Abschlussprüfung Sommer 2017

zum

Mathematisch-technischer Softwareentwickler/-in (IHK)

vor der IHK Aachen

Entwicklung eines Softwaresystems

Thema:

Carsharing Simulator

Programmiersprache:

C#

Prüfling:	Heitbrock, Felix
Prüflingsnummer:	101 20505
Bearbeitungszeitraum:	15.05.2015-19.05.2015
Ausbildungsbetrieb:	Werkzeugmaschinenlabor RWTH Aachen
	Steinbachstraße 19
	52074 Aachen

I Inhaltsverzeichnis

1 Benutzeranleitung

1.1 Systemvoraussetzungen und Hinweise zum Aufruf

Das Programm wurde unter Windows 10 (64-Bit) in der Programmiersprache C# entwickelt und getestet. Es wird eine Benutzung unter Windows 10 empfohlen, lauffähig sollte das Programm aber auch in anderen Versionen wie etwa Windows 7 sein. Benötigt wird außerdem das .NET Framework, welches mindestens in der Version 4.6 installiert sein muss.

1.2 Installation

Zur Installation der Software muss die gelieferte ZIP-Datei entpackt werden. Alle entstehenden Verzeichnisse, bzw. Dateien sollten mit Schreib- und Leserechten für den ausführenden Benutzer versehen werden.

1.3 Programmstart

Das Programm kann ausgeführt werden, indem die Batch-Datei „run.bat“ im Hauptverzeichnis ausgeführt wird. Es werden dann anschließend alle Dateien im Ordner „Input“ an das Programm übergeben, welches daraufhin je eine Ausgabedatei im Ordner „Output“ generiert. Liegt ein Fehler vor wird keine Ausgabedatei generiert und eine (Fehler-)Meldung wird in die „Error.log“ Datei geschrieben.

In dem Verzeichnis „Testfälle“ sind die Testfälle, die in Kapitel 6 beschrieben werden, zu finden.

Das Programm lässt sich auch manuell über die Kommandozeile mit folgendem Befehl ausführen:

ABLAGEVERZEICHNIS/CarsharingSimulator.exe [in-file] [out-file] [genauigkeit]

Dabei steht „[in-file]“ für den Pfad (inklusive Name) zur Datei, die eingelesen werden soll und „[out-file]“ für den Pfad (inklusive Name) zu der Ausgabedatei, die generiert werden soll. Die „[genauigkeit]“ muss nicht zwingend angegeben werden (standardmäßig 0.0001).

1.4 Ausführung der Testfälle

Die Testfälle befinden sich im Verzeichnis „TestCases“. Sollen diese ausgeführt werden, müssen die jeweiligen Dateien in den Ordner Input kopiert werden. Ist das getan, können sie mit dem Script „run.bat“ ausgeführt werden.

2 Aufgabenanalyse

2.1 Problemstellung

Für die MATSE Firma soll eine Simulation für eine Carsharing Dienstleistung erstellt werden. Innerhalb einer Stadt können Autos geliehen und an anderer Stelle zurückgegeben werden. Nachfrage und Rückgabe der Autos wird über Funktionen, genauer Polynome des 4. Grades, beschrieben. Es soll nun der Bedarf ermittelt werden welcher sich aus der Nachfrage und Rückgabe von Autos ergibt. Der Bedarf gibt an, zu welchem Zeitpunkt zwischen 0 und 24 Stunden wie viele Autos nachgefragt wurden bzw. abgestellt wurden. Es soll pro Quadrat, welches einen Teil einer Stadt darstellt, je eine Bedarfsfunktion ermittelt werden. Ist die Funktion des Bedarfs bekannt, so lässt sich daraus der Endzustand der Stadt ermitteln. Der Endzustand gibt an wie die neue Verteilung der Autos nach 24 Stunden aussieht. Weiterhin soll der maximale Bedarf an Autos ermittelt werden, also zu welchem Zeitpunkt die Bedarfsfunktion maximal wird.

2.2 Ablauf

Das Programm liest die Größe der Simulation sowie Nachfrage- und Angebotsfunktionen aus einer Eingabedatei ein. Anhand der Daten wird dann die Bedarfsfunktion berechnet und die End- und Maximalbedarfszustände ermittelt. Die Ergebnisse der Simulation werden dann in eine Ausgabedatei geschrieben. Das Ganze wird als Konsolenanwendung laufen und die Dateipfade zur Ein und Ausgabedatei werden als Parameter übergeben. Als dritter Parameter kann noch die Genauigkeit angegeben werden. Wird diese nicht mit übergeben wird standardmäßig der Wert 0.0001 angenommen.

2.3 Format der Eingabedatei

Die Eingabe erfolgt über eine Datei, die folgendermaßen strukturiert ist:

```
# AHausen
# m
1
# Polynome Nachfrage
0 0.2023761 -0.0287711 0.0016925 -0.0000352
# Polynome Abstellungen
0.434782 0 0 0 0
```

Formatierung einer gültigen Eingabe

Die erste Kommentarzeile beschreibt die Eingaben. In diesem Fall handelt es sich um Daten zu der Stadt „AHausen“, weshalb der entsprechende Kommentar eingefügt wurde. Als Beschreibung zählt immer nur die erste Zeile, die anderen Kommentarzeilen werden ignoriert. In der ersten Nicht-Kommentarzeile wird das *m* angegeben. Es bestimmt wie viele Quadrate für die Stadt simuliert werden sollen. Es werden m^2 Quadrate simuliert. Damit eine Berechnung

sinnvoll ist, muss m ganzzahlig, positiv und größer als 0 sein. Durch die Angabe von m wird auch bestimmt wie viele Polynome angegeben werden müssen. Erforderlich sind m^2 Polynome für die Nachfrage und nochmal genauso viele für die Abstellungen. Die Polynome besitzen maximal den Grad 4. Es gibt also pro Polynom je 5 Vorfaktoren ($a+bx+cx^2+dx^3+ex^4$). Diese Vorfaktoren werden durch Leerzeichen getrennt angegeben und können negative oder positive Gleitkommazahlen sein.

2.4 Format der Ausgabedatei

Die Resultate werden in einer Ausgabedatei gespeichert, die den Folgenden Aufbau besitzt:

```
# AHausen
Abstellung in Q_11 zu t=2,3
Nachfrage in Q_11 zu t=3,76
Abstellung in Q_11 zu t=4,6
Nachfrage in Q_11 zu t=5,87
Abstellung in Q_11 zu t=6,9
Nachfrage in Q_11 zu t=7,86
Abstellung in Q_11 zu t=9,2
Nachfrage in Q_11 zu t=9,89
Abstellung in Q_11 zu t=11,5
Nachfrage in Q_11 zu t=11,96
Abstellung in Q_11 zu t=13,8
Nachfrage in Q_11 zu t=14,03
Nachfrage in Q_11 zu t=16,06
Abstellung in Q_11 zu t=16,1
Nachfrage in Q_11 zu t=18,06
Abstellung in Q_11 zu t=18,4
Nachfrage in Q_11 zu t=20,17
Abstellung in Q_11 zu t=20,7
Abstellung in Q_11 zu t=23
Nachfrage in Q_11 zu t=23,43
Endzustand des Tages:
0
Maximaler Bedarf:
1
```

Formatierung der Ausgabe

In der Ausgabedatei wird zunächst die Beschreibung (also die erste Kommentarzeile) aus der Eingabedatei eingefügt. Daraufhin folgt der Simulationsverlauf. Im Simulationsverlauf wird abgebildet wann (t), wo (Q) und was (Nachfrage bzw. Nachfrage) eine Änderung auftritt. Eine Änderung bedeutet die Abstellung eines Autos oder Nachfrage nach einem Auto. Die Einträge in der Historie haben immer folgendes Format: „[Aktion] in Q_[x][y] zu t=[t]“. Der Zeitpunkt der Änderung (also t) wird immer bis auf 2 Nachkommastellen gerundet. Nach dem Simulations-

verlauf folgt die Zeile „Endzustand des Tages:“, nach der die Endzustände ausgegeben werden. Dabei wird zeilenweise mit einem Leerzeichen getrennt die Endergebnisse der Quadrate ausgegeben. Es gibt m² Ergebnisse die in m² Zeilen ausgegeben werden. Die Endzustände sind Ganzzahlig und können negative Werte annehmen. Nach der Zeile „Maximaler Bedarf:“ folgt der maximale Bedarf pro Quadrat. Diese werden genau wie die Endergebnisse zeilenweise ausgegeben mit m Werten pro Zeile und m Zeilen also insgesamt m² Werten. Der maximale Bedarf kann nicht negativ werden da der Startwert für jeden Bedarf bei 0 liegt. Der maximale Bedarf ist also eine positive, ganze Zahl größer oder gleich 0.

2.5 Module

Um die geforderten Anforderungen erfüllen zu können werden mehrere Module im Programm benötigt. Es wird ein Modul benötigt, um die Eingabedatei einzulesen und in eine geeignete Datenstruktur zu überführen und die Ausgabedatei zu schreiben. Weiterhin wird ein Modul für die Berechnung der Bedarfsfunktion und eines für die Berechnung der Simulation bzw. des maximalen Bedarfs und Endzustands benötigt.

