Dokumentation zur IHK-Abschlussprüfung Sommer 2020

Entwicklung eines Softwaresystems

**Geschrieben von: Felix Warschewski**

**Prüflingsnummer 101 20580**

**Tag der Einreichung: 26.06.2020**

**Programmiersprache: C#**

Inhaltsverzeichnis

[1 Eigenhändigkeitserklärung 3](#_Toc43896132)

[2 Änderungen zu Tag 1 4](#_Toc43896133)

[3 Beschreibung der Zufallszahlengeneratoren und Verfahren 4](#_Toc43896134)

[4 Diskussion der Güte-Testverfahren und der Parameter für den LCG 5](#_Toc43896135)

[5 Diskussion der Testbeispiele 6](#_Toc43896136)

[6 Benutzeranleitung 7](#_Toc43896137)

[6.1 Laufzeitumgebung 7](#_Toc43896138)

[6.2 Installation 7](#_Toc43896139)

[6.3 Dateistruktur 7](#_Toc43896140)

[6.4 Dateiformat 7](#_Toc43896141)

[6.5 Aufrufbefehle 8](#_Toc43896142)

[7 Vergleich und Interpretation der Ergebnisse 8](#_Toc43896143)

[8 Zusammenfassung und Ausblick 8](#_Toc43896144)

[9 Programmcode im Anhang 9](#_Toc43896145)

# Eigenhändigkeitserklärung



# Änderungen zu Tag 1

Sämtliche Variablen und Eigenschaften mit dem Datentyp long wurden in double umgeändert. Dies betreffen vor allem die Einstellparameter, sowie Rechnungen und hat den Vorteil, dass sämtliche mathematische Funktionen nicht umgewandelt werden müssen.

Zur Speicherung der Beispiel-LCGs verwende ich eine Konstanten-Klasse. In dieser werden die Beispielparameter für die LCGs als statische Eigenschaften festgelegt.

Darüber hinaus habe ich anstatt eine einfache Verteilungsklasse eine Verteilungsschnittstelle implementiert, welche zwei Unterklassen (Gleichverteilung, Standardnormalverteilung) hat. Diese Unterklassen besitzen eine Funktion transformiere und in der transformiere-Funktion der Standardnormalverteilt-Klasse wird die Polarmethode angewendet.

# Benutzeranleitung

## Laufzeitumgebung

Das verwendete Betriebssystem ist Windows 10 64-bit Version. Die Programmiersprache ist C# (Version C# 7.3) und es wurde in der Entwicklungsumgebung Visual Studio 2017 Enterprise Edition geschrieben. Das Zielframework der Konsolen-Applikation ist das .NET Core 2.1.

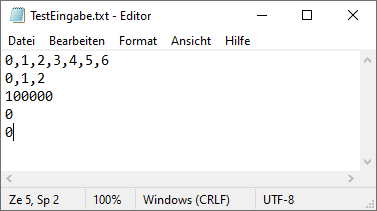
## Installation

Die Installation erfolgt über das Extrahieren der vorhandenen zip-Datei.

## Dateistruktur

## Dateiformat

Mithilfe des args-Parameter in der Main-Methode kann ein Dateipfad übergeben werden. Dieser Pfad sollte auf eine Textdatei zeigen, in der Daten zur Bedienung der Konsolenanwendung stehen.



Die erste Zeile beschreibt die Zufallszahlengeneratoren, welche überprüft werden sollen. Die zweite Zeile beschreibt die Güte-Testverfahren, die angewendet werden sollen. Die dritte Zeile legt fest, wie hoch die Sequenzlänge ist, also wie viele Zufallszahlen für die Prüfungen erstellt werden. Die vierte Zeile legt ein k fest. Bei der seriellen Autokorrelation ist das k die Ordnung der Autokorrelation und steht für den Abstand der verglichenen Paare. Bei dem Sequenz-Up-Down-Test bestimmt das k die Länge einer bestimmten Bitfolge. Gibt man 0 für k an, erstellt die serielle Autokorrelation einen

zufälligen Abstand k und bei dem Sequenztest wird über alle k iteriert und letztlich die Differenz ausgegeben. Die letzte Zeile gibt die Art der Verteilung der Zufallszahlen an. Man kann unterscheiden zwischen standardnormalverteilten Zufallszahlen und gleichverteilten Zufallszahlen. Hierbei sind gleichverteilte Zufallszahlen der Standard.

