Entwicklung einer Klassenbibliothek zur Erzeugung autokorrelierter Zufallszahlen

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2017

Autor(en): Anthony Delay
Philipp Bütikofer
Betreuer: Prof. Dr. Andreas Rinkel
Lukas Kretschmar

Inhalt

1. Abstract

Das Simulieren von komplexen Prozessen gewinnt immer mehr an Wert für ein Unternehmen und bildet einen immer wichtigeren Bestandteil in der Evaluation von neuen Dienstleistungen. In der diskreten Ereignissimulation wird mit automatisch generierten Zufallszahlen gearbeitet, welche sich an eine gewünschte Randverteilung halten. Nun hat man festgestellt, dass die Ergebnisse dieser Simulationen von gemessenen Realdaten stark abweichen können. Der Grund liegt darin, dass sich in der realen Welt Autokorrelationen bilden können. Dieser Umstand wird innerhalb von Simulationstools nicht berücksichtigt.

Diese Studienarbeit befasst sich mit drei Aspekten, welche unser Vorgehen definieren. Zuerst wird die mathematische Grundlage der Autokorrelation und des autoregressiven Modells ARTA aufgezeigt. Anschliessend wird dieser ARTA-Prozess in Form von einer Klassenbibliothek umgesetzt. Die daraus entstehenden Zufallszahlen werden analysiert und ausgewertet. Im Zuge der Implementation wird ein Plug-In für die Simulationssoftware Simio erzeugt, welche die ARTA-Klassenbibliothek einbindet und das Generieren von autokorrelierten Zufallszahlen innerhalb von Simio ermöglicht. Zum Schluss wird dieses Plug-In verwendet, um eine konkrete Simulation zu speisen und ein Vergleich mit normalen Zufallszahlen zu ermöglichen.

Die realisierte Klassenbibliothek ist im Stande autokorrelierte Zufallszahlen zu erzeugen. Dabei ist der Grad der Autokorrelation über die Korrelationskoeffizienten steuerbar.
Die Auswertungen zeigen auf, dass die Zufallszahlen von Arta.Standard im gewünschten Bereich liegen. Als Referenz wurden Zufallszahlen mithilfe der Java-Implementation erzeugt und diese anschliessend mit den unseren Verglichen.
In Form von einfachen Simulationsmodellen wurde einerseits die Funktionalität des Simio-AddIns getestet, andererseits wurde ein erster Vergleich zwischen Simio-generierten Zahlen und ARTA-Zahlen getätigt. Bei diesen Vergleichen wird ersichtlich, dass es signifikante Unterschiede zwischen den beiden Generierungsarten entstehen.

2. Einführung und Motivation

In der diskreten Ereignissimulation werden Zufallszahlen zur Beschreibung von Arbeitsschritten und auftretenden Ereignissen benötigt. Standardmässig werden diese Zufallszahlen so erzeugt, dass sie keine Autokorrelationen (Abhängigkeiten) aufweisen.
Die Realität sieht jedoch anders aus[1]. Es hat sich gezeigt, dass in der Praxis häufig ebendiese Autokorrelationen auftreten. Aufgrund dieser Abhängigkeiten können simulierte und reale Ergebnisse stark voneinander abweichen. Im Rahmen der Studienarbeit HS2017/18 soll eine Klassenbibliothek (Arta.Standard) entwickelt werden, welche es ermöglicht, autokorrelierte Zufallszahlen zu erzeugen. Der Grad der Autokorrelation kann selbst definiert werden. Arta.Standard soll so implementiert werden, dass eine Einbindung in die Simulationssoftware Simio oder andere Simulationstools möglich ist.

3. Zugrundeliegende Arbeiten

Als Fundament für die vorliegende Studienarbeit dienen die Veröffentlichungen «Autoregressive to anything: Time-series input processes for simulation[2]» und «JARTA — A Java library to model and fit Autoregressive-To-Anything processes[3]».

[1] Pereira, D. et al.: Autocorrelation effects in manufacturing systems performance: a simulation analysis. In: Laroque, C. et al. (Hrsg.): Proceedings of the Winter Simulation Conference (WSC), Berlin, 9.–12. Dez. 2012, S. 123:1–123:12.
[2] Modeling and generating multivariate time-series input processes using a vector autoregressive technique, 10.1145/937332.937333
[3] JARTA — A Java library to model and fit Autoregressive-To-Anything processes, 10.1109/WSC.2013.6721508

Marne C. Cario und Barry L. Nelson. beschreiben den ARTA-Prozess auf der mathematischen Ebene. ARTA (Autoregressive-to-anything) stellt ein bewährtes Modell zur Erzeugung von zufällig generierten Zahlen, mit gegebener Randverteilung und einem Autokorrelation aufweisendem Muster dar. «JARTA — A Java library to model and fit Autoregressive-To-Anything processes» stellt eine Java Implementation vor, welche den ARTA-Prozess abbildet. Mit JARTA werden die Ansätze von ARTA in eine JAVA-Library abgebildet. An einem konkreten Beispiel einer Lagerhaussimulation zeigen Tobias Uhlig und Oliver Rose die Funktionsweise und Wichtigkeit der Abhängigkeiten, wenn es um das Modellieren von stochastischen Prozessen geht. Der Sourcecode von JARTA ist frei verfügbar.

4. Autokorrelation [bis 25.10.2017]

Dieser Abschnitt wird den Begriff der Autokorrelation, deren grundlegende Eigenschaften und Charakteristiken erläutern. Anschliessend wird auf die Bereiche, welche Autokorrelation aufweisen eingegangen. Zum Abschluss wird Autokorrelation anhand eines Beispiels aufgezeigt.

4.1 Definition

Autokorrelation setzt sich aus den Wörtern Auto und Korrelation zusammen. «Korrelation» beschreibt einen Zusammenhang zwischen mindestens zwei oder mehreren Merkmalen, Zuständen, Funktionen oder Ereignissen. Diese Merkmale können sich je nach Anwendungsgebiet sehr stark unterscheiden. Das Präfix «Auto» zeigt auf, dass die Funktion oder Reihe mit sich selbst korreliert. Dies bedeutet, dass ähnliche oder gleiche Muster erkennbar sind.
Bei Autokorrelation sind also die Werte einer Variable zum Zeitpunkt t_n mit den Werten derselben Variable in zeitlich vergangenen Perioden abhängig. Die Autokorrelation ist immer zeitabhängig. Der Zusammenhang zwischen Autokorrelation und Zeit kann in Form von Korrelationsfunktionen ausgedrückt werden. Eine Korrelationsfunktion zeigt an, wie viel Ähnlichkeit zwischen der ursprünglichen (t_n) und der, um eine Zeit t_{n+m}, verschobenen Folge besteht.

4.2 Korrelationskoeffizienten

Korrelation gilt als Mass eines Zusammenhangs. Dieses Mass kann numerisch in Form von Korrelationskoeffizienten ausgedrückt werden und beantwortet die Frage nach der Stärke und der Richtung des Zusammenhangs. Bei Korrelationskoeffizienten handelt es sich um Zahlen, welche in einem Intervall zwischen -1 und 1 liegen. Eine Korrelation die den Koeffizienten 1 aufweist wird als perfekte positive, bei -1 als perfekte negative Korrelation bezeichnet. Je weiter sich der Korrelationskoeffizient dem Wert 0 nähert, umso schwächer ist die Korrelation. Ein Korrelationskoeffizient mit dem Wert 0 bedeutet, dass keine Korrelation vorhanden ist und die Werte perfekt verteilt[4] sind.

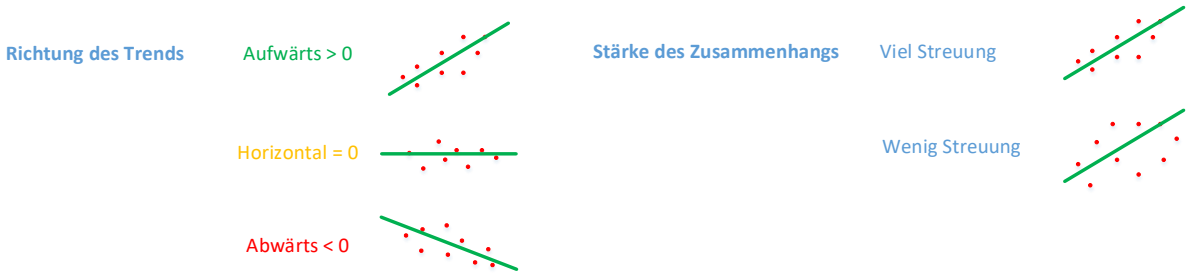


Abbildung 1: Korrelationskoeffizient

Im Zusammenhang mit dem Begriff Korrelationskoeffizient taucht der Ausdruck Pearson-Korrelation auf. Dieser ist nach Karl Person benannt, welcher das Mass der Korrelation in Form des Korrelationskoeffizienten

[4] Siehe Kapitel [x] zum Thema Verteilungen

in Zusammenarbeit mit Auguste Bravais entwickelt hat. Dies ist daher speziell erwähnenswert, da auf den Algorithmus von Pearson innerhalb der in dieser Arbeit erzeugten Klassenbibliothek zurückgegriffen wird.

4.3 Anwendungsbereiche

Autokorrelation kann durch mathematische Formeln ausgedrückt werden, jedoch wird sie in jedem Anwendungsbereich domänenspezifisch definiert. Als die signifikantesten Anwendungsgebiete gelten Statistik, Signalanalyse, Informationstheorie und die Softwaretechnik.

Autokorrelation in der Statistik: In der Statistik wird durch die Autokorrelation das Mass des Zusammenhangs zwischen zwei Zufallsvariablen beschrieben. Am häufigsten wird dieses Mass in Form der Korrelationskoeffizienten (Pearson) angegeben.

Autokorrelation in der Signalanalyse und Bildverarbeitung: In diesem Anwendungsgebiet wird eine Autokorrelationsfunktion genutzt, um die Korrelation eines Signales mit sich selbst zu unterschiedlichen Zeitverschiebungen eingesetzt. Somit kann beispielsweise der Zusammenhang zwischen Faltung und Autokorrelation aufgezeigt werden. In der Bildverarbeitung wird die zeitliche Komponente durch eine örtliche ersetzt. Dadurch lässt sich beispielsweise Objekterkennung realisieren.

Autokorrelation in der Softwaretechnik: Anwendung findet die Autokorrelation hier im sogenannten Korrelationstest. Dieser beschreibt ein Verfahren, welches die Plausibilität einzelner Parameter einer Funktion und deren Kombinationen überprüft.

Autokorrelation in der Informationstheorie: Durch Autokorrelation können in der Informationstheorie, insbesondere der Kryptographie, Analysen von Verschlüsselungsverfahren durchgeführt werden.

4.4 Partielle Korrelation

Unter der partiellen Korrelation versteht man das nicht-berücksichtigen von Dritteinflüssen. Eine Korrelation zwischen zwei statistischen Werten a und b kann unter Umständen auf einen gemeinsamen Faktor c zurückgeführt werden. Um diesen Effekt auszuschalten kann das Konzept der partiellen Korrelation eingesetzt werden. Durch eine partielle Korrelation wird der dritte Faktor entweder ausgeschaltet oder gezielt kontrolliert, so dass dieser das Resultat nicht verfälschen kann.

4.5 Durbin-Watson-Test

Die gebräuchlichste Methode um die Existenz von Autokorrelation zu belegen, stellt der Durbin-Watson-Test dar. Durch diesen statistischen Test kann geprüft werden, ob eine Autokorrelation der 1. Ordnung vorliegt. Autokorrelation 1. Ordnung bedeutet, dass aufeinanderfolgende Glieder der Reihe bzw. ihrer Residualgrössen[5] korrelieren. Das Ergebnis eines Durbin-Watson-Tests ist ein numerischer Wert im Bereich von 0 bis 4.