2.6 Genauigkeitsprobleme und Laufzeit

Das Programm rechnet mit Gleitkommazahlen. Es kann also sehr leicht zu Rundungsfehlern kommen. Die Berechnungen bauen jedoch in diesem Anwendungsfall nicht aufeinander auf, es wird also nicht mit einem Ergebnis einer ungenauen Rechnung weiter gerechnet. Jede Nullstelle wird erneut berechnet, es kommt also nicht zu großen Problemen durch Ungenauigkeiten, die sich im Verlauf der Berechnung aufaddieren und das Ergebnis in großem Maße beeinflussen. Allerdings wird bei der Bestimmung der Nullstellen auf ein numerisches Verfahren gesetzt, das Bisektionsverfahren. Es wird also in den meisten Fällen nur eine Nahrung der Nullstelle bestimmt. Wie genau diese Näherung ausfallen soll, lässt sich über den 3. Kommandozeilenparameter einstellen. Standardmäßig wird als Genauigkeit 0.0001 verwendet. Eine Genauigkeit von 0.001 bedeutet dass 18 Iterationen (da Start Intervall von 0 bis 24) des Bisektionsverfahrens angewendet werden und das Intervall, in dem sich die Nullstelle befindet nun 0.0001 groß ist. Das Genauigkeitsproblem ist also nicht kritisch und lässt sich vernachlässigen.

Ein größeres Problem stellt die Größe des Problems dar. Mit der Größe der Stadt, also der Anzahl an Quadraten (Abstellplätze für Autos), steigt auch die Anzahl an Bedarfsfunktionen, die berechnet werden müssen. Ein weiteres Problem ist die Steigung der Nachfrage- und Angebotsfunktionen. Liegt eine starke Steigung vor, müssen besonders viele Nullstellen berechnet werden. Bei der Funktion $100 \times x^3$ z. B. müssen schon 1382400 Nullstellen berechnet werden. Damit nimmt die Größe des Problems besonders schnell zu, was bedeutet, dass deutlich mehr Rechenschritte benötigt werden und die Berechnung insgesamt sehr lange dauern kann.

2.7 Sonderfälle und Fehlerfälle

Alle Abweichungen von der Formatierung der Eingabe werden als Fehlerfall gewertet. Ausnahmen stellen Änderungen wie das Einfügen von Leerzeichen dar. Dieser Fall wird als Sonderfall behandelt. Andere Sonderfälle sind:

- besonders lange Zahlen, also Zahlen mit vielen Nachkommastellen
- Änderungen knapp vor oder nach 24
- Änderungen von Nachfrage und Abstellung zur exakt derselben Zeit

3 Verbale Beschreibung des Verfahrens

Wie schon in der Aufgabenanalyse beschrieben, werden mehrere Module benötigt, damit das Programm die Eingabedaten einlesen, die Simulation berechnen und das Ergebnis in eine Ausgabedatei schreiben kann. Diese sind:

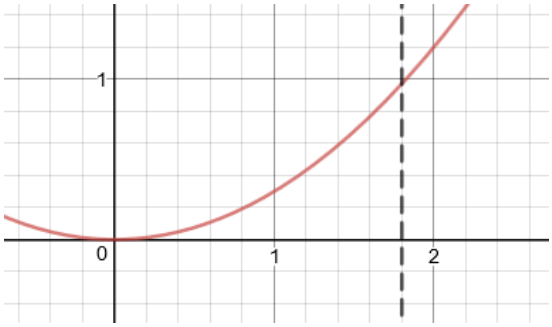
1. Einlesen & Überführen in Datenstruktur
2. Berechnung von Bedarfsfunktion
3. Simulation bzw. Berechnung von Endzustand und maximalem Bedarf
4. Ausgabe

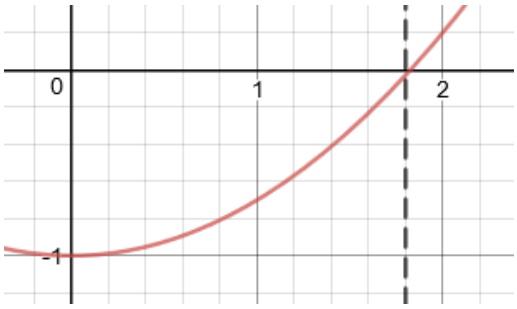
3.1 Einlesen & Überführen in eine Datenstruktur

In diesem Modul werden alle benötigten Daten eingelesen. Das Interface ILesen ermöglicht es theoretisch verschiedene Datenquellen zu unterstützen. Im Rahmen dieser Prüfung werden die Daten aus einer Eingabedatei eingelesen. Dazu wird zunächst der erste Kommentar der Eingabedatei eingelesen und gespeichert – er soll später als Beschreibung auch für die Ausgabedatei dienen. Gekennzeichnet sind die Kommentarzeilen jeweils mit # als erstes Zeichen der Zeile. Abgesehen vom ersten Kommentar werden die anderen Kommentare nicht benötigt und im weiteren Verlauf ignoriert. Als erste Nicht-Kommentarzeile wird m eingelesen. Dabei steht m für die Anzahl der zu simulierenden Teile einer Stadt. Eine Stadt, besteht aus einem Raster mit m² Quadraten. Ein Quadrat repräsentiert einen Teil der Stadt auf denen Autos abgestellt und angefordert werden können. Pro Quadrat werden 2 Polynome benötigt, die in der Eingabedatei angegeben werden müssen – Die Nachfrageverteilung und die Angebotsverteilung (auch Abstellverteilung). Bei m² Quadraten gibt es also m² + m² Polynome. Diese Polynome werden je in einer Zeile angegeben und deshalb zeilenweise eingelesen. Sind diese Daten eingelesen werden sie in eine Datenstruktur überführt so, dass sie verarbeitet werden können.

3.2 Berechnung von Bedarfsfunktion

Die Bedarfsfunktion wird aus dem Integral der Polynome des Angebots und der Nachfrage (bzw. Abstellungen) ermittelt. Die Bedarfsfunktion soll dabei nur positive oder negative Ganzzahlige Werte annehmen können. Dazu muss also bestimmt werden, wann genau ein (komplettes) neues Auto angefragt oder abgegeben wird. Gesucht ist also der Zeitpunkt, zu dem die Integrierten Polynome von Angebot und Nachfrage einen ganzzahligen Wert erreichen (zB. 1,2,3…). Dieser Zeitpunkt lässt sich ermitteln, wenn man die Nullstelle der integrierten Funktion verschoben nach unten ermittelt. Anschaulich:



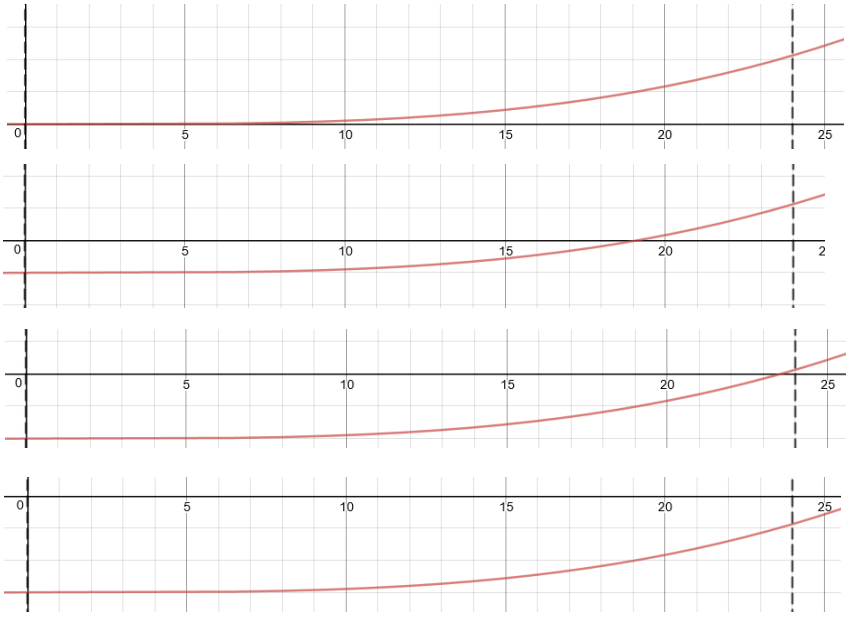


Es werden also zunächst einmal die Integrale der Polynome benötigt. Da gegeben ist, dass es sich immer um Polynome des max. 4. Grades handelt, kann man folgende Formel anwenden:

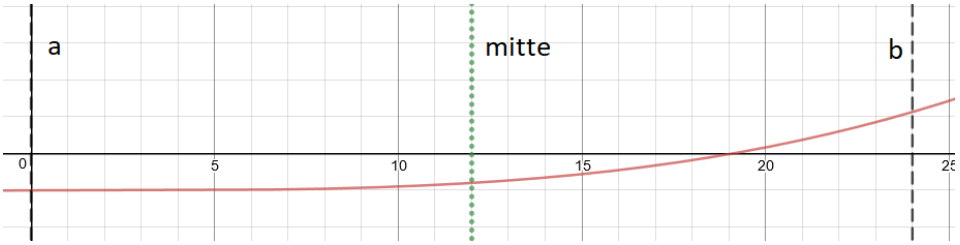
$$p(x) = a + bx + cx^2 + dx^3 + fx^4$$

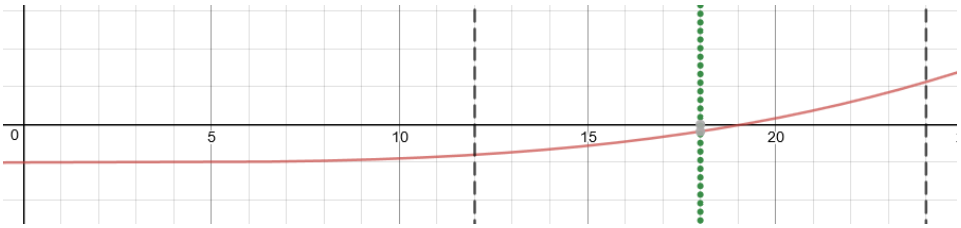
$$P(x) = ax + \frac{b}{2}x^2 + \frac{c}{3}x^3 + \frac{d}{4}x^4 + \frac{f}{5}x^5$$

Sind die Integrale bestimmt, so muss jeweils für P(x)-n die Nullstelle bestimmt werden. Wobei n bei 1 beginnt und solange um 1 erhöht wird bis keine Nullstelle mehr im Intervall [0.24] gefunden wird. Anschaulich wird die Funktion solange nach unten verschoben, bis sie komplett unterhalb der x Achse im Interval [0,24] verläuft:



Jede dieser Nullstellen ist also eine Änderung um +1 oder -1 in der Bedarfsfunktion (+1 bei Nachfrage und -1 bei Abstellung). Zur Ermittlung der Nullstellen (NST) wird das Bisektionsverfahren verwendet. Das Bisektionsverfahren sucht nach einer NST in einem





Intervall von a bis b. Ist ein Vorzeichenwechsel der Funktionswerte innerhalb dieses Intervalls gegeben so wird die Mitte zwischen a und b ermittelt. Durch das Berrechnen des mittleren Funktionswertes ist nun klar auf welcher Seite von der Mitte ausgesehen die NST liegen muss (rechts wenn zwischen der Mitte und b ein Vorzeichenwechsel vorliegt, sonst links). Je nachdem wo das Vorzeichen liegt, wird a und b neu gesetzt und das Intervall in dem sich die Nullstelle befindet, wurde halbiert.

Man hört auf wenn man genau auf der Nullstelle landet oder die Gewünschte genauigkeit erreicht hat. In dem Anwendungsfall Carsharing Simulation kann das intervall immer zwischen 0 und 24 beginnen. Für die Anzahl der benötigten Iterationen gilt deshalb diese Formel:

$$Anzahl\ Iterationen = \frac{\ln(\frac{24}{genauigkeit})}{\ln(2)}$$

3.3 Simulation bzw. Berechnung von Endzustand und maximalem Bedarf

Um den Endzustand zu ermitteln, muss zunächst wie im vorherigen Modul beschrieben, die Bedarfsfunktion ermittelt werden. Sind für jeden Teil der Stadt (also für jedes der m² Quadrate) die Bedarfsfunktionen ermittelt, so lässt sich der Endzustand bestimmen indem man von den Bedarfsfunktionen den Funktionswert zum Zeitpunkt 24 ausliesst. Die Gesamtheit der Funktionswerte zum Zeitpunkt 24 ist dann der Endzustand der Simulation.

Um den maximalen Bedarf einer Bedarfsfunktion zu ermitteln, wird das Maximum aus allen Ermittelten Funktionswerten einer Bedarfsfunktion ermittelt. Es wird also über alle Änderungen iteriert und der größte Funktionswert gespeichert.

Um den Simulationsverlauf zu ermitteln werden alle Änderungen der Bedarfsfunktionen nach Zeit sortiert in einer Liste abgespeichert. Treten mehrere Änderungen zur gleichen Zeit auf, so hat die Änderung „Auto abstellen“ Vorrang vor der Änderung „Auto anfordern“.

3.4 Schreiben in Ausgabedatei

Sind die auszugebenden Daten generiert worden, so werden diese in Textform überführt und in der Ausgabedatei abgespeichert. Als Zusatz wird die erste Kommentarzeile (falls vorhanden) an den Anfang der Ausgabedatei als Beschreibung geschrieben. Nach der Beschreibung

folgt der Simulationsverlauf. Zum Schluss werden Noch der Endzustand und der maximale Bedarf für jeden Teil der Stadt in die Datei geschrieben.

3.5 Datenstruktur

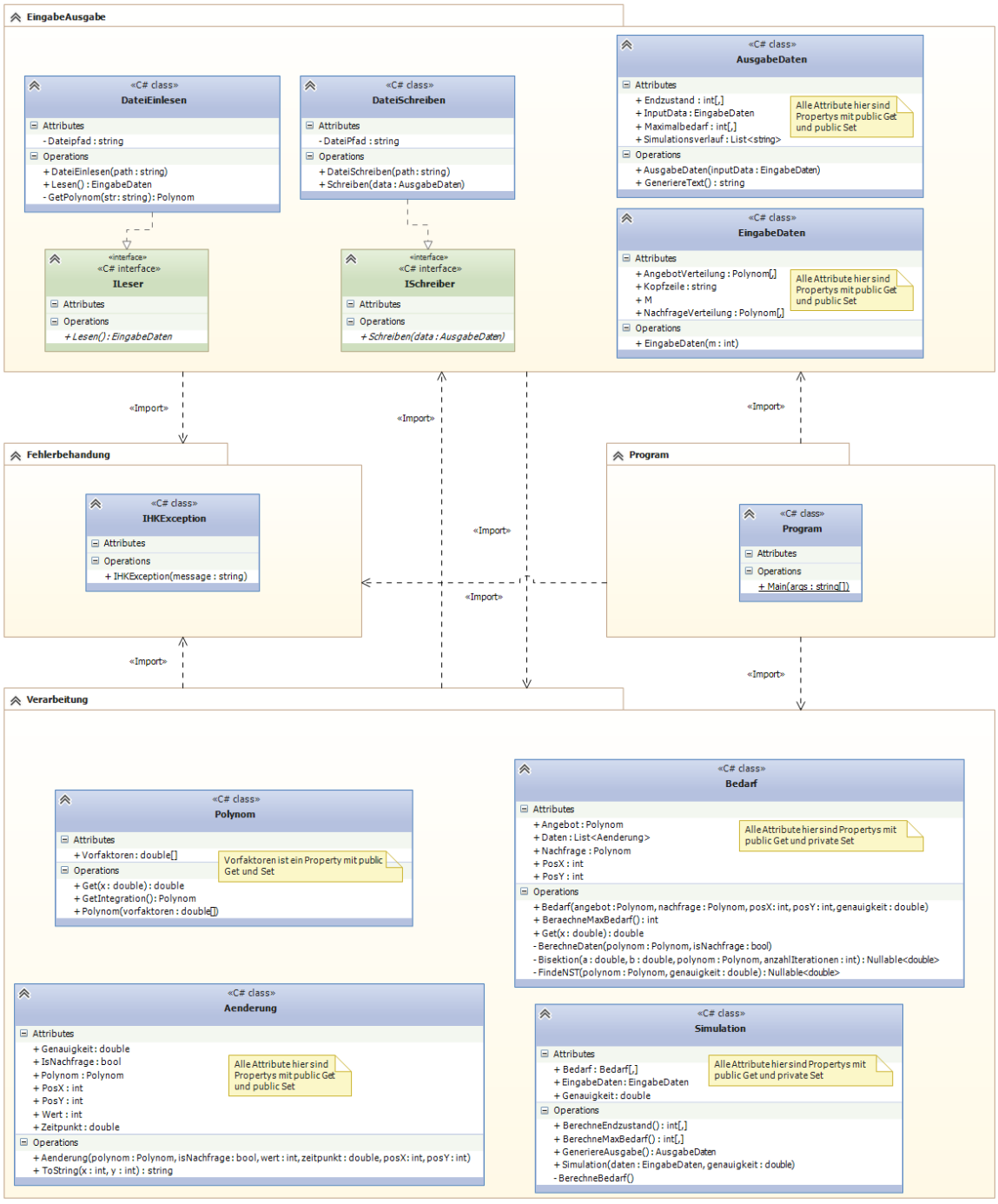
Die eingelesenen Daten werden in einer Klasse gespeichert, die als Attribute den ersten Kommentar der Eingabedatei, m, und die Polynome für Angebot und Nachfrage enthält. Zum Speichern der Polynome wird eine eigene Klasse erstellt. Sie enthält die Vorfaktoren des Polynoms. Die Angebots- und Nachfrage-Polynome werden in einem 2D Feld gespeichert wobei die Position, an der sich die Polynome befinden auch die Position des Teils der Stadt ist. Die Bedarfsfunktionen, die aus den Polynomen ermittelt werden, werden in einer separaten klasse gespeichert. Diese enthält die zugehörigen Angebots- und Nachfrage-Polynome und eine Liste in der die Änderungen abgespeichert sind. Die Änderungen speichern das zugehörige Polynom, Zeitpunkt, Wert, Position und Typ des Polynoms (also ob Nachfrage oder Abstellung). Die ermittelten Daten werden schließlich in einer Klasse zusammengefasst, die den Simulationsverlauf, Endzustand und Maximalbedarf enthält.