## Aufrufbefehle

Innerhalb der aufzurufenden Textdatei:

Zeile 1 – Generatoren:

1. Ansi-C
2. Minimal Standard
3. RANDU
4. SIMSCRIPT
5. NAG’s LCG
6. Maple’s LCG
7. Datumsbasiert(eigener)

Zeile 2 – Güte-Testverfahren

1. Serielle Autokorrelation
2. Sequenz-Up-Down-Test
3. Eigener Güte-Test

Zeile 3 – Sequenzlänge

Zeile 4 – k

Wenn k = 0 dann werden die allgemeinen Bedingungen übernommen. Siehe 6.4 Dateiformat. Wenn k != 0 dann wird die k-te Ordnung des Verfahrens berechnet und zurückgegeben.

# Beschreibung der Zufallszahlengeneratoren und Verfahren

Zu implementieren waren folgende Generatoren und Testverfahren:

* Linearer Kongruenz-Generator mit 6 verschiedenen Einstellungsmöglichkeiten
* Umwandlung gleichverteilter in normalverteilte Zufallszahlen mithilfe der Polar-Methode
* Eigener Zufallszahlengenerator (Datumsbasiert)
* Güte-Testverfahren Serielle Autokorrelation
* Güte-Testverfahren Sequenz-Up-Down-Test
* Eigenes Güte-Testverfahren

Der Lineare Kongruenz-Generator (LCG) errechnet Zufallszahlen auf Basis der Kongruenzrechnung. Hierfür sind 4 Eingabeparameter wichtig. Das Modul m, der Multiplikator a, das Inkrement c und der Startwert x0. Des Weiteren waren sechs zu implementierenden Verfahren für den LCG in Tabellenform gegeben. Diese Verfahren unterscheiden sich in der Wahl ihrer Einstellungsparameter. Der LCG funktioniert, wie eine Folge, wobei jeder Aufruf eine andere und zufällige Zahl erstellen soll.

Der LCG erzeugt vor Allem gleichverteilte Zufallszahlen. Deswegen soll eine weitere Verteilung in Betracht gezogen werden. Mithilfe der **Polar-Methode** sollen gleichverteilte Zufallszahlen in standardnormalverteilte Zufallszahlen transformiert werden. Hierbei werden euklidische Koordinaten, welche aus dem Zufallsgenerator kommen, in Polarkoordinaten umgewandelt und wenn die beiden Koordinaten im Punkt zusammen im Einheitskreis liegen, werden sie zu zwei unabhängigen standardnormalverteilten Zufallszahlen transformiert. Dieses Prinzip der Verteilungstransformation kann auf jedes Verfahren des Linearen Kongruenz-Generators angewendet werden.

Der eigene Zufallszahlengenerator funktioniert auf Basis der Zeit und Laufzeit des Programmes. Es werden die Sekunden, Minuten, Stunden, Tage, Monate als Integer aufsummiert, sowie die aktuelle Laufzeit des Programmes in Millisekunden. Darüber hinaus wird gespeichert, wie viele Zufallszahlen bereits mit einer Instanz des Generators erzeugt wurden. Die Summe wird mit der Anzahl der bereits erzeugten Zufallszahlen multipliziert. Das Ergebnis wird anschließend mit Modulo 1000 gerechnet

und durch die 1000 geteilt. So ergibt sich eine Zufallszahl zwischen 0 und 1 mit 3 Nachkommastellen. Das bedeutet sie kann bis 1000 hochskaliert werden generierte Nullen angehängt zu bekommen. Will man höher skalieren benötigt man ein höheres Modul.

