAX,ABSERR){
# Suppose X,Y are non-negative random variables with
# distirbution function F, then compute P[X+Y<=s]
#
### INPUT:
```

```
#
# S       strictly positive real number
# F       distribution function
# AL      partitioning parameter alpha
# NMAX    maximum number of triangles
# ABSERR desired absolute error
#
### OUTPUT: A list with
#
# RESULT the calculated value
# ERROR  the error estimate
# IER    return parameter
#
# IER==0 success, IER==1 means NMAX reached
#
### LOCAL VARIABLES:
#
# A  the heap, organized as array
# N  the heap counter

### Volume for square function
 Q <- function(X,Y,H) F(X+H,Y+H)-F(X+H,Y)-F(X,Y+H)+F(X,Y)

### Initialize empty heap and zero heap counter
 A <- array(dim = c(NMAX+1,5))
 N <- 0
###Counter for the siplex-splits
 COUNT <- 1
### Insert simplex x,y,h,z into heap
 INSHEAP <- function(X,Y,H,Z){
     if(N >= NMAX) stop("Heap full")
     N <<- N+1

### Local module for absolute error estimation
#
# SI  Simplex (X,Y),(X+H,Y),(X,Y+H)
# SQ  Square (X,Y),(X+AL*H,Y),(X,Y+ALH),(X+AL*H,Y+AL*H)
#
## Volume(SQ)
 Q0 <- Q(X,Y,AL*H)

## Lower bound for volume(SI)
 LB <- Q0 - Q(X+AL*H,Y+AL*H,(1-2*AL)*H)
```

```
## Upper bound for volume(SI)
 UB <- Q0 + Q(X,Y+AL*H,(1-AL)*H) + Q(X+AL*H,Y,(1-AL)*H)

## Absolute error estimate for volume(SI)
 AE <- UB - LB

### Update RESULT and ERROR
 RESULT <<- RESULT + Z*Q0
 ERROR <<- ERROR + AE

### Put simplex onto heap;
### MUST call UPHEAP immediately afterwards!!!
 A[N+1,] <<- c(AE,X,Y,H,Z)

# Return the error estimate
 return(AE)
}

### Upheap operation to restore
### heap property after insertion
 UPHEAP <- function(K){
    V <- A[K+1,]
    A[1,1] <<- Inf
   while(A[(J <- K%/%2)+1,1] <= V[1]){
     A[K+1,] <<- A[J+1,]
     K <- J
   }
    A[K+1,] <<- V
}

### Remove element from heap;
### MUST call DOWNHEAP immediately afterwards!!!
 REMHEAP <- function(){
   if(N <= 0) stop("Heap empty")
  V <- A[2,]
  N <<- N-1
  A[2,] <<- A[N+2,]
  return(V)
}

### Downheap operation to restore
### heap property after deletion
```

```
 DOWNHEAP <- function(K){
   V <- A[K+1,]
   K <- 1
   L <- N%/%2
   while(K <= L){
      J <- 2*K
      if(J<N && A[J+1,1]<A[J+2,1]) J <- J+1
      if(V[1] >= A[J+1,1])break
      A[K+1,] <<- A[J+1,]
      K <- J
   }
   A[K+1,] <<- V
}

#
#### MAIN LOOP
#

### Initialize first simplex and put it onto the heap
 X <- 0; Y <- 0; H <- S; Z <- 1;
 RESULT <- 0; ERROR <- 0
 INSHEAP(X,Y,H,Z); UPHEAP(N);

  while(ERROR > ABSERR){

### Check heap overflow
   if(N > NMAX-3)
      return(list(RESULT = RESULT,ERROR = ERROR,IER = 1))

### Remove simplex from top of heap;
### Call it the parent simplex
   V <- REMHEAP(); DOWNHEAP(1)
   AE <- V[1]; X <- V[2]; Y <- V[3]; H <- V[4]; Z <- V[5];

### If absolute error estimate is substantial (?)
### split the simplex (area rule)
   if(AE*S^2 > ABSERR*H^2){
   COUNT <- COUNT + 1
### Children simplices
    X1 <- X; Y1 <- Y+AL*H; H1 <- (1-AL)*H; Z1 <- Z
    X2 <- X+AL*H; Y2 <- Y; H2 <- (1-AL)*H; Z2 <- Z
    X3 <- X+AL*H; Y3 <- Y+AL*H; H3 <- (1-2*AL)*H; Z3 <- -Z
```

```
### Put children onto heap;
### INSHEAP updates RESULT and ERROR
    INSHEAP(X1,Y1,H1,Z1); UPHEAP(N);
    INSHEAP(X2,Y2,H2,Z2); UPHEAP(N);
    INSHEAP(X3,Y3,H3,Z3); UPHEAP(N);