3.6 Fehlerbehandlung

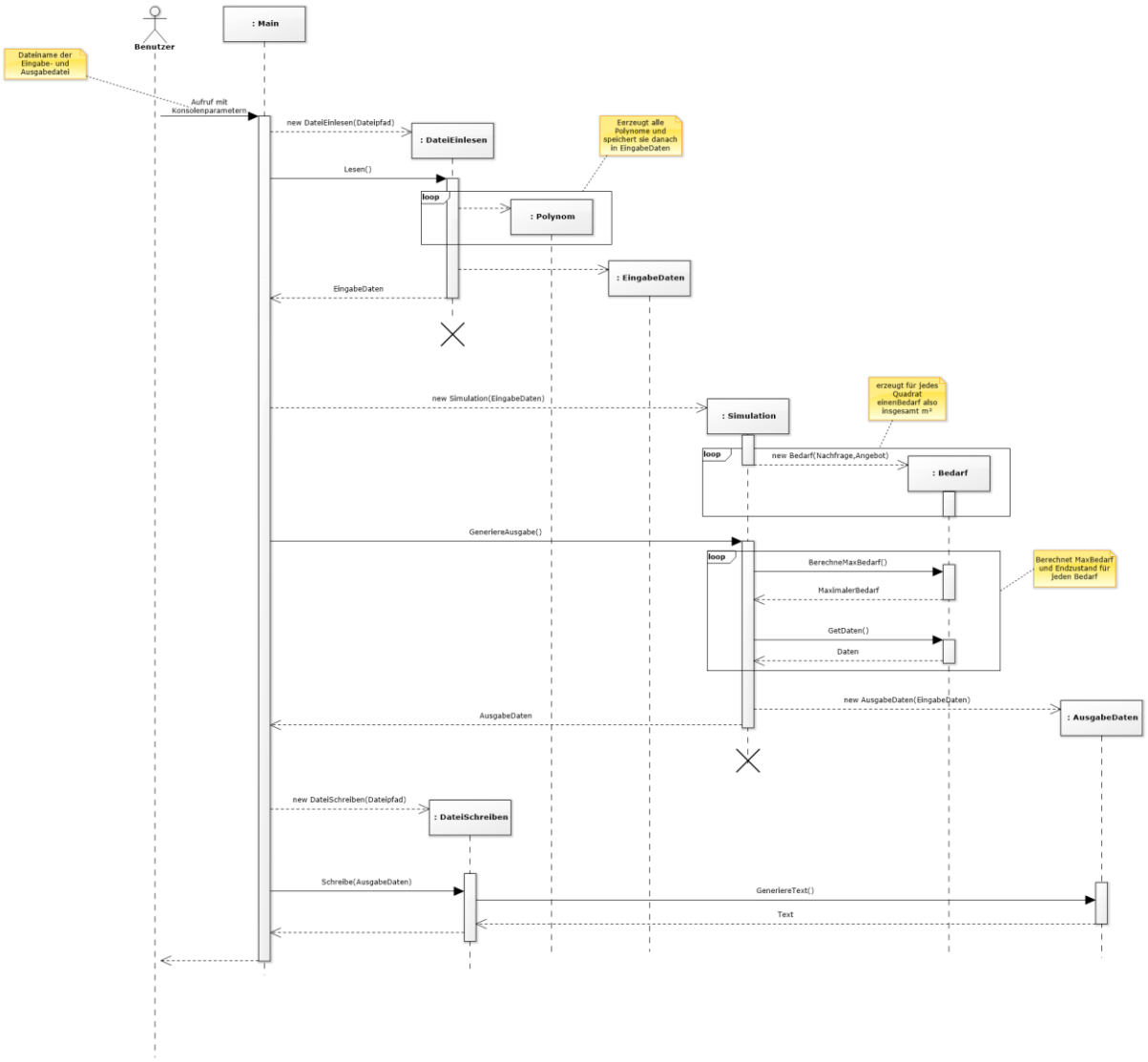
Wird wärend der Laufzeit des Programms festgestellt, dass ein Fehler vorliegt, weil z. B die eingegebenen Daten nicht richtig verarbeitet werden können, so wird eine Exception geworden, die eine Fehlermeldung an den Benutzer ausgibt und das Programm beendet.

4 Programmkonzeption

4.1 UML Klassendiagramm

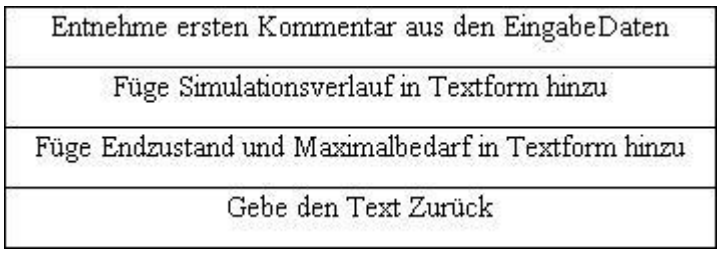


4.2 Programmablauf im Sequenzdiagramm

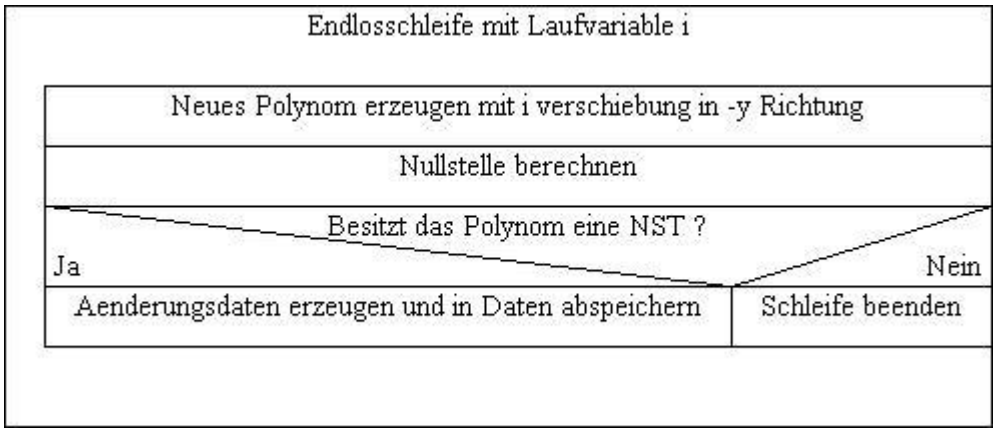


4.3 Nassi-Schneiderman-Diagramme

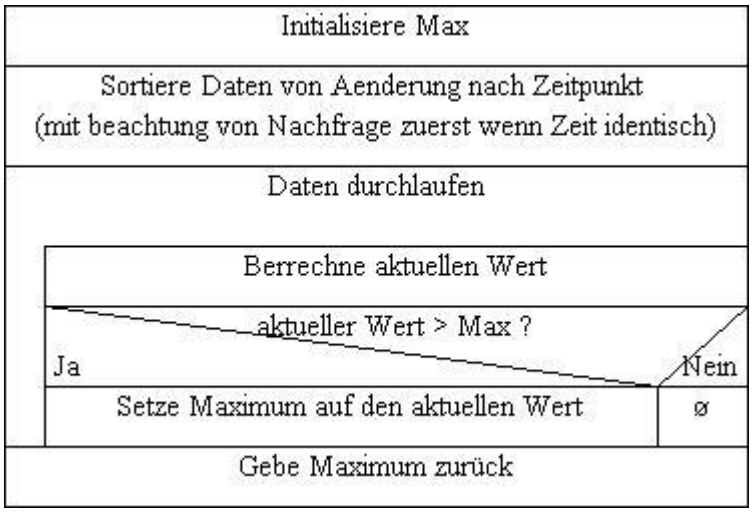
4.3.1 AusgabeDaten::GeneriereText()