Das Güte-Testverfahren mithilfe der seriellen Autokorrelation misst die Abhängigkeit einer Folge von Zahlen. Die Zufallszahlen sollen paarweise möglichst unabhängig sein und im Optimalfall eine Korrelation von p=0 aufweisen. Das Verfahren verwendet eine Sequenz von Zufallszahlen, einen Mittelwert berechnet aus dieser Sequenz und eine Ordnung k. Je nachdem wie hoch die Ordnung ist, prüft man auf weiter entferntere oder näher aneinanderlegende Paare in der Sequenz. Die Wahl von k kann das Ergebnis des Güte-Tests stark beeinflussen. Daher sollte man zum Testen von bestimmten Generatoren die Verfahren mehrmals mit unterschiedlichen Ordnungen durchlaufen lassen. Nur so erhält man ein gültiges Ergebnis.

Das nächste Güte-Testverfahren ist der Sequenz-Up-Down-Test. Dieser prüft wie viele Folgen bestimmter Länge von stetig größer oder kleiner werdenden Zahlen in einer Zufallszahlensequenz vorhanden sind. Es wird eine Bit Maske erstellt mit den Werten 0 und 1. Der Wert 0 wird in die Maske eingefügt, sobald ein folgender Wert kleiner ist als der jetzige. Wenn der folgende Wert hingegen größer als der derzeitige ist, wird eine 1 in die Bit Maske eingefügt. Das Ergebnis des Verfahrens gibt einen Aufschluss darüber, wie viele Folgen es mit einer bestimmten Länge gab. Je höher die Zahl bei niedrigen Werten für k (Kettenlänge) ist, desto besser funktioniert Zufallszahlengenerator, da die Werte öfter hin und her springen und sich nicht stetig aufbauen.

Das eigene Güte-Testverfahren erstellt auch eine Bitmaske. Hier werden alle Werte gleich dem Sequenz-Up-Down-Verfahren in die Bitmaske geschrieben. Anschließend wird die Bitmaske aufsummiert und durch die Anzahl der vorhandenen Bits der Bitmaske dividiert. Dies gibt Aufschluss über den Mittelwert der Wechsel der Zahlenfolgen. Wenn der Mittelwert deutlich unter 0.5 liegt, ist das ein Zeichen für sehr wenige Wechsel zwischen längeren Folgen. Das bedeutet die Güte des Generators eher schlecht wäre, als wenn der Mittelwert über 0.5 liegt. Je höher desto öfter wechselt er innerhalb der Zufallszahlensequenz und desto zufälliger werden die Zahlen generiert.

# Diskussion der Güte-Testverfahren und der Parameter für den LCG

Es werden sechs verschiedene Verfahren für den Linearen Kongruenz-Generator vorgestellt. Diese unterscheiden sich teils stark und haben deswegen auch eine andere Abfolge zur Erzeugung von Zufallszahlen. Um zwei Beispielverfahren aus der Menge der sechs gegebenen Verfahren zu wählen, verwende ich den Ansi-C LCG und den RANDU LCG. Beide haben ein Modul von 2 hoch 31. Der Ansi-C hat einen deutlich höheren Multiplikator von 1103515245 im Vergleich zu dem RANDU-Multiplikator von 65539. Das Inkrement und der Startwert bei dem Ansi-C ist auch höher. Allein dies gibt eine Aussage darüber, wie schnell der Algorithmus den Wert des Moduls erreicht und überschlägt. Wenn er den Wert des Moduls übertrifft fängt er wieder bei 0 an und zählt sich nach oben. Daher kann man sagen, dass der Ansi-C deutlich schneller sein Modul erreicht und einen Wechsel von größeren und kleineren Zahlen besitzt. Der RANDU hingegen braucht länger und besitzt mehr Folgen von aufeinander größer werdenden Zufallszahlen. Dies sagt aus, dass für das Sequenz-Up-Down-Verfahren der Ansi-C LCG eine bessere Güte für eine kleine Sequenzlänge besitzt. Sobald aber der Parameter x eine gewisse Größe erreicht hat, werden die Sprünge immer größer und das Ergebnis