Wert des Tests	Korrelationskoeffizient	Bedeutung
d = 2	0	Keine Autokorrelation
d = 0	1	Perfekte positive Autokorrelation
d = 4	-1	Perfekte negative Autokorrelation

Der DW-Test ist durch den folgenden Term definiert:

$$d = \frac{\sum_{t=2}^{T}(\varepsilon_t - \varepsilon_{t-1})^2}{\sum_{t=1}^{T}\varepsilon_t^2}$$

Ausdruck	Bedeutung
$\sum_{t=2}^{T}$	Summiert alle Sequenzglieder zwischen t = 2 und T, wobei t und T die Menge aller Werte definieren. Die Anzahl der Beobachtungen entspricht dem Start und -Endwert der Zeitreihe.
$(\varepsilon_t - \varepsilon_{t-1})^2$	Entsprechen den Residuen/Werte der Reihe.
$\sum_{t=1}^{T}\varepsilon_t^2$	*[TODO]*

[5] Vertiefter Einblick Residuum: https://de.wikipedia.org/wiki/Residuum_(Statistik)

4.6 Beispiel Autokorrelation

Folgend dargestellt ist ein Beispiel zur Autokorrelation aus dem Bereich der Kryptographie. In der Kryptographie stellt die Autokorrelation eine Kennzahl für die Ähnlichkeit von Teilen eines Dokuments dar. Mithilfe der Autokorrelation kann unter Umständen die Schlüssellänge eines verschlüsselten Dokuments ermittelt werden.
Als Grundlage dient die Verschlüsselungsmethode Vigenère-Chiffre. Bei diesem Verfahren handelt es sich um ein Substitutionsverfahren, wobei der Klartext in Monogramme (einzelne Zeichen) zerlegt wird und diese anschliessend durch Geheimtextzeichen substituiert werden.
In diesem Beispiel wird vom Standardalphabet mit 26 Buchstaben ausgegangen. Daraus wird eine Matrix (Vigenère-Quadrat) erstellt, welches die 26 Buchstaben immer um eine Position verschoben darstellt.

Klartext

Schlüssel	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	**Geheimtext**
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	

Tabelle 1: Vigenère-Quadrat

Anschliessend muss ein Schlüssel gewählt werden. Dieser sollte möglichst lang und aus einer möglichst zufälligen Sequenz der Buchstaben des Alphabets bestehen. Der Kreuzungspunkt der einzelnen Buchstaben des Klartextes und des Schlüssels können nun innerhalb des Vigenère-Quadrats abgelesen werden und ergeben so das neue, verschlüsselte Zeichen.

Mithilfe der Software Cryptool[6] können solch einfache Verschlüsselungsverfahren aufgezeigt und analysiert werden. Cryptool verwendet folgende Autokorrelationsfunktion C(t), welche die Ähnlichkeit einer Folge[7] (s[i]) = s[1], s[2], …, s[n] und der um t Stellen verschobenen Folge (s[i+t]= s[1 + t], s[2 + t], s[n + t].

$$C(t) = \frac{(A(t) - D(t))}{n}$$

Wobei A(t) = Anzahl der übereinstimmenden Glieder der Folgen s[i] und s[i + t] im betrachteten Abschnitt und D(t) = Anzahl der nicht übereinstimmenden Glieder derselben Folgen und Abschnitt ist. n beschreibt die Länge der Sequenz.
Zur Veranschaulichung werden diese Formeln in ein Autokorrelationsdiagramm umgesetzt.

4.6.1 Beispiel 1 – starke Autokorrelation

Schlüssel:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Klartext:
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXY ZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWX YZ
Verschlüsselter Text:
ACEGIKMOQSUWYACEGIKMOQSUWYACEGIKMOQSUWYACEGIKMOQSUWYACEGIKMOQSUWYACEGIKM OQSUWYACEGIKMOQSUWYACEGIKMOQSUWYACEGIKMOQSUWYACEGIKMOQSUWYACEGIKMOQSUWYA CEGIKMOQSUWY

Dadurch, dass die Vigenère-Chiffre auf Substitution und Verschiebung der einzelnen Zeichen basiert, korrelieren die einzelnen Zeichen nach einer gewissen Zeit bzw. Verschiebung miteinander. Das erste, triviale, Beispiel zeigt eine sehr starke Autokorrelation, da der Klartext lediglich ein Vielfaches des Schlüssels ist.

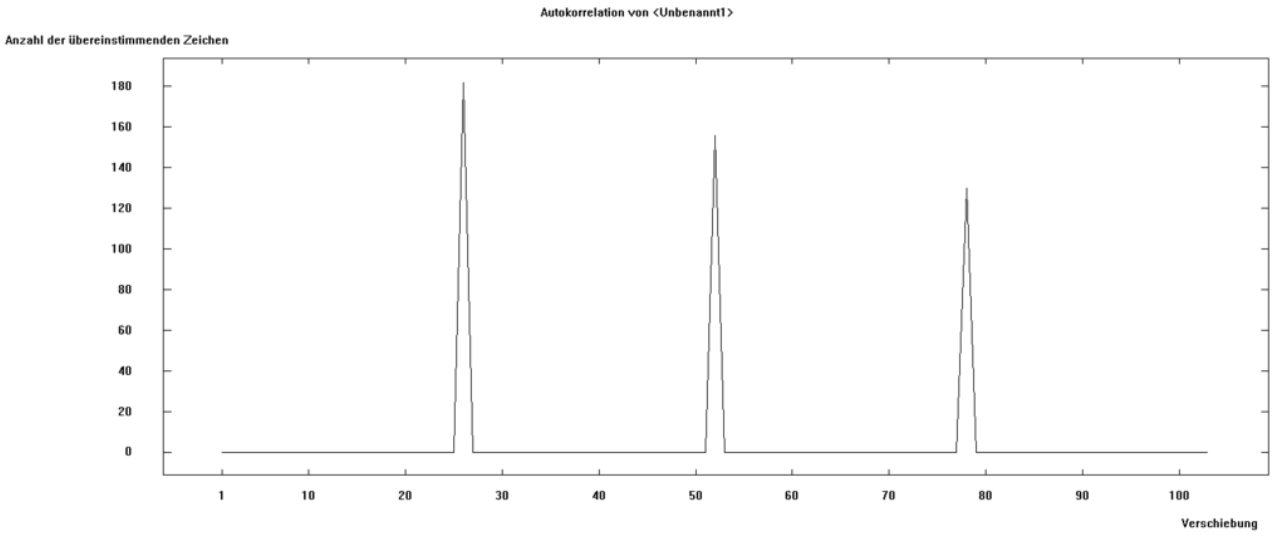


Abbildung 2: Autokorrelation des unverschlüsselten Textes, Bsp. 1

Wird nun die Autokorrelation des unverschlüsselten Textes betrachtet, so kann man die Verschiebung um 26 Zeichen klar erkennen.

[6] https://www.cryptool.org/de/cryptool1
[7] Die Indexierung der Folge ist 1 - basiert

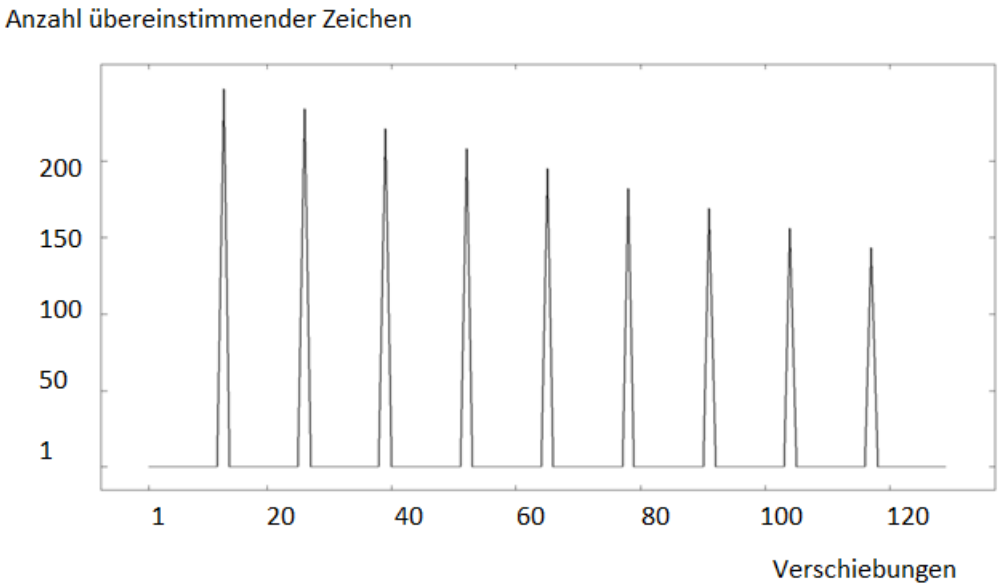


Abbildung 3: Autokorrelation verschlüsselter Text, Bsp.1

Man würde erwarten, dass auch beim verschlüsselten Text die Autokorrelation bei 26 Verschiebungen am stärksten ist. Jedoch ist dies ein Trugschluss. Die Autokorrelation ist bei 13 Verschiebungen deutlich am stärksten.
Weiter ist der Vergleich zwischen verschlüsseltem Text und des Klartextes spannend. Dort kann gesehen werden, dass «ABCDEFGHIJKLMNOPQRSTUVWXYZ» der Zeichenkette «ACEGIKMOQSUWYACEGIKMOQSUWY» entspricht. Es wird erkannt, dass genau nach 13 Zeichen wiederrum das Zeichen «A» auftaucht, was auf die obengenannte Struktur des Vigenère-Quadrat hinweist.

4.6.2 Beispiel 2

Schlüssel:
AUTOKORRELATION

Klartext: [8]
Die Giraffen sind eine Gattung der Säugetiere aus der Ordnung der Paarhufer. Ursprünglich wurde ihr mit Giraffa camelopardalis und der Trivialbezeichnung Giraffe nur eine einzige Art zugewiesen. Molekulargenetische Untersuchungen zeigen jedoch, dass die Gattung wenigstens vier Arten mit sieben eigenständigen Populationen umfasst. Die Giraffen stellen die höchsten landlebenden Tiere der Welt. Zur Unterscheidung vom verwandten Okapi werden sie auch als Steppengiraffen bezeichnet.

Verschlüsselter Text:
Dcx Usfrwjpn lqbq ecgs Qokkyyg wmf Fäuaxhssiv efs wmf Brxgixu uvv Aatzvhfyk. Ibggiürrlbkv julws svi dme Gbzosfu vowscftlrwizvs ogr nsi Kvtvbizoetxwmvelrr Gbzosfy gib szei pighwte Ukh jixvatelmb. Zofxyezrikpnxbwfcbx Ixhviwfcacbteh sssuve npdhkv, qaml rss Xrxeugo krnczgdsej ztek Ifgeh fwd gzvfpn xqurnmmäbnwxvr Aoiczntchbob ldjlslb. Rve Abfktwvr dtxtzrn xbs röqyjxpn eibqlyusxrve Xtekm rrr Qxzd. Nli Yytxzgphybrebx msx vxzknnxmsx Cbrtt wxzrrn mbs kity ews Lbscpygusfrwjpn umnriwaboh.