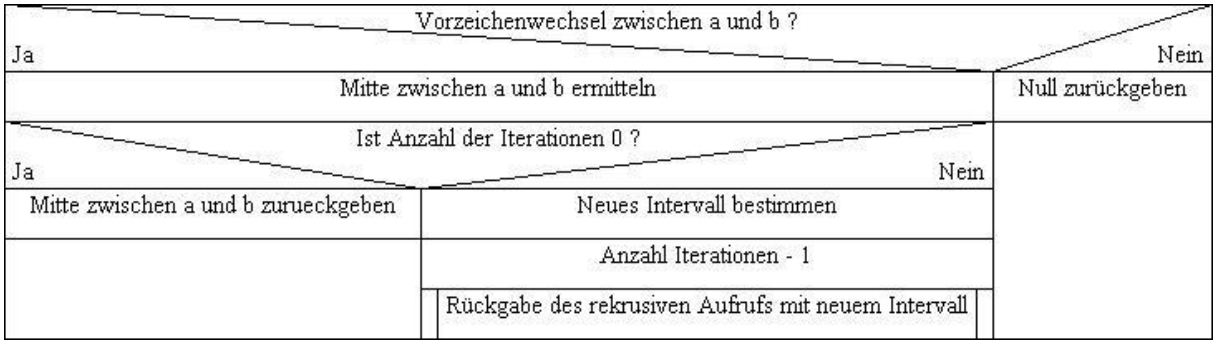
4.3.2 Bedarf::BerechneDaten(polynom,isNachfrage)



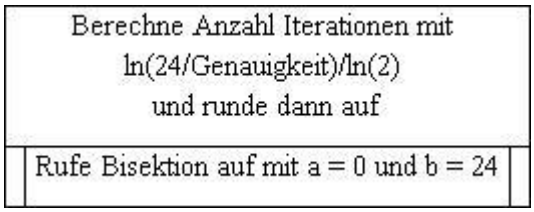
4.3.3 Bedarf::BerechneMaxBedarf()



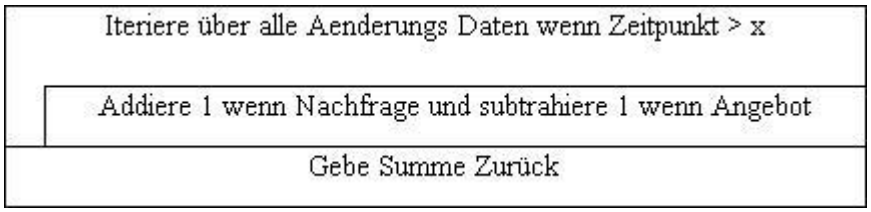
4.3.4 Bedarf::Bisektion(a,b,polynom,anzahlIterationen)



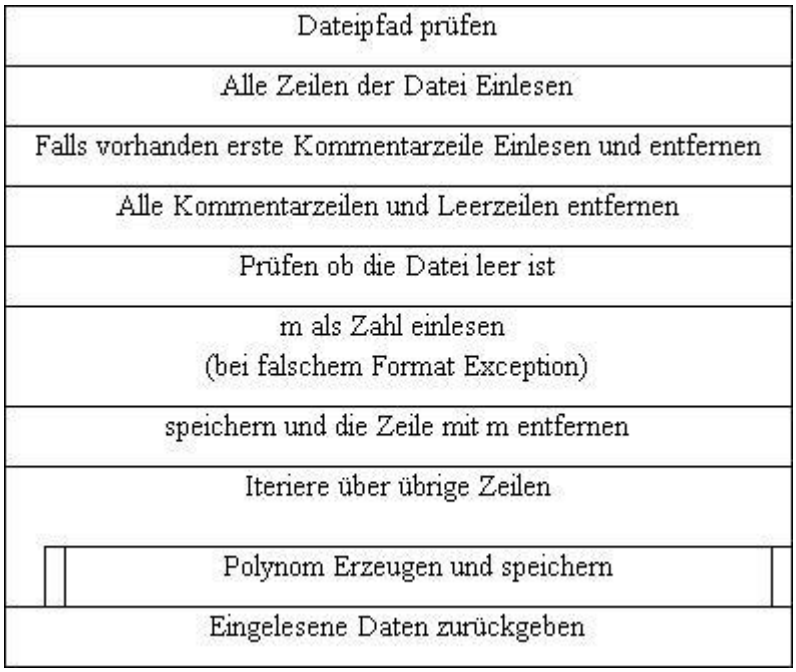
4.3.5 Bedarf::FindeNST(polynom,genauigkeit)



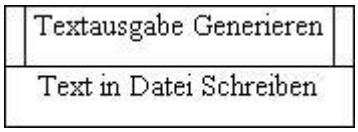
4.3.6 Bedarf::Get (x)



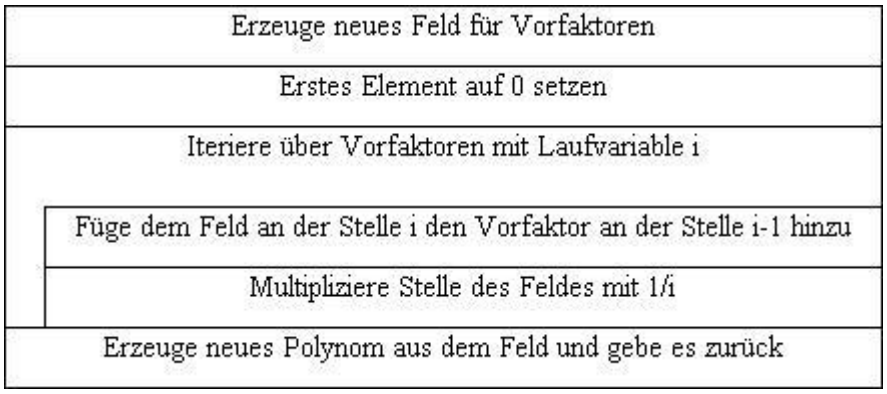
4.3.7 DateiEinlesen::Lesen ()



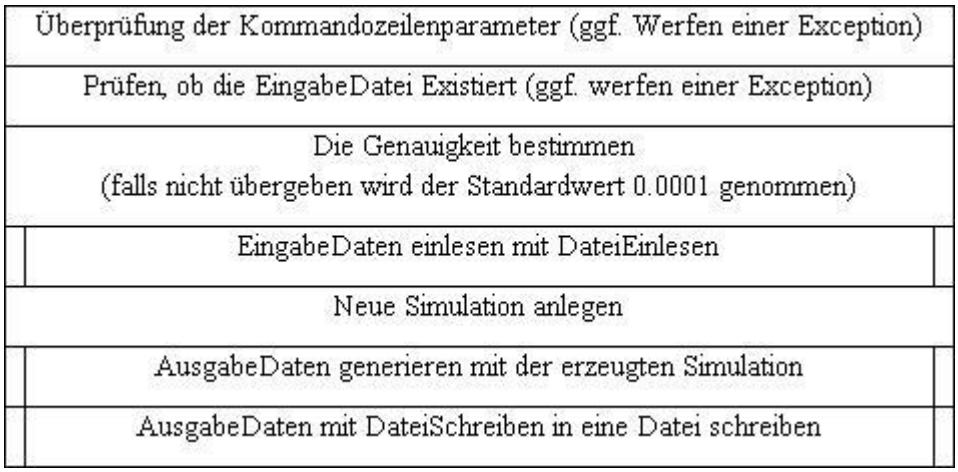
4.3.8 DateiSchreiben::Schreiben(AusgabeDaten)



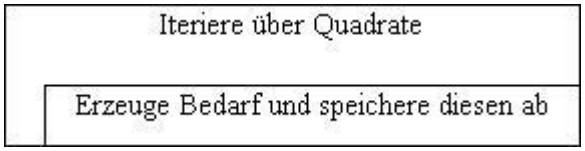
4.3.9 Polynom::GetIntegration()



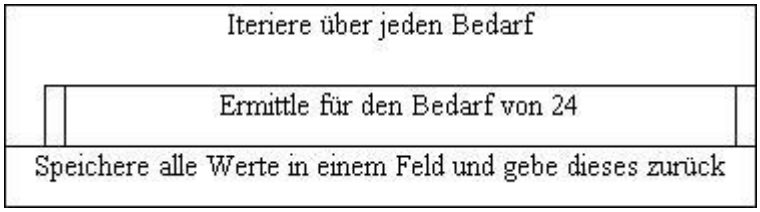
4.3.10 Program::Main (args)



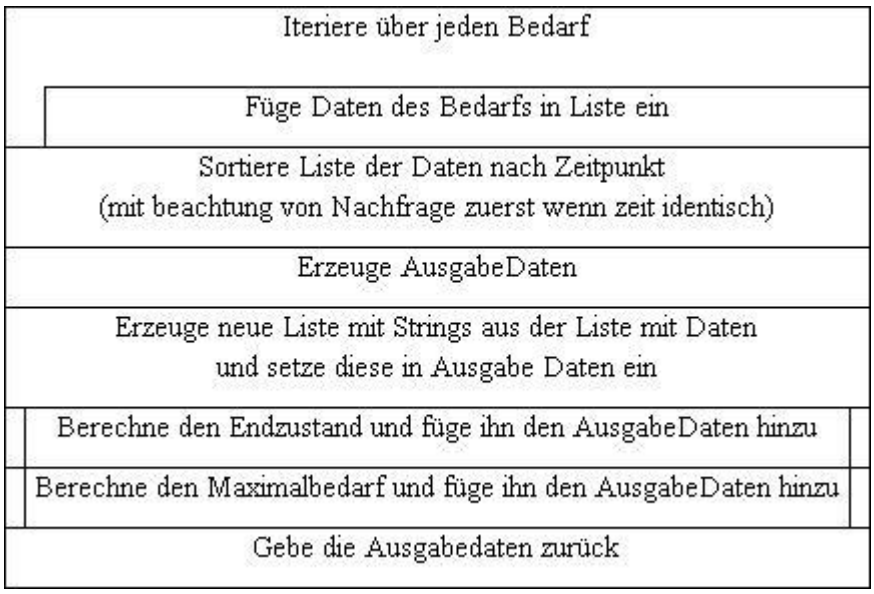
4.3.11 Simulation::BerechneBedarf ()



4.3.12 Simulation::BerechneEndzustand ()



4.3.13 Simulation::GeneriereAusgabe()



5 Abweichung von der handschriftlichen Ausarbeitung

Am ursprünglichen Konzept haben sich im Laufe der Bearbeitung einige kleinere Änderungen ergeben. Im Folgenden werden diese behandelt.