überschlägt öfter das Modul. Dies bedeutet, dass die Werte öfter variieren und die Folgen von aufeinanderfolgenden größer werdenden Zufallszahlen deutlich kleiner werden. Daher sollte man die beiden Zufallszahlengeneratoren mit mehreren beliebig großen Sequenzlängen testen. Es wird also angenommen, je länger die Sequenzlänge desto besser ist die Güte für den RANDU LCG im Up-Down-Test. Der Ansi-C sollte hingegen schon relativ früh eine gute Güte besitzen und keine drastische Verbesserung bei steigender Sequenzlänge erhalten.

Wenn man nun die beiden LCGs RANDU und Ansi-C hinsichtlich des seriellen Autokorrelation-Testverfahrens vergleicht und die bisher genannten Aspekte betrachtet, besitzt der Ansi-C für eine kleine Sequenzlänge und ein klein gewähltes k (Abstand verglichener Zahlen in Sequenz) eine deutlich schlechtere Güte als der RANDU. Der RANDU ist anfangs nicht so sehr gestreut und besitzt kleinere Abweichungen zum Mittelwert. Der Ansi-C besitzt deutlich größere Abweichungen, da er viel stärker skaliert. Wenn man den Abstand der verglichenen Zufallszahlen erhöht, kann das Phänomen auftreten, dass der Ansi-C wieder besser wird, da die Zahlen recht willkürlich werden. Hingegen kann es beim RANDU LCG passieren, dass die Werte, bei höherem k, deutlich auseinandergehen, da sie nicht so schnell den Wert des Moduls erreichen und wieder von vorne anfangen.

Die Vermutung besagt also, dass mit der seriellen Autokorrelation für eine kleine Sequenzmenge mit kleinem k (Abstand der verglichenen Zufallszahlen) der RANDU LCG eine bessere Güte besitzt. Aber sobald der Abstand der verglichenen Zufallszahlen größer wird, nimmt die Güte von RANDU ab und der Ansi-C wird besser.

Im Endeffekt kann man aber sagen: Je höher die Sequenzlänge, desto besser wird die Güte der Zufallszahlengeneratoren. Das „Lag“ k kann bei verschiedenen LCGs die Wirkung auf die Güte variabel beeinflussen.

Bei dem Maple’s LCG kommt es aufgrund von den Eingabeparametern immer zu einem Wechsel von ungeraden und geraden Zahlen nach dem Erreichen des Moduls. Da der Startwert, sowie der Multiplikator ungerade ist, entstehen anfangs nur ungerade Zahlen. Sobald das Produkt größer dem ungeraden Modul wird, entstehen gerade Zahlen. Dies wiederholt sich immer wieder. Das bedeutet, dass bis zum ersten Erreichen des ersten Moduls nur der halbe Zahlenraum zur Verfügung steht und nicht der komplette. Dies wird erst beim zweiten Erreichen des Moduls ausgeglichen. Dies ist ein deutliches Muster in der Zufallszahlengenerierung und sollte bei der seriellen Autokorrelation mit kleinem k erkannt werden.

# Diskussion der Testbeispiele

Die Testbeispiele geben Aufschluss über die Güte der Zufallszahlengeneratoren. Deswegen muss man einen differenzierten Blick auf die Testverfahren werfen und kann nicht nur ein Testverfahren mit einer bestimmten Sequenzlänge und einer bestimmten Ordnung verwenden. Es müssen mehrere unterschiedliche Testvorgänge entstehen, welche sich möglichst unterscheiden.

Aus den Annahmen aus Unterpunkt 4 werden die verschiedenen Punkte zwischen dem Ansi-C und dem Randu LCG verglichen, wobei jeweils eine differenzierte Sequenzlänge angegeben ist. Hierbei unterscheidet man auch zwischen den beiden Testverfahren Autokorrelation und Up-Down-Test. Es muss auch zwischen den Abständen der verglichenen Zufallszahlen unterschieden werden, um über die Güte der Zufallszahlengeneratoren zu entscheiden.