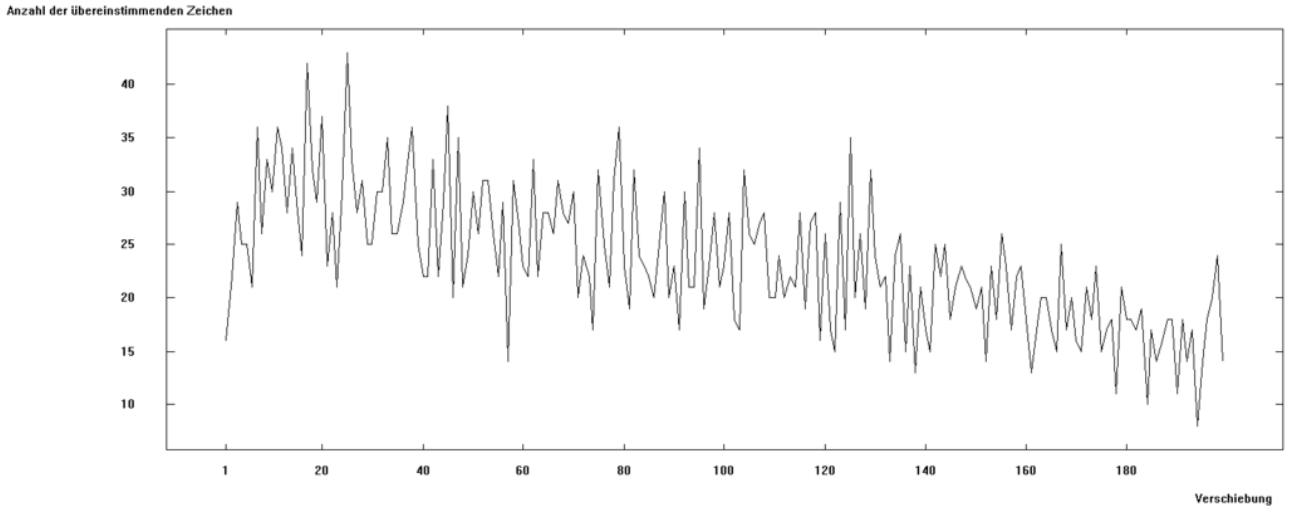


Abbildung 4 Autokorrelation des Klartextes

[8] Quelle : https://de.wikipedia.org/wiki/Giraffen

Auf den ersten Blick vermittelt dieses Diagramm einen willkürlichen Eindruck. Jedoch können auch hier autokorrelierte Strukturen erkannt werden. Diese sind nicht mehr nach einer fixen Anzahl Zeichen erkennbar wie in Beispiel 1, trotzdem sind vereinzelte, sehr starke Korrelationen ersichtlich.

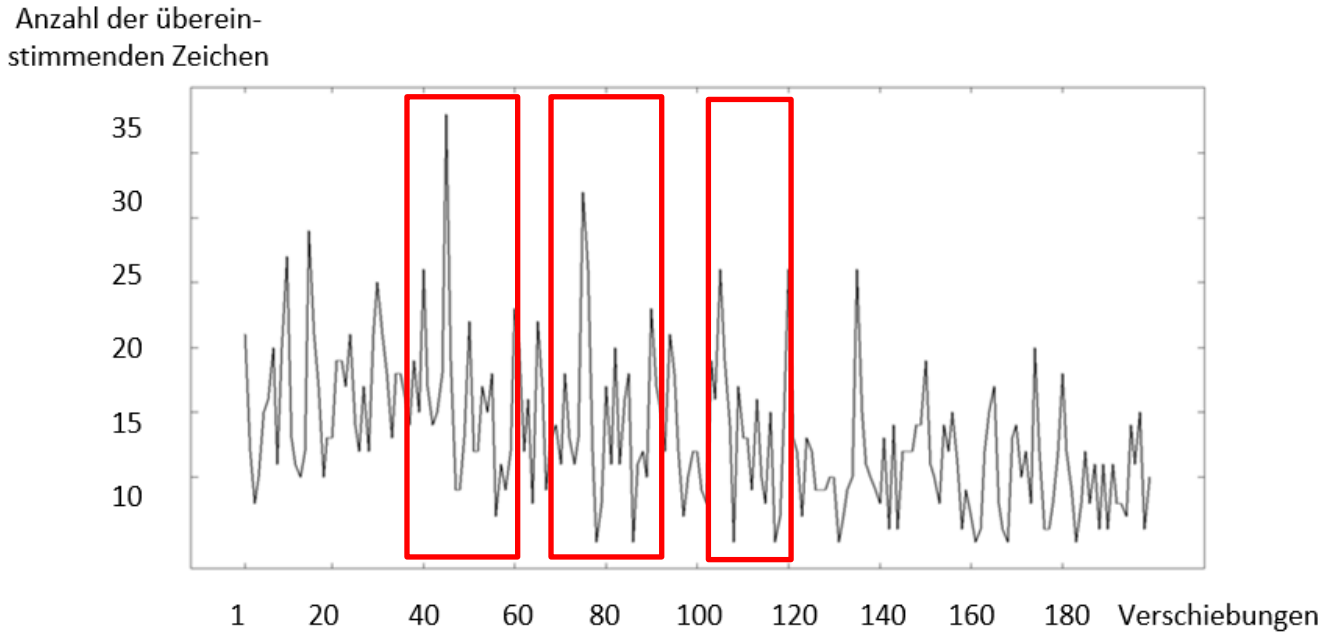


Abbildung 5: Autokorrelation des verschlüsselten Textes, Bsp. 2 – Giraffen

Durch die Analyse mithilfe Cryptool kann der höchste Übereinstimmungswert der Zeichen zu einem bestimmten Shift (Verschiebung) errechnet werden. Am meisenten Übereinstuimmungen werden nach einer Verschiebung von 45 festgestellt, an dieser Stelle werden 38 Übereinstimmungen gemessen. Ein weiterer hoher Übereinstimmungswert, 32, wird nach 75 Verschiebungen erkannt. Die restlichen Spitzen zeigen einen Übereinstimmungswert von 23 – 26 an, bspw. bei einer Verschiebung von 105.

5. Autoregressive to anything

Dieses Kapitel befasst sich mit dem ARTA-Prozess, welcher die Grundlage des Projektes bildet/vorgibt. Anhand mathematischer und graphischer Elemente sollen die mitwirkenden Komponenten veranschaulicht werden.
Folgende Grafik bildet die einzelnen Bestandteile des ARTA-Prozesses ab. In den folgenden Kapiteln wird auf die grundlegenden Elemente eingegangen.

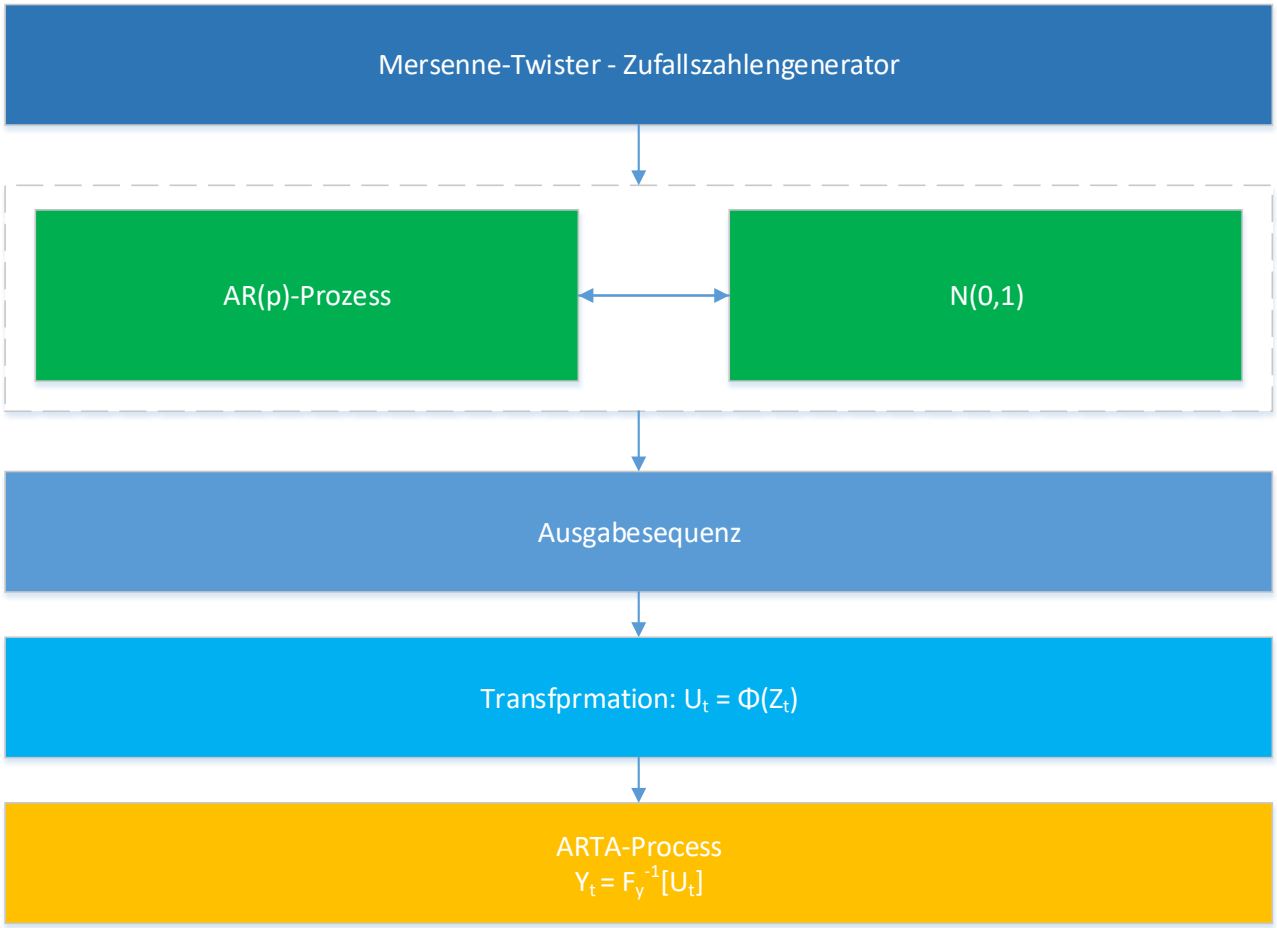


Abbildung 6: Grafische Darstellung der Bestandteile eines ARTA-Prozesses

5.1 Zufallszahlen – Mersenne-Twister

Der ARTA-Prozess benötigt eine Inputsequenz. Diese entstammt aus einem Zufallszahlengenerator. Die Generierung der Zufallszahlen basiert auf dem Algorithmus des Mersenne-Twister[9], entwickelt von Makoto Matsumoto und Takuji Nishimura, 1997. Der Algorithmus existiert in zwei Varianten, wir verwenden MT19937. Die andere Variante wird TT8800 genannt, arbeitet grundsätzlich nach dem gleichen Prinzip, kann jedoch nur eine kleinere Datenmenge verarbeiten.

[9] *Matsumoto, M.; Nishimura, T. (1998). "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator" 3–30. doi:10.1145/272991.272995*

Mersenne-Twister weist drei Eigenschaften auf, welche ihn für die vorliegende Implementation qualifizieren.

1. Er weist eine extrem lange Periode auf. Dies ist ein Kriterium, welches die Güte des Generators beschreibt. Die Periodenlänge des Mersenne-Twister beträgt $p = 2^{19937} – 1$ (Mersenne-Primzahl).
2. Alle Werte bzw. Bits der Ausgabesequenz sind hochgradig gleichverteilt. Im Fall des Mersenne-Twister erfolgt diese Verteilung bis zur 623 Dimension[10]. Daraus resultiert eine extrem geringe Korrelation zwischen den aufeinanderfolgenden Zufallszahlen.
3. Der Algorithmus ist schnell. Eine Ausnahme bilden hier Rechenarchitekturen bzw. -Systeme, welche nur über einen sehr begrenzten Arbeitsspeicher verfügen.

Arta.Standard implementiert den Mersenne-Twister innerhalb der Klasse *MersenneTwister*. Im folgenden Abschnitt wird anhand des Codes die Funktionsweise des zugrundeliegenden Algorithmus erklärt.
Die Grundlage bildet eine Zahlensequenz. Die Startwerte liegen bei Y_1 bis Y_N, wobei N = 624.Die ersten 624 Werte sind im Idealfall echte Zufallszahlen, jedoch funktioniert der Algorithmus auch mit Pseudozufallszahlen. Arta.Standard erzeugt diese Zufallszahlen innerhalb der Klasse RandomSource, wobei es sich in diesem Fall lediglich um Pseudozufallszahlen handelt. Die weiteren Werte mit N > 624 werden folgendermassen berechnet:

$$\mathbf{h = Y_{i-N} - Y_{i-N} \bmod 2^{31}\ Y_{i-N+1} \bmod 2^{31}}$$
$$\mathbf{Y_i = Y_{i-227} \text{ XOR } h/2 \text{ XOR } ((h \bmod 2) * 0x9908B0DF)}$$

Abschliessend wird ein Tempering durchgeführt, dadurch wird die Gleichverteilung der Zufallszahlen sichergestellt.