5.1 Prüfen der Eingabedatei in Methode Lesen

Im ursprünglichen Konzept ist in der Methode DateiEinlesen::Lesen() keine Prüfung enthalten, ob die Datei leer ist und ob der Dateipfad überhaupt gültig ist. Diese Funktionen sowie das Prüfen von m und der Polynome wurde ergänzt. Diese Änderung war notwendig, da so der Benutzer eine passende Fehlermeldung erhält mit der er die Fehler in der Eingabe korrigieren kann.

5.2 Anpassung der Formel für Berechnung der Iterationen

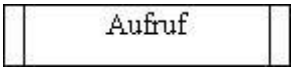
Im Ursprünglichen Konzept hat sich ein kleiner Fehler eingeschlichen. Die Formel mit der die Anzahl der Iterationen aus der Genauigkeit berechnet werden soll, beinhaltet einen Fehler. Die richtige Formel lautet:

$$Anzahl\ Iterationen = \frac{\ln(\frac{24}{genauigkeit})}{\ln(2)}$$

Zuvor wurde im Zähler der Formel $\ln(24 \times genauigkeit)$ berechnet, was durch einen Umformungsfehler entstanden ist.

5.3 Allgemeine Änderungen an Nassi-Schneiderman-Diagrammen

Im ursprünglichen Konzept wurden Funktionsaufrufe in den Diagrammen nicht eindeutig gekennzeichnet. In der aktuellen Version werden diese mit dem Folgenden Kästchen gekennzeichnet worden:



5.4 Änderungen an der Main Methode

Beim Anlauf der Main Methode hat sich soweit nichts geändert. Das Nassi-Schneidermann Diagramm wurde nur etwas angepasst, damit die Reihenfolge der im Code entspricht.

5.5 IHKException

Im ursprünglichen Konzept gab es im UML Klassendiagramm keine Exception (Im Konzept wurde aber dennoch erwähnt mit Exceptions zu arbeiten). Dies ist notwendig, um dem Benutzer eine ordentliche Fehlermeldung anzeigen zu können.

5.6 Bedarf Genauigkeit

Im ursprünglichen Konzept wurde die Genauigkeit als Parameter den Funktionen übergeben. Jetzt ist es so, dass die Genauigkeit in Bedarf eine Property der Klasse ist. So ist es möglich auch zu einem Späteren Zeitpunkt zu erkennen mit welcher Genauigkeit die Bedarfsfunktion ermittelt wurde.

5.7 Position

Im ursprünglichen Konzept wurde in der Änderung nicht die Position der Änderung gespeichert. Diese wird jedoch benötigt um den korrekten String für die Änderung bestimmen zu können. Daher wurden die Properties PosX und PosY hinzugefügt. Der Vollständigkeit halber wurde auch dem Bedarf die Properties PosX und PosY hinzugefügt.

6 Testfälle

Die Testfälle werden nach dem Prinzip des Black Box Testens ausgewählt. Die Tests werden also ohne Kenntnisse über die innere Funktionsweise des zu testenden Systems entwickelt. Die Testfälle werden also anhand der Anforderungen ausgewählt. Die genaue Beschaffenheit der Software wird nicht betrachtet, sondern nur ihre ein und ausgaben.

(Alle hier behandelten Testfälle befinden sich im Verzeichnis Testfälle und müssen in den Ordner Input verschoben werden um sie auszuführen.)

6.1 Normalfälle

Nachfolgend werden Normalfälle getestet. Normalfälle sind Fälle, die genau den Vorgaben entsprechen

6.1.1 IHK_Beispiel1.in

6.1.1.1 Eingabe

```
# AHausen
# m
1
# Polynome Nachfrage
0 0.2023761 -0.0287711 0.0016925 -0.0000352
# Polynome Abstellungen
0.434782 0 0 0 0
```

6.1.1.2 Ausgabe

```
# AHausen
Abstellung in Q_11 zu t=2,30
Nachfrage in Q_11 zu t=3,76
Abstellung in Q_11 zu t=4,60
Nachfrage in Q_11 zu t=5,87
Abstellung in Q_11 zu t=6,90
Nachfrage in Q_11 zu t=7,86
Abstellung in Q_11 zu t=9,20
Nachfrage in Q_11 zu t=9,89
Abstellung in Q_11 zu t=11,50
Nachfrage in Q_11 zu t=11,96
Abstellung in Q_11 zu t=13,80
Nachfrage in Q_11 zu t=14,03
Nachfrage in Q_11 zu t=16,06
Abstellung in Q_11 zu t=16,10
Nachfrage in Q_11 zu t=18,06
Abstellung in Q_11 zu t=18,40
```

Nachfrage in Q_11 zu t=20,17
Abstellung in Q_11 zu t=20,70
Abstellung in Q_11 zu t=23,00
Nachfrage in Q_11 zu t=23,43
Endzustand des Tages:
0
Maximaler Bedarf:
1

6.1.1.3 Diskussion

Durch Vergleich mit dem Beispiel in der Aufgabenstellung fällt auf, dass die Ergebnisse vollständig übereinstimmen. Dieser Normalfall weist also keine Fehler oder Besonderheiten auf.

6.1.2 IHK_Beispiel2.in

6.1.2.1 Eingabe

```
# BStadt
# m
2
# Polynome Nachfrage
0 0 0 0 0
0.20833333333333334 0 0 0 0
0 0.03333333333333333 -0.001388888888888889 0 0
0 0 0 0 0
# Polynome Abstellungen
0 0 0 0 0
0 0.048 -0.002 0 0
0.14285714285714285 0 0 0 0
0.058823529411764705 0 0 0 0
```

6.1.2.2 Ausgabe

```
# BStadt
Nachfrage in Q_12 zu t=4,80
Abstellung in Q_21 zu t=7,00
Abstellung in Q_12 zu t=7,22
Nachfrage in Q_21 zu t=8,93
Nachfrage in Q_12 zu t=9,60
Abstellung in Q_12 zu t=10,94
Abstellung in Q_21 zu t=14,00
Nachfrage in Q_21 zu t=14,02
Nachfrage in Q_12 zu t=14,40
Abstellung in Q_12 zu t=14,45
Abstellung in Q_22 zu t=17,00
```

```
Abstellung in Q_12 zu t=18,53
Nachfrage in Q_12 zu t=19,20
Nachfrage in Q_21 zu t=20,35
Abstellung in Q_21 zu t=21,00
Nachfrage in Q_12 zu t=24,00
Endzustand des Tages:
0 1
0 -1
Maximaler Bedarf:
0 1
1 0
```

6.1.2.3 Diskussion

Durch Vergleich mit dem Beispiel in der Aufgabenstellung fällt auf, dass die Ergebnisse vollständig übereinstimmen. Dieser Normalfall weißt also keine Fehler oder Besonderheiten auf.

6.1.3 Normalfall-VersetzteGeraden.in

In diesem Test wird überprüft, was passiert, wenn die Polynome die gleiche Steigung besitzen, aber einen anderen y Achsenabschnitt.