### Remove old error estimate for parent simplex
    ERROR<-ERROR-AE
    }
}
return(list(RESULT = RESULT,ERROR = ERROR,
            IER = 0, N = N, COUNT = COUNT))
}

S = 1
AL = 2/3
NMAX = 1000000
ABSERR = 1E-7

DEL = 1.2  #Copula parameter
##Parameters of Pareto distribution
TH1 = 0.9 #Shape parameter
TH2 = 1.8 #Shape parameter

### Pareto distributions
F1 <- function(X) 1-(1+X)^(-TH1)
F2 <- function(X) 1-(1+X)^(-TH2)

### Compute Clayton copula
G <- function(X,Y){
   if(X == 0||Y == 0) return(0)
   if(X == 1) return(Y)
   if(Y == 1) return(X)
   return((X^(-DEL)+Y^(-DEL)-1)^(-1/DEL))
}

### Compute multivariate distribution
F <-function(X,Y) G(F1(X),F2(Y))

EXACT <- 0.315835041363441

TMP <- TWOAEP(S,F,AL,NMAX,ABSERR)
RESULT = TMP$RESULT
```

```
ERROR = TMP$ERROR
IER = TMP$IER
N = TMP$N
COUNT = TMP$COUNT
### Print results
cat("RESULT =",RESULT,"  EXACT =",EXACT,"  IER =",IER,"\n")
cat("ERROR =",ERROR,"  EXACT-RESULT =",EXACT-RESULT,"\n")
cat("NMAX =",NMAX,"ANZAHL DER SIMPLIZES =",N,"\n")
cat("ANZAHL DER FUNKTIONSAUSWERTUNGEN =",COUNT,"\n"  )
```

Literaturverzeichnis

[1] Arbenz, P., Embrechts, P. und Puccetti, G., *The AEP algorithm for the fast computation of the distribution of the sum of dependent random variables.* Bernoulli vol. 17 ,no. 2 (2011), 562-591.

[2] Basel Comittee on Banking Supervision, *International Convergence of Capital Measurement and Capital Standards.* Bank for International Settlement, 2006.

[3] Chernobai, A. S., Rachev, S. T. und Fabozzi, F. J., *Operational Risk – A Guide to Basel II Capital Requirements, Models, and Analysis.* Wiley, 2007.

[4] Dutang, C., Goulet, V. und Pigeon, M., *actuar: An R Package for Actuarial Science.* Journal of Statistical Software, vol. 25, no. 7 (2008), 1-37.

[5] Embrechts, P. und Hofert, M., *Practices and issues in operational risk modeling under Basel II.* Lithuanian Mathematical Journal, vol. 51, no. 2 (2011), 180-193.

[6] Galeotti, M., *Computing the probability measure of a d-dimensional simplex via overlapping hypercubes.* Working Papers - Mathematical Economics, Universita' degli Studi di Firenze, Dipartimento di Scienze per l'Economia e l'Impresa, 2013.

[7] Hahn, T., *Cuba - a library for multidimensional numerical integration.* Computer Physics Communications, vol. 168, no. 2 (2005), 78-95.

[8] Hahn, T., Bouvier, A. und Kiêu, K., *R2Cuba: Multidimensional Numerical Integration.* R package version 1.0-11.
Online: `http://CRAN.R-project.org/package=R2Cuba` (Version: 2013)

[9] Hofert, M., Kojadinovic, I., Maechler, M. und Yan, J., *copula: Multivariate Dependence with Copulas.* R package version 0.999-12.
Online: `http://CRAN.R-project.org/package=copula` (Version: 2014)

[10] Johnson, S. G. und Narasimhan, B., *cubature: Adaptive multivariate integration over hypercubes.* R package version 1.1-2.
Online: `http://CRAN.R-project.org/package=cubature` (Version: 2013)

[11] McNeil, A.J., Frey, R. und Embrechts, P., *Quantitative Risk Management – Concepts, Techniques and Tools.* Princeton University Press, 2005.

[12] Nelsen, R. B., *An Introduction to Copulas*, second ed., Springer, 2006.

[13] Nolan, J. P. and with parts based on code by Genz, A., *SimplicialCubature: Numerical integration of functions over simplices*. R package version 1.0. Online: `http://CRAN.R-project.org/package=SimplicialCubature` (Version: 2014)

[14] Panjer, H. H., *Operational Risk – Modeling Analytics*. Wiley Series in Probability and Statistics, 2006.

[15] Sedgewick, Robert, *Algorithmen in C++ – Teil 1-4*. Pearson Studium, 2002.

[16] Shevchenko, P. V., *Modelling Operational Risk Using Bayesian Inference*. Springer, 2011.

[17] Ueberhuber, C. W., *Numerical Computation – Methods, Software, and Analysis*. Springer, 1997.