Mersenne-Twister Algorithmus (Tempering)	Implementation

$X = Y_i \text{ XOR } Y_i / 2^{11}$
$Y = x \text{ XOR } ((x * 2^7) \& 0x9D2C5680)$
$Z = y \text{ XOR } ((y * 2^{15}) \& 0xEFC60000)$
$Z_i = z \text{ XOR } z / 2^{18}$
	`x ^= y >> 11;`
	`y = y ^ (y << 7 & - 0x9D2C5680;`
	`z ^= y << 15 & - 0xEFC60000;`
	`z ^= z >> 18;`
	`return z;`

[10] N-Dimensional: Wird die Ausgabesequenz in Tupel von je n Zahlen zerlegt, so sind diese gleichverteilt im n-dimensionalen Raum.

5.2 Zeitreihen / AR-Prozesse

Eine Zeitreihe[11] beschreibt eine Sequenz von Werten, welche sich an eine bestimmte Struktur halten. Diese Struktur wird durch einen Zeitkoeffizienten definiert. Die einzelnen Werte sind an den entsprechenden Zeitpunkt gebunden. Folgendes Beispiel soll die Grundidee einer Zeitreihe verdeutlichen.

$$Y_t = \frac{1}{2^t}$$

t - Werte	0	1	2	3	4	5
$Y_t = \frac{1}{2^t}$	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$

Tabelle 2 Beispiel Zeitreihe

Die Werte der Zeitreihe Y weisen zum Zeitpunkt t immer die Hälfte des vorhergegangenen Wertes des Zeitpunktes t – 1 auf.
Bei einem AR-Prozess muss der Wert zum Zeitpunkt t nicht nur vom Wert des Zeitpunktes t – 1 abhängen, sondern es ist denkbar, dass er auch vom Wert des Zeitpunktes t – 2 abhängt. Solche autoregressiven Prozesse werden in folgendermassen beschrieben:

AR(p)

Der Parameter p gibt dabei die höchste zeitliche Verzögerung (Lag) an. Beim obigen Beispiel ist dieser Lag = 1. Daher kann die Zeitreihe als AR(1) beschreiben werden.

Ein ARTA-Prozess modelliert eine stationäre Zeitserie. Die Basis bildet dabei ein stationärer, autoregressiver Gaussprozess (AR). Die Werte einer autoregressiven Zeitreihe hängen nicht systematisch vom vorhergegangen Werte ab, sondern können auch von Werten zu einem früheren Zeitpunkt abhängen. Der ARTA-zugrundeliegende AR-Prozess ist folgendermassen definiert:

$$\mathbf{AR(p) = Z_t = \alpha_1 Z_{t-1} + \alpha_2 Z_{t-2} + \alpha_p Z_{t-p} + \varepsilon_t, t = 1, 2, \ldots, n}$$

Z_t definiert den stationären AR(1)-Prozess, $\boldsymbol{\varepsilon_t}$ steht für zufällige, unabhängige Zufallsvariable der Normalverteilung N(0, 1). Die Randverteilung des AR-Prozess Z_t soll einer Normalverteilung N(0,1) genügen, dazu wird die Varianz σ^2 folgendermassen angepasst:

$$\sigma^2 = 1 - \alpha_1 r_1 - \alpha_2 r_2 - \alpha_p r_p$$

Der Autokorrelationskoeffizient r_h zum Lag *h* wird folgendermassen beschrieben:

$$r_h = Corr[Z_t, Z_{t+h}]$$

Anschliessend wird $\{Z_t, N(0,1)\}$ durch die Transformation $U_t = \Phi(Z_t)$ in eine stetige Verteilung $U(0,1)$ überführt. Φ stellt dabei die Standardnormalverteilung dar. Durch Inversionsverfahren[12] kann nun U_t in den ARTA-Prozess Y_t umgewandelt werden.

$$Y_t = F_Y^{-1}[U_t]$$

[11] Quelle: Zeitreihenanalyse- Einstieg und Aufgaben von Thomas Mazzoni, FernUniversität in Hagen
[12] Quelle: Der Einfluss von Autokorrelation in komplexen Materialflusssystemen, Rank, Schmidt, Uhlig

5.3 ARTA und Autokorrelation

Dem AR-Prozess liegt eine natürlich autokorrelierte Struktur zugrunde. Diese ist durch den Lag (Zeitverzögerung), welche durch den Parameter p ausgedrückt wird, gegeben. Die Herausforderung liegt nun darin, diese Autokorrelation auf den darüber liegenden ARTA-Prozess zu transformieren. Diese Aufgabe wird durch ein Yule-Walker-Gleichungssystem[13] gelöst, welches ein numerisches Suchverfahren zur Bestimmung der Regressionskoeffizienten darstellt.

5.4 Verteilungen

Die von einem ARTA-Prozess erzeugten Zufallszahlen unterliegen einer definierten Verteilung. Jeder Wahrscheinlichkeitsverteilung kann eine Verteilungsfunktion [14]zugeordnet werden. Dabei entspricht der Wert der Verteilungsfunktion an der Stelle x der Wahrscheinlichkeit, dass die zugehörige Zufallsvariable X einen Wert annimmt, der kleiner oder gleich x ist. Die Verteilungsfunktion kann durch zwei Definitionen ausgedrückt werden, einerseits durch das Wahrscheinlichkeitsmass oder mittels einer Zufallsvariable.

Definition mittels Wahrscheinlichkeitsmass: Auf dem Ereignisraum der reellen Zahlen sei das Wahrscheinlichkeitsmass P gegeben. Dies kann durch die Funktion

$$\boldsymbol{F_p}: \mathbb{R} \rightarrow [\mathbf{0}, \mathbf{1}]$$

Ausgedrückt werden. Die Verteilungsfunktion von P lautet:

$$\boldsymbol{F_p(x) = P((-\infty, x])}$$

Die Funktion gibt an der Stelle x an, mit welcher Wahrscheinlichkeit ein Ergebnis aus der Menge (-∞, x] eintritt.

Definition mittels Zufallsvariable: Ist X eine reelle Zufallsvariable, so definiert sich die Verteilungsfunktion von X folgendermassen:

$$\boldsymbol{F_p(x) = P(X \leq x)}$$

Durch P($X \leq x$) wird die Wahrscheinlichkeit ausgedrückt, dass X einen Wert kleiner oder gleich x annehmen wird.

In den folgenden Unterkapiteln, wollen wir die gängigsten Verteilungen im Zusammenhang mit ARTA kurz erklären und deren Definition aufzeigen.

5.4.1 Normalverteilung

Die Normalverteilung[15] auch Gaussverteilung genannt, stellt ein wichtiger Typ stetiger Wahrscheinlichkeitsverteilungen dar. Ihre grosse Bedeutung beruht unter anderem auf dem zentralen Grenzwertsatz[16].

Name	Normalverteilung

[13] https://de.wikipedia.org/wiki/Yule-Walker-Gleichungen, Peter J. Brockwell, Richard A. Davis: *Time Series. Theory and Methods, Springer*. 2. verb. Aufl. Springer, New York 2006, ISBN 978-0-387-97429-3.
[14] Quellen: https://de.wikipedia.org/wiki/Verteilungsfunktion
Wahrscheinlichkeitsrechnung und Statistik, A. Müller, HSR
[15] Bild und «Steckbrief» entnommen aus dem Skript zu Wahrscheinlichkeit und Statistik von A. Müller, HSR
[16] Zentraler Grenzwertsatz: https://de.wikipedia.org/wiki/Zentraler_Grenzwertsatz

Dichtefunktion	$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Verteilungsfunktion	Leine elementare Funktion
Erwartungswert	μ
Varianz	σ^2
Anwendung	• Messwerte • Summe vieler kleiner Einflüsse • Approximation der Binomialverteilung

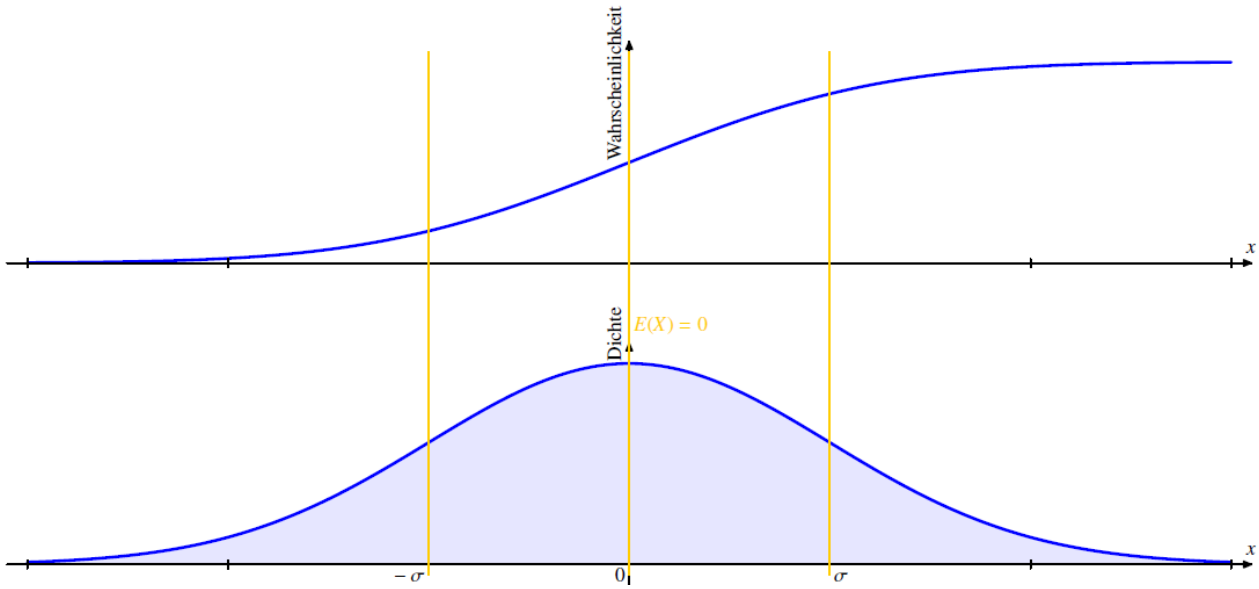


Figure 1 : Verteilungsfunktion (oben) und Dichtefunktion (unten) der Normalverteilung

5.4.2 Exponentialverteilung

Bei der Exponentialverteilung[17] handelt es sich um eine stetige Wahrscheinlichkeitsverteilung über eine Menge positiver reeller Zahlen, welche durch eine Exponentialfunktion gegeben ist. Ihr Einsatzgebiet liegt in der Beantwortung der Frage der Dauer von zufälligen Zeitintervallen.