6.1.3.1 Eingabe

```
# CHausen
# m
1
# Polynome Nachfrage
1 0.005 0 0 0
# Polynome Abstellungen
0 0.005 0 0 0
```

6.1.3.2 Ausgabe

```
# CHausen
Nachfrage in Q_11 zu t=1,00
Nachfrage in Q_11 zu t=1,99
Nachfrage in Q_11 zu t=2,98
Nachfrage in Q_11 zu t=3,96
Nachfrage in Q_11 zu t=4,94
Nachfrage in Q_11 zu t=5,91
Nachfrage in Q_11 zu t=6,88
Nachfrage in Q_11 zu t=7,85
Nachfrage in Q_11 zu t=8,81
Nachfrage in Q_11 zu t=9,76
Nachfrage in Q_11 zu t=10,71
Nachfrage in Q_11 zu t=11,66
```

```
Nachfrage in Q_11 zu t=12,60
Nachfrage in Q_11 zu t=13,54
Nachfrage in Q_11 zu t=14,48
Nachfrage in Q_11 zu t=15,41
Nachfrage in Q_11 zu t=16,33
Nachfrage in Q_11 zu t=17,26
Nachfrage in Q_11 zu t=18,17
Nachfrage in Q_11 zu t=19,09
Abstellung in Q_11 zu t=20,00
Nachfrage in Q_11 zu t=20,00
Nachfrage in Q_11 zu t=20,91
Nachfrage in Q_11 zu t=21,81
Nachfrage in Q_11 zu t=22,71
Nachfrage in Q_11 zu t=23,61
Endzustand des Tages:
24
Maximaler Bedarf:
24
```

6.1.3.3 Diskussion

Die beiden Polynome sind identisch bis auf ihren Startwert. Durch diese Verschiebung wird immer ein Auto mehr nachgefragt als abgegeben. Deshalb ist der Endzustand auch wie erwartet 24, da pro Stunde immer 1 Auto mehr abgegeben wird als angefragt.

6.2 Sonderfälle

Nachfolgend werden Sonderfälle getestet. Diese enthaltenen Besonderheiten, also Gegebenheiten, die bei Normalfällen in der Regel nicht auftreten.

6.2.1 Sonderfall-LangeZahlen.in

In diesem Test wird überprüft, ob das Programm mit sehr langen Gleitkommazahlen umgehen kann. Dazu wird eine Zahl des Beispiels IHKBeispiel2 verlängert.

6.2.1.1 Eingabe

```
# BStadt
# m
2
# Polynome Nachfrage
0 0 0 0 0
0.20833333333333334 0 0 0 0
0 0.03333333333333333333333333333333333333333333333333333333333333333333333333 -
0.001388888888888889 0 0
0 0 0 0 0
```

```
# Polynome Abstellungen
0 0 0 0 0
0 0.048 -0.002 0 0
0.14285714285714285 0 0 0 0
0.058823529411764705 0 0 0 0
```

6.2.1.2 Ausgabe

```
# BStadt
Nachfrage in Q_12 zu t=4,80
Abstellung in Q_21 zu t=7,00
Abstellung in Q_12 zu t=7,22
Nachfrage in Q_21 zu t=8,93
Nachfrage in Q_12 zu t=9,60
Abstellung in Q_12 zu t=10,94
Abstellung in Q_21 zu t=14,00
Nachfrage in Q_21 zu t=14,02
Nachfrage in Q_12 zu t=14,40
Abstellung in Q_12 zu t=14,45
Abstellung in Q_22 zu t=17,00
Abstellung in Q_12 zu t=18,53
Nachfrage in Q_12 zu t=19,20
Nachfrage in Q_21 zu t=20,35
Abstellung in Q_21 zu t=21,00
Nachfrage in Q_12 zu t=24,00
Endzustand des Tages:
0 1
0 -1
Maximaler Bedarf:
0 1
1 0
```

6.2.1.3 Diskussion

Die Ausgabe zeigt, dass das Programm mit längeren Gleitkommazahlen umgehen kann. Das Ergebnis ist dasselbe wie beim Testfall IHKBeispiel2.in.

6.2.2 Sonderfall-LeerZeilen.in

In diesem Test wird überprüft, ob Leerzeichen in der Eingabedatei das Programm behindern. Dazu wird IHKBeispiel1 abgeändert, sodass einige Leerzeichen eingefügt werden.

6.2.2.1 Eingabe

```
# AHausen

# m
```

```
1

# Polynome Nachfrage

0 0.2023761 -0.0287711 0.0016925 -0.0000352


# Polynome Abstellungen



0.434782 0 0 0 0
```

6.2.2.2 Ausgabe

```
# AHausen
Abstellung in Q_11 zu t=2,30
Nachfrage in Q_11 zu t=3,76
Abstellung in Q_11 zu t=4,60
Nachfrage in Q_11 zu t=5,87
Abstellung in Q_11 zu t=6,90
Nachfrage in Q_11 zu t=7,86
Abstellung in Q_11 zu t=9,20
Nachfrage in Q_11 zu t=9,89
Abstellung in Q_11 zu t=11,50
Nachfrage in Q_11 zu t=11,96
Abstellung in Q_11 zu t=13,80
Nachfrage in Q_11 zu t=14,03
Nachfrage in Q_11 zu t=16,06
Abstellung in Q_11 zu t=16,10
Nachfrage in Q_11 zu t=18,06
Abstellung in Q_11 zu t=18,40
Nachfrage in Q_11 zu t=20,17
Abstellung in Q_11 zu t=20,70
Abstellung in Q_11 zu t=23,00
Nachfrage in Q_11 zu t=23,43
Endzustand des Tages:
0
Maximaler Bedarf:
1
```

6.2.2.3 Diskussion

Die Ausgabe zeigt, dass das Programm mit Leerzeichen und Zeilenumbrüchen umgehen kann. Das Ergebnis ist dasselbe wie beim Testfall IHKBeispiel1.in.

6.2.3 Sonderfall-Grenzen.in

In diesem Test wird überprüft, was passiert, wenn eine Änderung bzw. eine Nullstelle genau auf der Grenze, also bei 24 liegt.

6.2.3.1 Eingabe

```
# CHausen
# m
1
# Polynome Nachfrage
0.08333 0 0 0 0
# Polynome Abstellungen
0.083333333333333333333333333333333333333333333333 0 0 0 0
```

6.2.3.2 Ausgabe

CHausen

```
# CHausen
Abstellung in Q_11 zu t=12,00
Nachfrage in Q_11 zu t=12,00
Abstellung in Q_11 zu t=24,00
Endzustand des Tages:
-1
Maximaler Bedarf:
0
```

6.2.3.3 Diskussion

Es wurden 2 Polynome übergeben. Das Polynom für Abstellungen hat eine Änderung eigentlich bei genau 24 vom Programm berechnet, wird allerdings 23,9999542236328. Für den Benutzer wird diese Zahl bis auf 2 Nachkommastellen gerundet und dann ausgegeben. Das andere Polynom für die Nachfrage hat eine Änderung bei knapp über 24 (in etwa 24,001). Bei einer Ausgabe würde hier also auch auf 24,00 gerundet werden, die Änderung wird jedoch nicht als solche angezeigt. Dies ist jedoch ein Verhalten, das erwartet wird, da nur Änderungen im Intervall von 0 bis 24 berücksichtigt werden sollen.

6.2.4 Sonderfall-IdentischePolynome.in

In diesem Test wird überprüft, was passiert, wenn die Polynome für Nachfrage und Angebot identisch sind. Dieser Test soll zeigen, dass bei einer gleichzeitigen Änderung das Angebot, also die Abstellung, vor der Nachfrage Vorrang hat.

6.2.4.1 Eingabe

```
# CHausen
# m
```

```
1
# Polynome Nachfrage
0 0.005 0 0 0
# Polynome Abstellungen
0 0.005 0 0 0
```

6.2.4.2 Ausgabe

```
# CHausen
Abstellung in Q_11 zu t=20,0000152587891
Nachfrage in Q_11 zu t=20,0000152587891
Endzustand des Tages:
0
Maximaler Bedarf:
0
```

6.2.4.3 Diskussion

Die Ausgabe zeigt, dass die Abstellung tatsächlich vor der Nachfrage erfolgt obwohl die beiden Änderungen zur selben Zeit geschehen. Das Verhalten entspricht also den Anforderungen.

6.2.5 Sonderfall-GroßeZahlen.in

In diesem Test wird überprüft, was passiert, wenn schnell wachsende Polynome übergeben werden.

6.2.5.1 Eingabe

```
# CHausen
# m
1
# Polynome Nachfrage
1000 0 0 0 0
# Polynome Abstellungen
900 0 0 0 0
```

6.2.5.2 Ausgabe

```
# CHausen
Nachfrage in Q_11 zu t=0,00
Abstellung in Q_11 zu t=0,00
Nachfrage in Q_11 zu t=0,00
Abstellung in Q_11 zu t=0,00
Nachfrage in Q_11 zu t=0,00
Abstellung in Q_11 zu t=0,00
…
Nachfrage in Q_11 zu t=24,00
```

```
Abstellung in Q_11 zu t=24,00
Nachfrage in Q_11 zu t=24,00
Endzustand des Tages:
2400
Maximaler Bedarf:
2400
```

6.2.5.3 Diskussion

Die Ausführung des Programms nimmt einige Zeit in Anspruch, da sehr viele Nullstellen berechnet werden müssen. Die Ausgabedatei ist auch dementsprechend groß. Sie umfasst in diesem Fall 45605 Zeilen und ist 1,30 MB groß (weshalb die Ausgabe auch nicht vollständig hier abgebildet ist). Es fällt auf, dass sehr viele Änderungen zeitnah geschehen – die ersten 9 Änderungen sind, wenn auf 2 Nachkommastellen gerundet, identisch. Sollen solche Polynome effektiv berechnet werden, so sollte über die Wahl eines anderen Algorithmus nachgedacht werden, der vielleicht nicht so exakte Ergebnisse liefert, dafür aber größere Funktionen bearbeiten kann.