Es können durch die Methoden

1. Alle LCG-Verfahren mit Autokorrelation mit Sequenzlänge 10
2. Alle LCG-Verfahren mit Up-Down-Test mit Sequenzlänge 10
3. Alle LCG-Verfahren mit Autokorrelation mit Sequenzlänge 10
4. Alle LCG-Verfahren mit Up-Down-Test mit Sequenzlänge 10
5. Alle LCG-Verfahren mit Autokorrelation mit Sequenzlänge 10
6. Alle LCG-Verfahren mit Up-Down-Test mit Sequenzlänge 10
7. Ansi-C vs Randu Up-Down-Test mit Sequenzlänge 20
8. Ansi-C vs Randu Up-Down-Test mit Sequenzlänge 200
9. Ansi-C vs Randu Up-Down-Test mit Sequenzlänge 2000
10. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 20 und k = 5
11. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 200 und k = 50
12. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 2000 und k = 500
13. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 20 und k = 10
14. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 200 und k = 100
15. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 2000 und k = 1000
16. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 20 und k = 15
17. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 200 und k = 150
18. Ansi-C vs Randu Autokorrelation mit Sequenzlänge 2000 und k = 1500

Zeile 5 – Verteilung

1. Gleichverteilung
2. Standardnormalverteilung

# Vergleich und Interpretation der Ergebnisse

Erörterung des Einflusses der verwendeten Parameter der LCGs oder der Sequenzlänge bezüglich der Güte.

# Zusammenfassung und Ausblick

In weiteren Schritten könnte man die Klassenbibliothek um weitere LCG-Schemas erweitern. Damit stellt man eine breitere Funktionalität des LCG-Verfahrens bereit. Darüber hinaus kann man mehr Verteilungen an die Verteilungs-Schnittstelle anbinden. Dies ermöglicht die Transformation von Koordinaten oder Zahlen in andere Verteilungen. Dafür müsste man lediglich eine neue Klasse

anlegen mit der dazugehörigen „Transformiere()“-Methode der Verteilungsschnittstelle. Des Weiteren kann man Klassen an die GüteTests-Schnittstelle anklemmen. Man kann mehr

Testverfahren benutzen um die Güte der verschiedenen Generatoren, welche erzeugt wurden, zu bestimmen. Dies kann man alles mit beliebigen Parametern für die Anzahl der zufallsgenerierten

Element (Sequenzlänge) und die Ordnung k der Funktion ausführen, indem die Berechne()-Methode aus dem Interface „GüteTests“ verwendet wird. Die drei Schnittstellen sind Hauptbestandteile um diverse Funktionalitäten zu erweitern. Neue Tests einzuführen, in andere Verteilungen umzuwandeln oder komplett andere Zufallszahlen zu erstellen.

# Programmcode im Anhang

In der Programmierung für das Güte-Testverfahren Sequenz-Up-Down die Differenz zwischen Nopt(k) und N(k) gebildet. Wenn negative Werte für den Algorithmus herauskommen, bedeutet dies, dass häufig der Wert von N(k) größer als der von dem optimalen Wert Nopt(k) gewesen ist. Andererseits wenn öfter kleiner Werte für N(k) herauskommen, als die von Nopt(k), ist die Differenz der beiden Werte positiv. Ein Ergebnis für eine starke Güte eines Zufallszahlengenerators wäre hier, wenn der Wert nahe 0 liegen würden. Wenn der Wert zu weit von 0 abweicht, ist die Güte relativ schlecht für den geprüften Zufallszahlengenerator.

Das Sequenz-Testverfahren wurde so implementiert, wenn man eine Zahl für k mitgibt, gibt er die Differenz der gegebenen Folgenlänge zurück. Wenn für k eine 0 mitgegeben wird, wird die Gesamtdifferenz über alle Folgenlängen zurückgegeben.