Name	Exponentialverteilung
Dichtefunktion	$ae^{-ax}, a > 0$
Verteilungsfunktion	$1 - e^{-ax}$
Erwartungswert	$\frac{1}{a}$
Varianz	$\frac{1}{a^2}$
Anwendung	• Prozesse ohne Erinnerungsvermögen • Radioaktivität

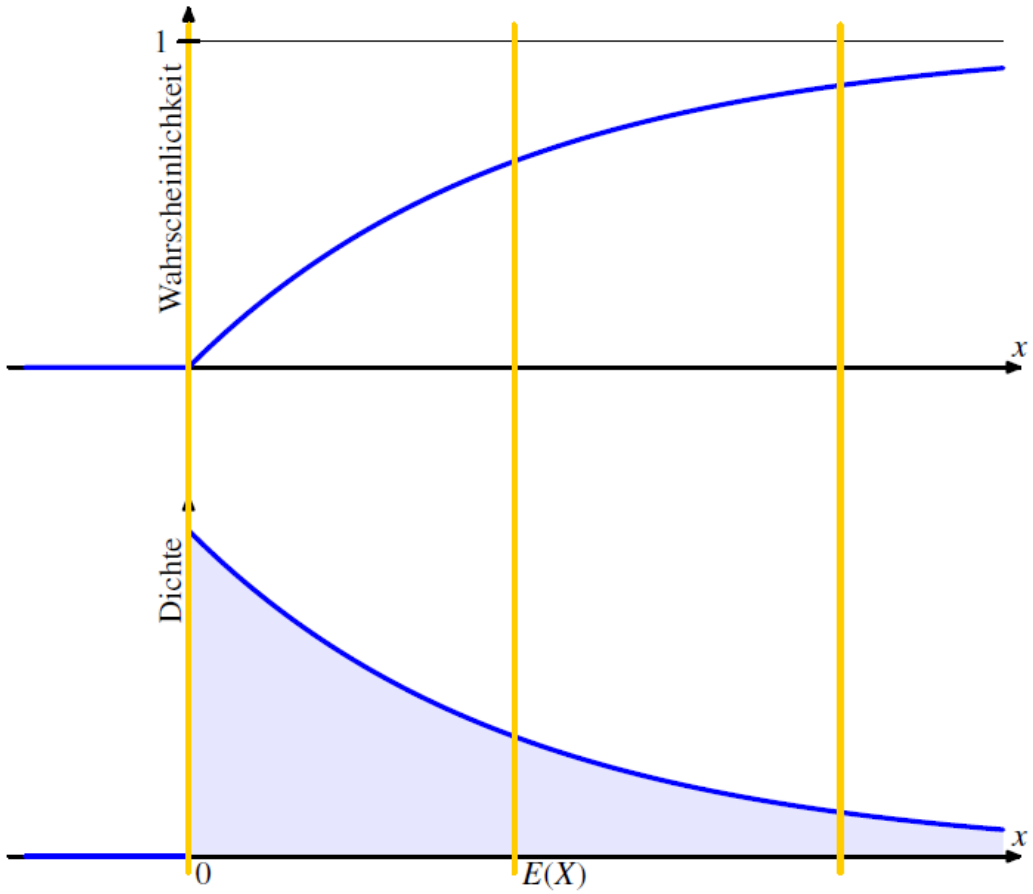


Figure 2: Verteilungsfunktion (oben) und Dichtefunktion (oben) der Exponentialverteilung

[17] Bild und «Steckbrief» entnommen aus dem Skript zu Wahrscheinlichkeit und Statistik von A. Müller, HSR

5.4.3 Stetige Gleichverteilung

Eine stetige Gleichverteilung [18]beschreibt eine stetige Wahrscheinlichkeitsverteilung. Dies bedeutet, dass Werte auf einem Intervall eine konstante Wahrscheinlichkeitsdichte aufweisen. Demnach ist gegeben, dass alle gleichlangen Teilintervalle ebenfalls dieselbe Wahrscheinlichkeit besitzen.

Name	Gleichverteilung
Dichtefunktion	$\begin{cases} \frac{1}{b-a} \\ 0 \end{cases}$
Verteilungsfunktion	$\begin{cases} 0 \\ \frac{x-a}{b-a} \\ 1 \end{cases}$
Erwartungswert	$\frac{a+b}{2}$
Varianz	$\frac{(b+a)^2}{12}$
Anwendung	• Verteilung von zufallszahlen • Keine bevorzugten Werte

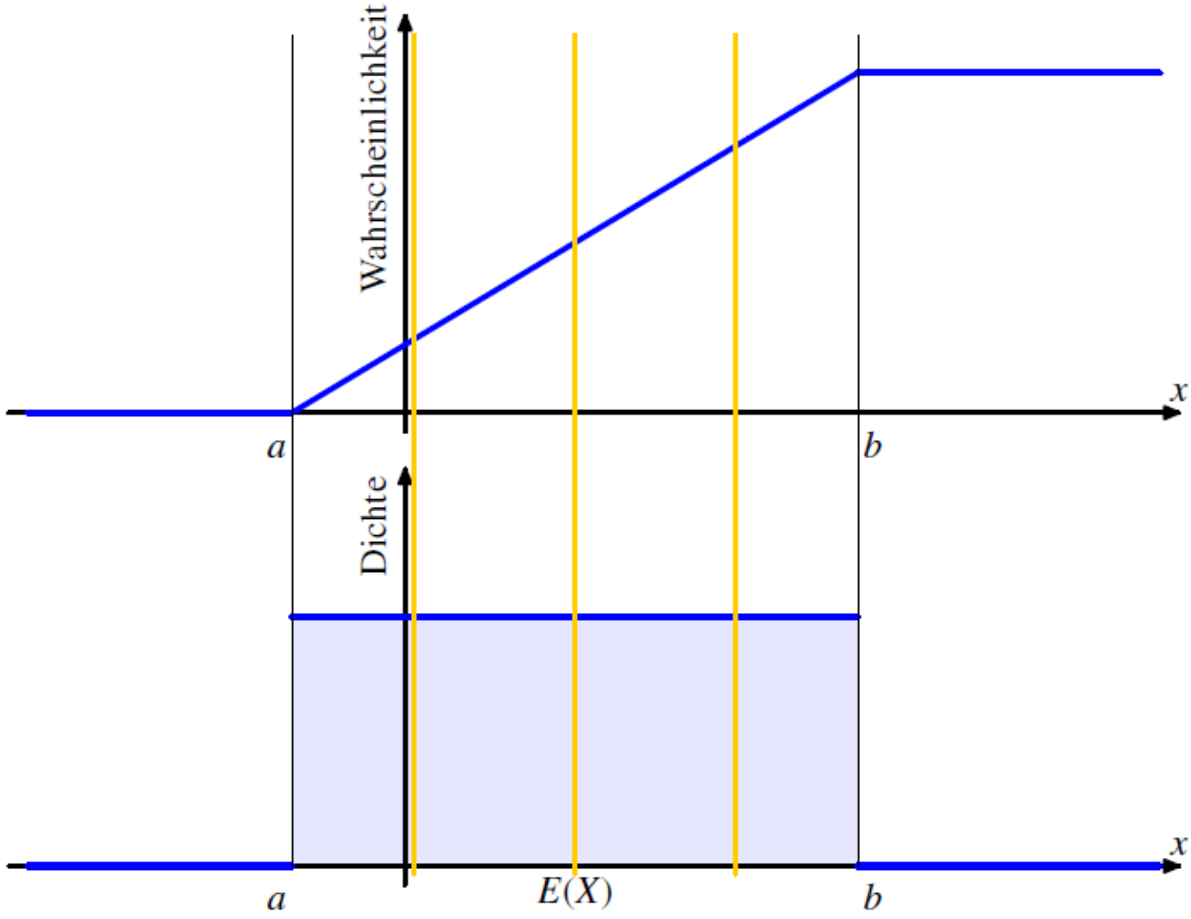


Figure 3: Verteilungsfunktion (oben) und Dichtefunktion (unten) der Gleichverteilung

[18] Bild und «Steckbrief» entnommen aus dem Skript zu Wahrscheinlichkeit und Statistik von A. Müller, HSR

5.4.4 PearsonsCorrelation

Die Klasse *AutoCorrelation* übernimmt die Funktion zur Errechnung der Korrelationskoeffizienten sowie der Erzeugung der Korrelationsmatrizen. Durch die Formel von Pearson kann leicht der Korrelationskoeffizient r zwischen zwei Variablen ermittelt werden. Folgendes numerische Beispiel soll dies verdeutlichen.

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\sqrt{\left(\sum X^2 - \frac{(\sum X^2)}{n}\right)\left(\sum Y^2 - \frac{(\sum Y^2)}{n}\right)}}$$

X – Werte	1	3	4	4
Y – Werte	2	5	5	8

Mithilfe der obenstehenden Formel können folgende Werte errechnet werden:

Ausdruck	Bedeutung/numerischer Wert
$\sum XY$	(1)(2) + (3)(5) + (4)(5) + (4)(8) = 69
$\sum X$	1 + 3 + 4 + 4 = 12
$\sum Y$	2 + 5 + 5 + 8 = 20
$\sum X^2$	$1^2 + 3^2 + 4^2 + 4^2 = 42$
$\sum Y^2$	$2^2 + 5^2 + 5^2 + 8^2 = 118$
n	Anzahl der Werte/ Länge der Zahlenreihe

Werden die errechneten Werte nun in die Gleichung eingesetzt, kann der Pearson-Korrelationskoeffizient ermittelt werden:

$$r = \frac{69 - \frac{12*20}{4}}{\sqrt{\left(42 - \frac{(12)^2}{4}\right)\left(118 - \frac{(20)^2}{4}\right)}} = 0.866$$

Das folgende Codefragment zeigt die Berechnung der Korrelationskoeffizienten, so wie sie in dieser Arbeit umgesetzt ist. Weiter übernehmen diese Klassen die Erzeugung der partiellen Korrelationskoeffizienten und der Korrelationsmatrizen.

5.4.5 Grenzen von ARTA [Philipp]

Ein weiterer Aspekt soll die Grenzen von ARTA aufzeigen. Damit ist gemeint, dass auf die angegebenen Schwächen, welche im Dokument «JARTA — A Java library to model and fit Autoregressive-To-Anything processes[19]» genannt sind. *[TODO] Weiterführen, Grenzen aufzeigen*

[19] JARTA — A Java library to model and fit Autoregressive-To-Anything processes, 10.1109/WSC.2013.6721508

6. Arta.Standard

[Überarbeiten]

Arta.Standard ist die Klassenbibliothek, welche als Grundlage der Modellierung stochastischer Prozesse darstellt und zur Integration in die Simulationssoftware Simio bereitstellt. Auf den folgenden Seiten sind die einzelnen, relevanten Klassen und Algorithmen dargelegt, welche essentiell zur Realisierung beitragen. Arta.Standard greift auf die Sammlung mathematischer Funktionen und Klassen der MathNet.Numerics[20]-Library zurück. Diese stellt eine Vielzahl an ausgewählten Klassen und Funktionen bereit, welche zur Modellierung des ARTA-Prozesses essentiell sind.

Arta.Standard basiert auf einer .Net Standard 1.6 Klassenbibliothek. Damit setzen wir auf dem untersten Layer auf, was Abhängigkeiten reduziert und die Bibliothek unabhängiger macht.