6.2.6 Sonderfall-GroßeStadt.in

In diesem Test wird überprüft, was passiert, wenn eine große Stadt simuliert werden soll. (Da sowohl die Eingabe- als auch Ausgabedatei sehr groß ausfällt, sind hier nur Teile davon abgebildet)

6.2.6.1 Eingabe

```
# BStadt
# m
100
# Polynome Nachfrage
0 0 0 0 0
0 0 0 0 0
0.20833333333333334 0 0 0 0
0 0.03333333333333333 -0.001388888888888889 0 0
0 0 0 0 0
…
```

6.2.6.2 Ausgabe

```
# BStadt
Nachfrage in Q_8243 zu t=4,80
Nachfrage in Q_2593 zu t=4,80
Nachfrage in Q_3668 zu t=4,80
Nachfrage in Q_518 zu t=4,80
Nachfrage in Q_9533 zu t=4,80
Nachfrage in Q_628 zu t=4,80
…
```

6.2.6.3 Diskussion

Man erhält hier ein ähnliches Verhalten wie bei dem Sonderfall „Sonderfall-GroßeZahlen.in“, werden nur genügend eigentlich gut berechenbare Funktionen übergeben, so wird auch hier die Anzahl an benötigten Rechenoperationen schnell so groß, dass die Berechnung gerne etwas länger dauern kann. In diesem Testfall wurden $100^2 + 100^2$ also 20000 Funktionen übergeben, weshalb schon die Eingabedatei sehr groß ist (Die Ausgabedatei ist noch etwas größer mit 1,01 MB). Bei so vielen unabhängigen Berechnungen bietet es sich gut an, den Algorithmus zu parallelisieren.

6.3 Fehlerfälle

Nachfolgend werden Fehlerfälle getestet. Diese Fälle können vom Programm nicht bearbeitet werden und erzeugen eine Fehlermeldung in der „ErrorLog.txt“ Datei.

6.3.1 Fehlerfall-LeereDatei.in

Dieser Test soll zeigen, dass eine leere Datei die entsprechende Fehlermeldung erzeugt. (Eine Datei gilt auch als leer, wenn nur Kommentare in der Datei sind)

6.3.1.1 Eingabe

```
# Diese Datei ist leer
```

6.3.1.2 Fehlermeldung

```
ERROR: Die Eingabedatei ist leer
```

6.3.1.3 Diskussion

Die erwartete Fehlermeldung wurde ausgegeben.

6.3.2 Fehlerfall-MKleiner1.in

Dieser Test soll zeigen, dass falls m kleiner als 1 ist, eine Fehlermeldung erzeugt wird.

6.3.2.1 Eingabe

```
# AHausen
# m
-1
# Polynome Nachfrage
0 0.2023761 -0.0287711 0.0016925 -0.0000352
# Polynome Abstellungen
0.434782 0 0 0 0
```

6.3.2.2 Fehlermeldung

```
ERROR: m="-1" hat das falsche Format (sollte eine positive Ganzzahl groesser 0 sein)
```

6.3.2.3 Diskussion

Die erwartete Fehlermeldung wurde ausgegeben.

6.3.3 Fehlerfall-PolynomFalschesFormat.in

Dieser Test soll zeigen, dass Polynome im falschen Format erkannt werden. (Falsches Format wenn die Anzahl an Vorfaktoren ungleich 5 ist)

6.3.3.1 Eingabe

```
# AHausen
# m
1
# Polynome Nachfrage
0 0.2023761    -0.0287711     0.0016925 -0.0000352
# Polynome Abstellungen
0.02 0.05 0 0
```

6.3.3.2 Fehlermeldung

```
ERROR: Die Angabe des Polynoms "0.02 0.05 0 0" liegt im falschen Format vor
```

6.3.3.3 Diskussion

Die erwartete Fehlermeldung wurde ausgegeben. (Beim Nachfragepolynom werden die Leerzeichen automatisch korrigiert)

6.3.4 Fehlerfall-FalscheAnzahlPolynome.in

Dieser Test soll zeigen, dass eine Fehlermeldung ausgegeben werden soll falls m und die Anzahl der angegebenen Polynome nicht übereinstimmen (genau m^2+m^2 Polynome müssen angegeben werden)

6.3.4.1 Eingabe

```
# AHausen
# m
1
# Polynome Nachfrage
0 0.2023761 -0.0287711 0.0016925 -0.0000352
0 0 0 0 0
# Polynome Abstellungen
0.434782 0 0 0 0
0 0 0 0 0
```

6.3.4.2 Fehlermeldung

```
ERROR: Es sind nicht genuegend Polynome angegeben worden (2 erwartet und 4 gegeben)
```

6.3.4.3 Diskussion

Die erwartete Fehlermeldung wurde ausgegeben.

6.3.5 Fehlerfall-GleitkommazahlM.in

Dieser Testfall soll abdecken, dass keine Gleitkommazahlen für m angenommen werden.

6.3.5.1 Eingabe

```
# AHausen
# m
1.5
# Polynome Nachfrage
0 0.2023761 -0.0287711 0.0016925 -0.0000352
# Polynome Abstellungen
0.434782 0 0 0 0
```

6.3.5.2 Fehlermeldung

```
ERROR: m="1.5" hat das falsche Format (sollte eine positive Ganzzahl groesser 0 sein)
```

6.3.5.3 Diskussion

Die erwartete Fehlermeldung wurde ausgegeben.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde eine Carsharing Simulation entwickelt, mit deren Hilfe sich ein Ablauf für Angebot und Nachfrage von Autos, sowie Endzustand und maximal benötigte Autos pro Stellplatz berechnen lassen. Die Software funktioniert einwandfrei und wurde ausgiebig getestet.

Durch Verwendung von Interfaces können die Eingabe und Ausgabe beliebig ausgetauscht werden. Es ist z. B. denkbar, statt von einer Datei die Daten aus einer GUI (Benutzerschnittstelle), einer Datenbank oder anderen Datenquellen zu lesen.

Zur Optimierung des Programmes ließe sich z. B. die Berechnung der Nullstellen parallelisieren. Die Berechnungen der einzelnen Bedarfsfunktionen sind voneinander unabhängig, was besonders gut für Parallelisierung geeignet ist.

Das Programm hat wie bereits in der Aufgabenanalyse angesprochen Probleme mit Funktionen, die eine sehr steile Steigung besitzen. Bei solchen Problemen würde es Sinn machen, nicht jede Nullstelle zu berechnen. Man könnte also zu Lasten der Genauigkeit den Algorithmus beschleunigen, um auch größere Probleme lösen zu können.

8 Entwicklungsumgebung

8.1 Hardware

Entwickelt wurde auf einem ASUS ROG GL552VW mit folgender Ausstattung:

System

Hersteller:	ASUSTek Computer Inc.
Modell:	GL552VW
Prozessor:	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz 2.59 GHz
Installierter Arbeitsspeicher (RAM):	8,00 GB
Systemtyp:	64-Bit-Betriebssystem, x64-basierter Prozessor

8.2 Betriebssystem

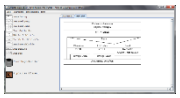
Als Betriebssystem kam Windows 10 Home in der 64-Bit Version zum Einsatz.

8.3 Entwicklungsumgebung

Als IDE (*integrated development environment*) wurde Microsoft Visual Studio Enterprise 2015 in der Version 14.0 Update 3 verwendet.

8.4 Diagramme

Das UML Klassendiagramm wurden in Microsoft Visual Studio Enterprise 2015 in einem Modellierungsprojekt erstellt (Aus dem UML Klassendiagramm kann man Code generieren, jedoch das Diagramm nicht updaten lassen, wenn Änderungen im Code stattfinden). Die Struktogramme (bzw. Nassi-Schneiderman-Diagramme) wurden mit dem Programm Stucktogrammeditor erstellt (http://www.whiledo.de/index.php?p=struktogrammeditor). Zur Erstellung des Sequenzdiagramms wurde das Programm SoftwareIdeasModeler verwendet.



software ideas modeler
DUSANRODINA

8.5 Dokumentation

Zum Erstellen dieser Dokumentation wurde Microsoft Word 2016 verwendet. Die HTML-Entwicklerdokumentation wurde mit doxygen generiert.

doxygen

9 Erklärung

Erklärung des Prüfungsteilnehmers:

Ich erkläre verbindlich, dass das vorliegende Prüfungsprodukt von mir selbstständig erstellt wurde. Die als Arbeitshilfe genutzten Unterlagen sind in der Arbeit vollständig aufgeführt. Ich versichere, dass der vorgelegte Ausdruck mit dem Inhalt der von mir erstellten digitalen Version identisch ist. Weder ganz noch in Teilen wurde die Arbeit bereits als Prüfungsleistung vorgelegt. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung des Prüfungsprodukts als Prüfungsleistung ausschließt.

Aachen, 18.05.2017 Felix Heitbrock

Ort und Datum Unterschrift