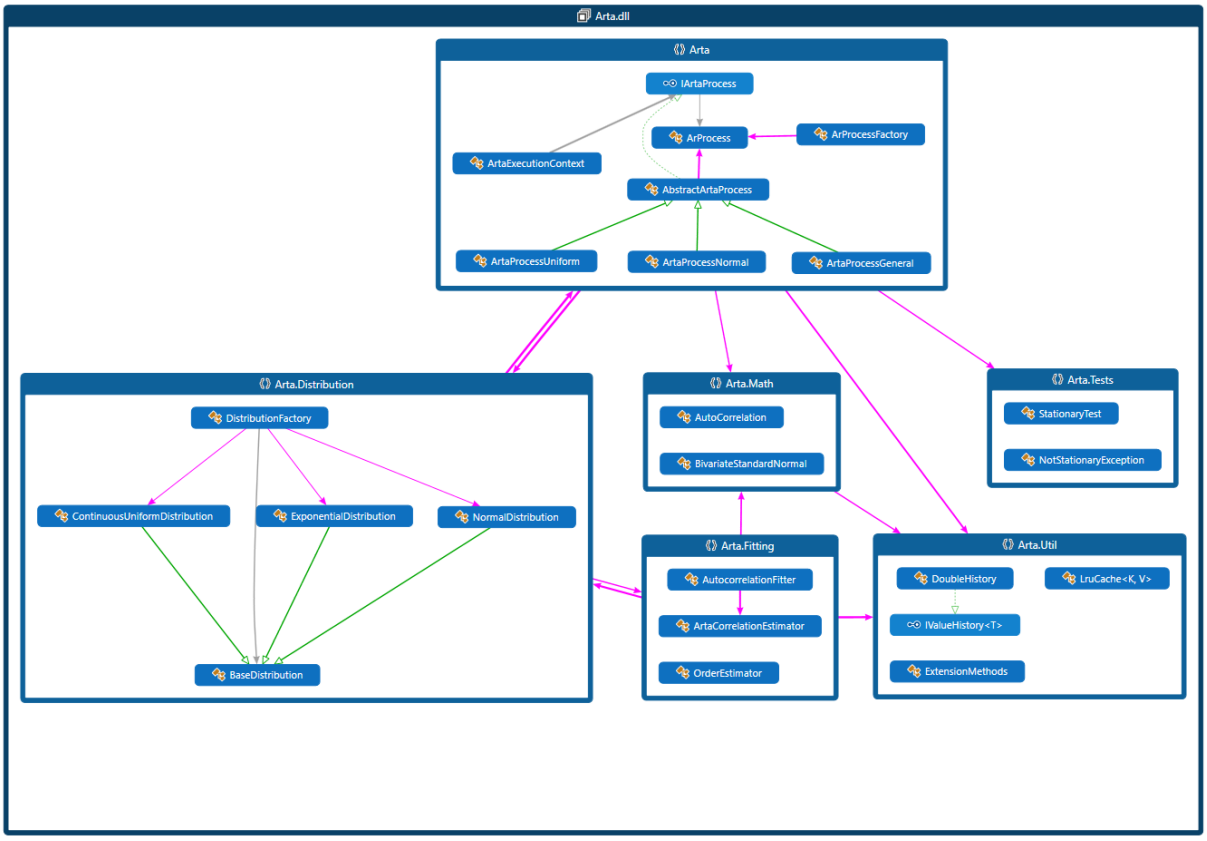


Abbildung 7: Klassendiagramm Arta.Standard

[Einführung grober Überblick mittels DomainModel]

[20] https://numerics.mathdotnet.com/
https://github.com/mathnet/mathnet-numerics

6.1 Arta

Dieser Namespace bildet die Fassade der Klassenbibliothek. Direkt darin sind die Klassen zur Erzeugung des AR- und ARTA-Prozesses angesiedelt.

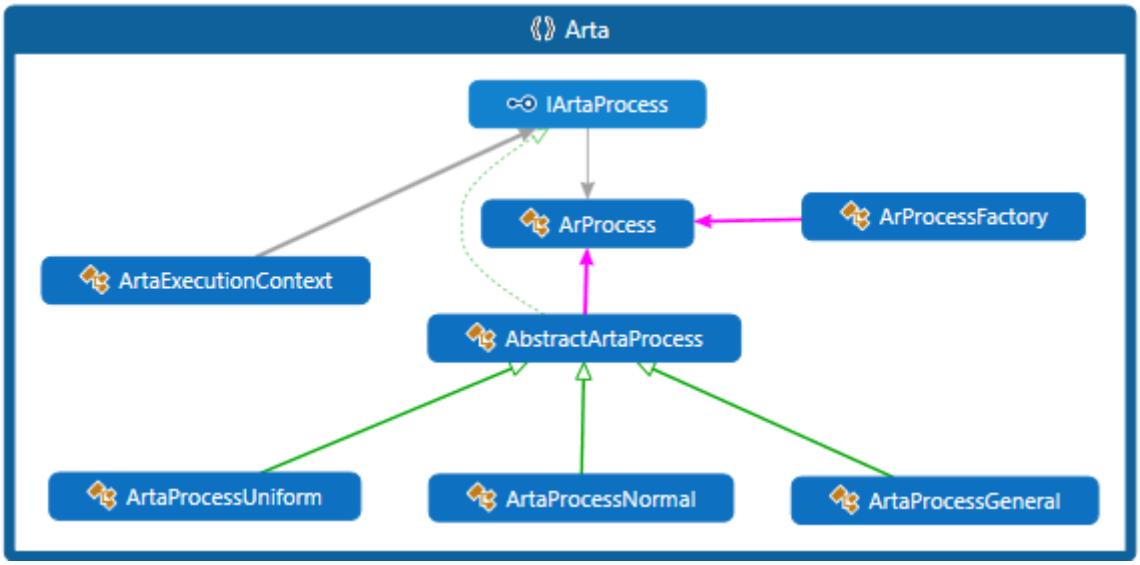


Abbildung 8: Überblick Namespace Arta

Erzeugt und gekapselt wird ein ARTA-Prozess durch einen *ArtaExecutionProcess*. Diesem werden die Korrelationskoeffizienten und die gewünschte Verteilung, per Enum-Typ, übergeben. Durch das Setzen des entsprechenden Verteilungstyp erzeugt die *DistributionFactory* einen entsprechenden ARTA-Prozess und gibt diesen zurück. Zuletzt wird durch die *ArProcessFactory* einen neuen AR-Prozess instanziiert. Anschliessend werden die entsprechenden Objekte zurückgegeben und im *ArtaExecutionContext* gekapselt.

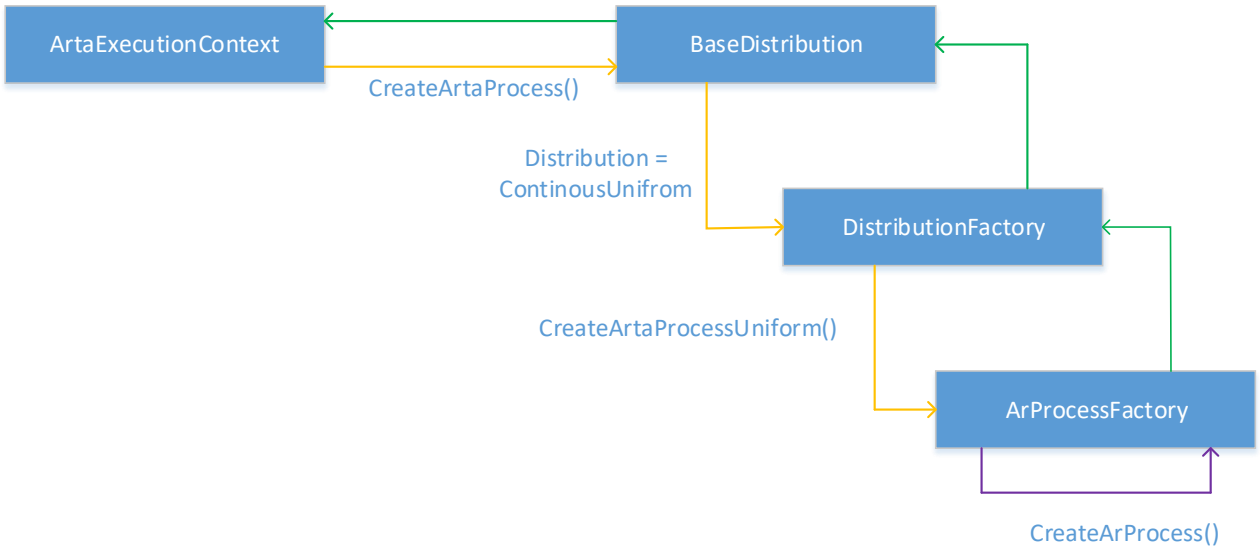


Abbildung 9: Abstrakte Erzeugung eines ARTA-Prozesses

Die abstrakte Klasse *AbstractArtaProcess* gilt als Basisklasse für alle spezifischen ARTA-Prozesse, welche in den Klassen *ArtaProcessNormal*, *ArtaProcessUniform* und *ArtaProcessGeneral* umgesetzt sind. Da sich die stetige Gleichverteilung und die Normalverteilung grundlegend von den Verteilungsarten unterscheiden, werden sie auch in einer separaten Klasse abgebildet. Für alle anderen Verteilungen soll die Klasse *ArtaProcessGeneral* zum Zuge kommen. Die einzelnen Klassen überschreiben die geerbten Methoden und implementieren zum Teil verteilungsspezifische Operationen.

```
class ArtaProcessNormal : AbstractArtaProcess
{
    public readonly double standardDeviation, mean;

    public ArtaProcessNormal(ArProcess ar, double mean, double variance) : base(ar)
    {
        this.mean = mean;
        standardDeviation = System.Math.Sqrt(variance);
    }

    protected override double Transform(double value)
    {
        return value * stdev + mean;
    }
}
```

Untenstehendes Sequenzdiagramm soll einen vertieften Einblick in die Erzeugung eines ARTA-Prozesses geben.

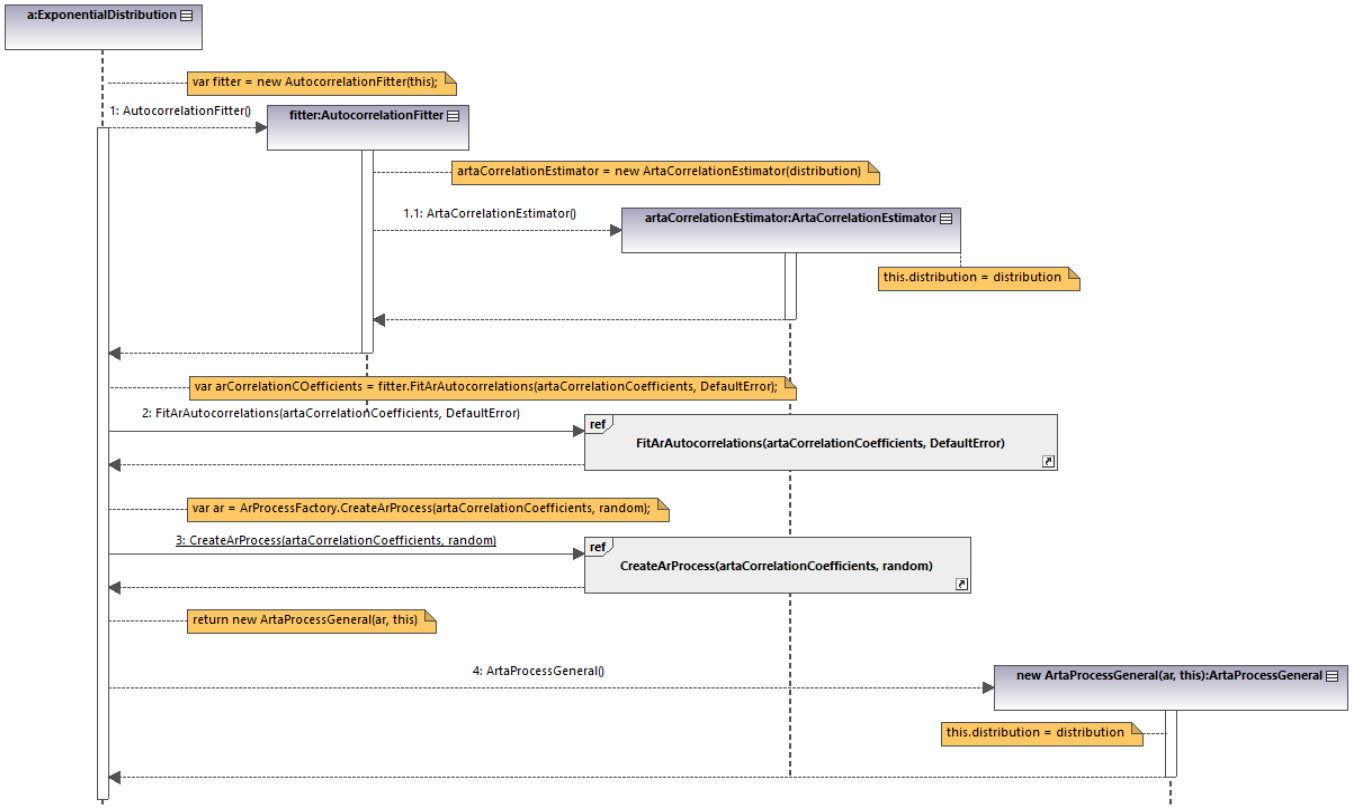


Abbildung 10: Sequenzdiagramm CreateArtaProcess()

6.2 Arta.Distribution

Dieser Namespace beherbergt alle Verteilungsklassen. Einerseits sind werden hier Klassen für die eigentliche Verteilung abgebildet, andererseits erzeugt die *DistributionFactory* je nach gewählter Verteilung einen entsprechenden ARTA-Prozess.

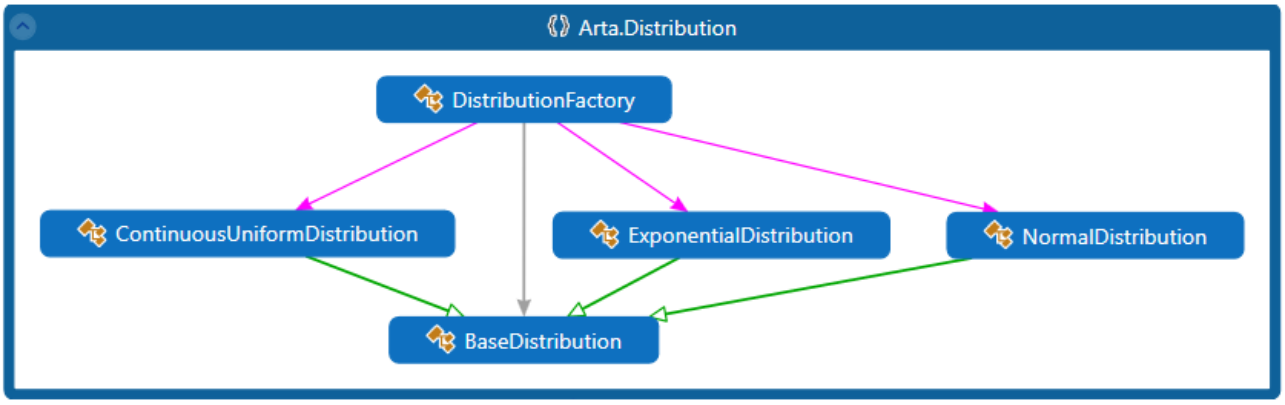


Abbildung 11: Überblick Namespace Arta.Distribution

Die Klasse *BaseDistribution* stellt die abstrakte Klasse für alle Verteilungen dar. Die konkreten Implementierungen der einzelnen Verteilungen finden in den gleichnamigen Klassen statt. Sie kapseln jeweils Objekte einer Verteilung und geben eine konkrete Verteilung aus Mathnet.Numerics zurück. Durch diese Kapselung und Generalisierung durch das Basisinterface können wir in der Implementation der gesamten Klassenbibliothek einen höheren Abstraktionslevel aufrechterhalten. Weiter ist die Erweiterung um weitere Verteilungen einfach realisierbar.

```
public override AbstractArtaProcess CreateArtaProcess(double[] artaCorrelationCoefficients, RandomSource random)
{
    var dim = artaCorrelationCoefficients.Length;
    var arCorrelationCoefficients = new double[dim];
    for (var i = 0; i < dim; i++)
        {
         arCorrelationCoefficients[i] = 2 * System.Math.Sin(System.Math.PI * artaCorrelationCoefficients[i] / 6);
        }
    ArProcess ar = ArProcessFactory.CreateArProcess(arCorrelationCoefficients, random);
    return new ArtaProcessUniform(ar, continuousUniform.LowerBound, continuousUniform.UpperBound);
}
```

6.3 Arta.Math und Arta.Fitting

Innerhalb dieser Namespaces werden die mathematischen Operationen zur Erzeugung des AR- und ARTA-Prozesses durchgeführt. Durch die Klasse *AutoCorrealation* wird eine Korrelationsmatrix auf der Basis der übergebenen Korrelationskoeffizienten generiert. Diese wird wiederum durch einen *AutocorrelationFitter* genutzt, um die Struktur des darunterliegenden AR-Prozesses, mittels der Methode Transform(), auf die des ARTA-Prozesses anzupassen. Weiter können die Autokorrelationsfunktionen ACFS und PACFS durch die Klasse *AutoCorrelation* ermittelt werden.

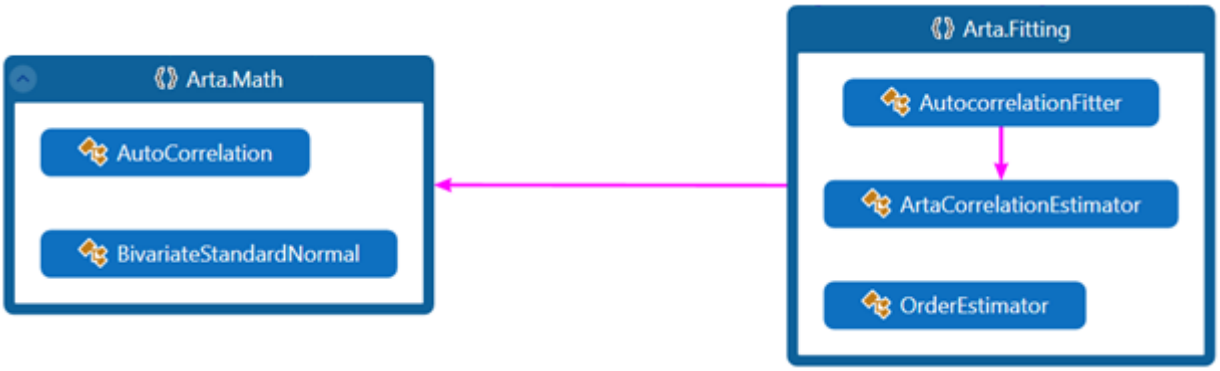


Abbildung 12: Namespace Arta.Math und Arta.Fitting

```
private double Transform(double value)
{
        var result = transformationCache.Get(value);
        if (result == 0)
        {
                result = standardNormal.CumulativeDistribution(value);
                result = distribution.InverseCumulativeDistribution(result);
                transformationCache.Add(value, result);
        }
        return result;
}
```

```
public static double CalculateAcf(double[] data, int lag)
{
      var acc = 0.0;
      var length = data.Length;
      if (lag < 0)
            throw new ArgumentException("Negative Lags are not allowed");
      if (lag > length)
            throw new ArgumentException("Lag exceeds sample size");
      if (lag == 0)
            acc = 1.0;
      else
            acc = Correlation.Pearson(data.CopyOfRange(0, length - lag), data.CopyOfRange(lag, length));
      return acc;
}

public static double[] CalculateAcfs(double[] data, int maxLag)
{
      var accs = new double[maxLag + 1];
      for (var lag = 0; lag <= maxLag; lag++){
            accs[lag] = CalculateAcf(data, lag);
      }
      return accs;
}
```

6.4 Statistische Tests [Philipp]

Tests sind in einem separaten Assembly *StatisticalTests* realisiert. Dabei handelt es sich lediglich um Tests der Klassenbibliothek an sich. Die Integration in Simio wird separat in Form eines Integrationstestes und verschiedener Szenarien getestet.

Als Werkzeug zum Testen und Auswerten der Klassenbibliothek und deren generierten Zahlen wurde *ArtaStatistics* implementiert. Dabei handelt es sich um eine Klasse, welche einen *ArtaExecutionContext* entgegennimmt und anschliessend Abfragen auf einzelne Elemente des ARTA-Prozesses ermöglicht. Ziel ist es, möglichst viele Einblicke in den ARTA-Prozess über ein *ArtaStatistics* -Objekt zu erhalten. Im Rahmen dieser Arbeit haben wir uns auf folgende Elemente konzentriert und diese in entsprechende Methoden abgefertigt.

Methode	Beschreibung
Initialize(int lag)	Initialisiert den ARTA-Prozess mit dem gegebenen Lag
Iterations(int iterations)	Setzt die Anzahl zu generierender ARTA-Zahlen
Acfs()	Berechnet die Autokorrelationsfunktionen (ACFS)
Pacfs()	Berechnet die partiellen Autokorrelations-funktionen(PACFS)
ArtaNumbers()	Erzeugt ARTA-Zahlen und speichert diese in ein Array
Order()	Errechnet die Order
Execute()	Führt die gewählten Methoden aus und stellt sie auf der Konsole dar
CorrelationMatrix()	Liefert die Korrelationsmatrix des Arta-Prozesses
ArProcess()	Erzeugt die Werte des AR-Prozesses, welche für den ARTA-Prozess transformiert werden
Reset()	Setzt die bereits vorgenommenen Einstellungen zurück

```
var executionContext = new ArtaExecutionContext(new ExponentialDistribution(), new double[] { -0.4, 0.5 });
var artaProcess = executionContext.CreateArtaProcess();

ArtaStatistics arta = new
ArtaStatistics(executionContext).Initialize(10).Iterations(1000).ArtaNumbers().Acfs().Pacfs().Excecute();
```

Codefragment 1: Nutzung ArtaStatistics

Mithilfe dieses *ArtaStatistics*-Objekts haben wir verschiedene ARTA-Prozesse, mit verschiedenen Parameter, erzeugt und anschliessend ausgewertet.

7. Integration Simio [bis 13.12.2017]

Durch das erzeugte User-defined Element wird es möglich, die Klassenbibliothek Arta.Standard innerhalb der Simulationssoftware Simio zu nutzen. Folgendes Kapitel zeigt den internen Aufbau und die Anwendung des ArtaElements auf.

7.1 Aufbau

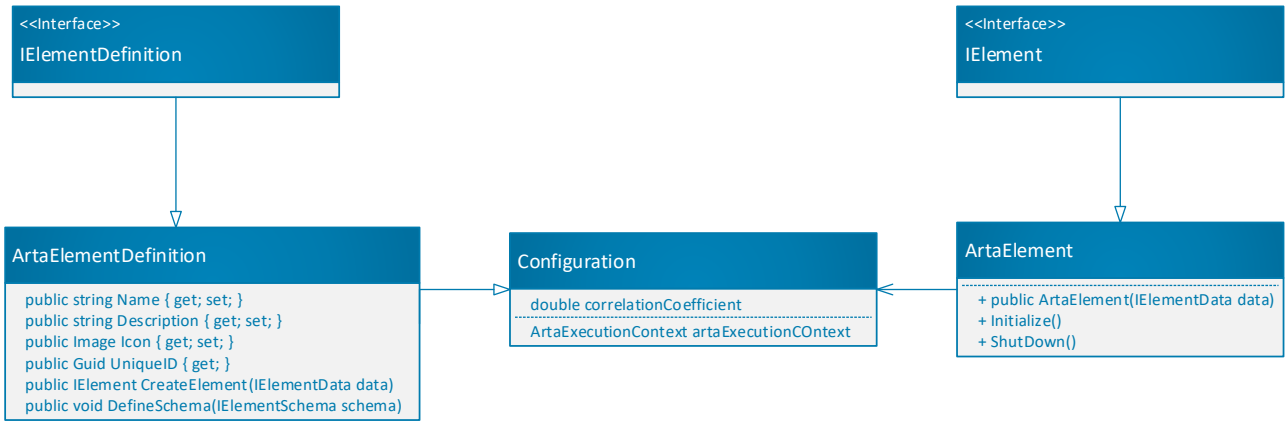


Abbildung 13: Aufbau ArtaElement

Innerhalb des *ArtaElement* wird durch einen *ArtaExecutionContext* ein neuer ARTA-Prozess erzeugt. Die Korrelationskoeffizienten sowie die Verteilung können über die graphische Benutzerschnittstelle im Simio wie gewünscht eingestellt werden. Diese Eigenschaften (Verteilung und Koeffizienten) werden durch entsprechende, im Schema (*ArtaElementDefinition*) definierte, Properties abgefangen und dem *ArtaExecutionContext* übergeben.
Die Herausforderung liegt darin, dass das ArtaElement zuverlässig die Zahlen aktualisiert und diese ordnungsgemäss an ein Simio-Objekt weitergibt. Wir haben dazu verschiedene Ansätze getestet, einerseits mit eigenen Properties, UserSteps und Functions.
Die einzige zuverlässige Lösung haben wir in einer Function gefunden. Diese ist in Form eines Delegates durch die Simio API gegeben. Dem Delegate wird eine Funktion übergeben, welche die nächste ARTA-Zahl als Rückgabewert besitzt.

```
public void DefineSchema(IElementSchema schema)
{
        IPropertyDefinition pd;
        pd = schema.PropertyDefinitions.AddRealProperty("CorrelationCoefficient1", 0.4);
        pd = schema.PropertyDefinitions.AddRealProperty("CorrelationCoefficient2", 0.5);
        schema.ElementFunctions.AddSimpleRealNumberFunction("Process",
                  "Returns \"Autoregressiv To Anything\" numbers with the given Correlationcoefficients.",
                  new SimioSimpleRealNumberFunc(ArtaProcess));
}

private static double ArtaProcess(object element)
{
        return artaExecutionContext.ArtaProcess.Next();
}
```

```
public ArtaElement(IElementData data)
{
      _data = data;
}
public void Initialize()
{
      var corr1 = _data.Properties.GetProperty("CorrelationCoefficient1");
      var corr2 = _data.Properties.GetProperty("CorrelationCoefficient2");
      correlationCoefficient1 = corr1.GetDoubleValue(_data.ExecutionContext);
      correlationCoefficient2 = corr2.GetDoubleValue(_data.ExecutionContext);
      artaExecutionContext = new ArtaExecutionContext(BaseDistribution.Distribution.ExponentialDistribution,
        new double[] { correlationCoefficient1, correlationCoefficient2 });
}
```

7.2 Anwendung

Das Erzeugte ArtaElement kann verschiedensten Simio-Bausteinen übergeben werden. In dieser Arbeit werden ausschliesslich InterarrivalTimes von verschiedenen Quellen erzeugt. Über den «Definitions»-Tab kann das ArtaElement als User-defined-Element in das Projekt integriert werden.

8. Test und Auswertung [[bis 25.11.2017]

Dieses Kapitel deckt die Verifikation der Klassenbibliothek ab. In einem ersten Schritt wollen wir die Daten, welche unsere Implementation ausgibt mit den der bestehenden (JARTA) vergleichen. Dazu erzeugen wir verschiedene ARTA-Prozesse mit verschiedenen Parametern, jeweils mit unserer Implementation und in JARTA. Anschliessend geben wir die Sequenz von ARTA-Zahlen in ein Excel-File aus und analysieren die Daten mittels Diagrammen.

Folgende Tabelle soll Aufschluss über die Parameter und Einstellungen über alle Vergleiche geben. Für alle Verteilungen werden jeweils 1500 ARTA-Zahlen erzeugt, um die Zahlen jedoch besser zu visualisieren, wird jeweils ein Ausschnitt von 100 analysiert.

Bezeichnung	Parameter
Anzahl analysierter Zahlen	100, 1500 erzeugt
Lag	10
Korrelationskoeffizienten	-0.4, 0.5

Parameterbezeichnung	Wert
Verteilung	ContinousUniform
LowerBound	-1
UpperBound	1

Parameterbezeichnung	Wert
Verteilung	Normal
Mean	0
Varianz	1

Parameterbezeichnung	Wert
Verteilung	Exponential
Mean	1

Wir gehen davon aus, dass sich die erzeugten Werte voneinander unterscheiden, jedoch gleiche Muster entstehen. Die Unterschiede der einzelnen Werte bezüglich ihrer effektiven Werte können wir auf unterschiedliche Implementationen der Zufallszahlengeneratoren zurückführen.

8.1 Vergleich ARTA-Zahlen

In allen Diagrammen können sich wiederholende Muster erkannt werden. Diese zeigen auf, dass eine Autokorrelation vorhanden ist. Ein schnell erkennbarer Unterschied liegt darin, dass sich bei unserer Implementation höhere Wertespitzen gebildet haben.

8.1.1 Continous Uniform

Das wichtigste Kriterium der stetig gleichverteilten ARTA-Zahlen liegt in ihrem Wertebereich. Dieser soll hier -1 und 1 nicht überschreiten. Aus der Grafik kann dies herausgelesen werden, sämtliche Werte befinden sich im Zielbereich. Weiter kann zwischen Arta.Standard und JARTA kein grosser Unterschied festgestellt werden. Beide Graphen zeigen die Struktur einer Autokorrelation.

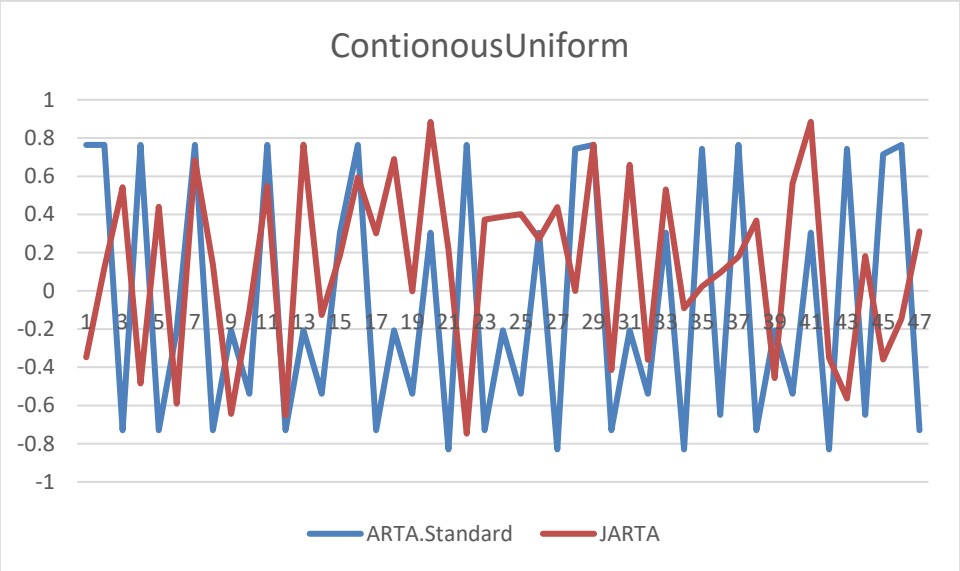


Abbildung 14: Vergleich ARTA-Zahlen mit ContinousUniform (-1, 1)

8.1.2 Normal

Vergleich man die ARTA-Zahlen können ähnliche Verläufe der Graphen ausgemacht werden. Erkenntlich sind die ruhigen Abschnitte, welche anschliessend auf eine Ansammlung von Ausreissern gefolgt werden. Dieses Muster wird ebenfalls in "JARTA – A Java Library to model and fit Autoregressive-to-Anyting processes" erwähnt.

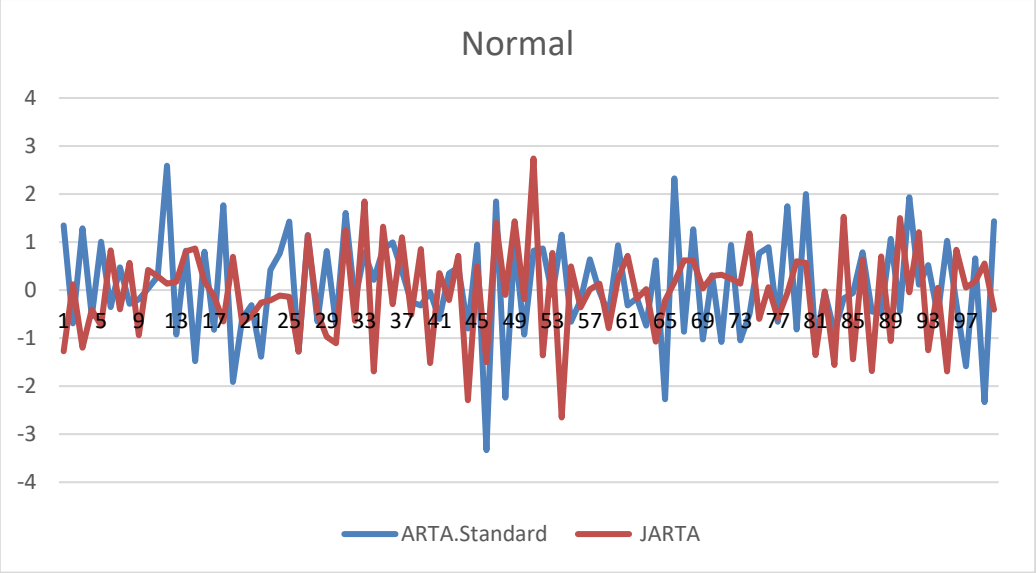


Abbildung 15: Generierte ARTA-Zahlen mit N (0,1)

8.1.3 Exponential

Ein Kriterium für exponentiell verteilte Zahlen wird durch ihren Wertebereich beschrieben. Dieser darf nicht kleiner als Null sein. Im Diagramm ist ersichtlich, dass Werte in der Nähe von Null vorkommen, diese Grenze jedoch nie überschreiten. Auch in dieser Verteilung sind bei unserer Implementation vermehrt hohe Wertespitzen erkennbar.

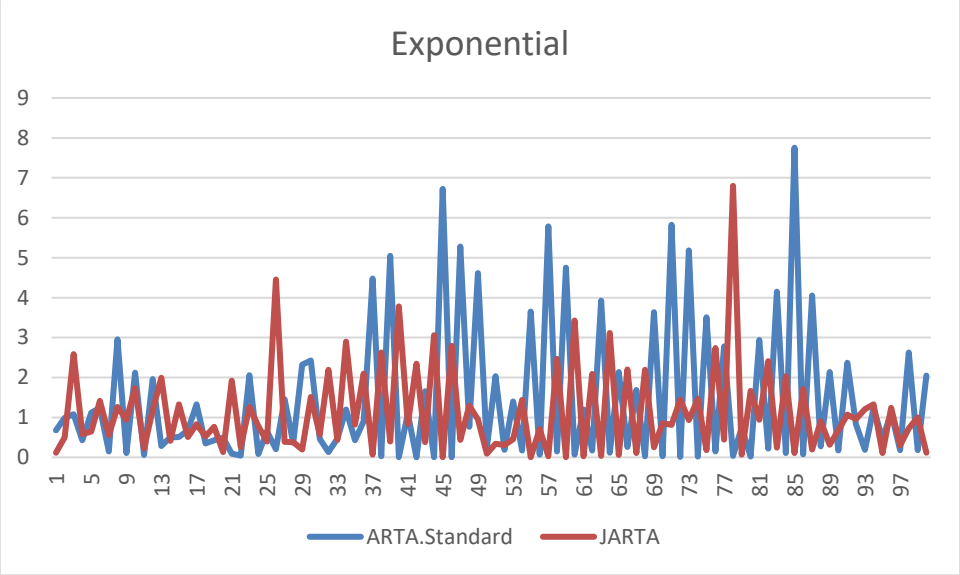


Abbildung 16: Vergleich der von ARTA generierten Zahlen, exponentiell verteilt

8.2 Vergleich ACFS

Folgend vergleichen wir die Autokorrelationsfunktionen der drei gewählten Verteilungen. Für alle Autokorrelationsfunktionen gilt, dass der erste Koeffizient jeweils den Wert 1 besitzen muss. Weiter müssen für den Wert des Lags ebenfalls gleichviele ACFS berechnet werden. Diese beiden Kriterien sind durch Arta.Standard erfüllt. Die ACFS-Werte müssen stetig abnehmen und sich 0 annähern. Durch die Beobachtung der Graphen wird schnell ersichtlich, dass unsere Implementation eine langsamere Annäherung der ACFS an Null durchführt. Dieses Verhalten liegt an den verschiedenen Implementationen von den Zufallszahlengeneratoren. Nach mehrmaligen Durchläufen der beiden Implementationen können wir dies mit Gewissheit sagen. In unseren Werten kann die gleiche Struktur wie in den JARTA-Werten wiedererkannt werden.

8.2.1 ContinousUniform

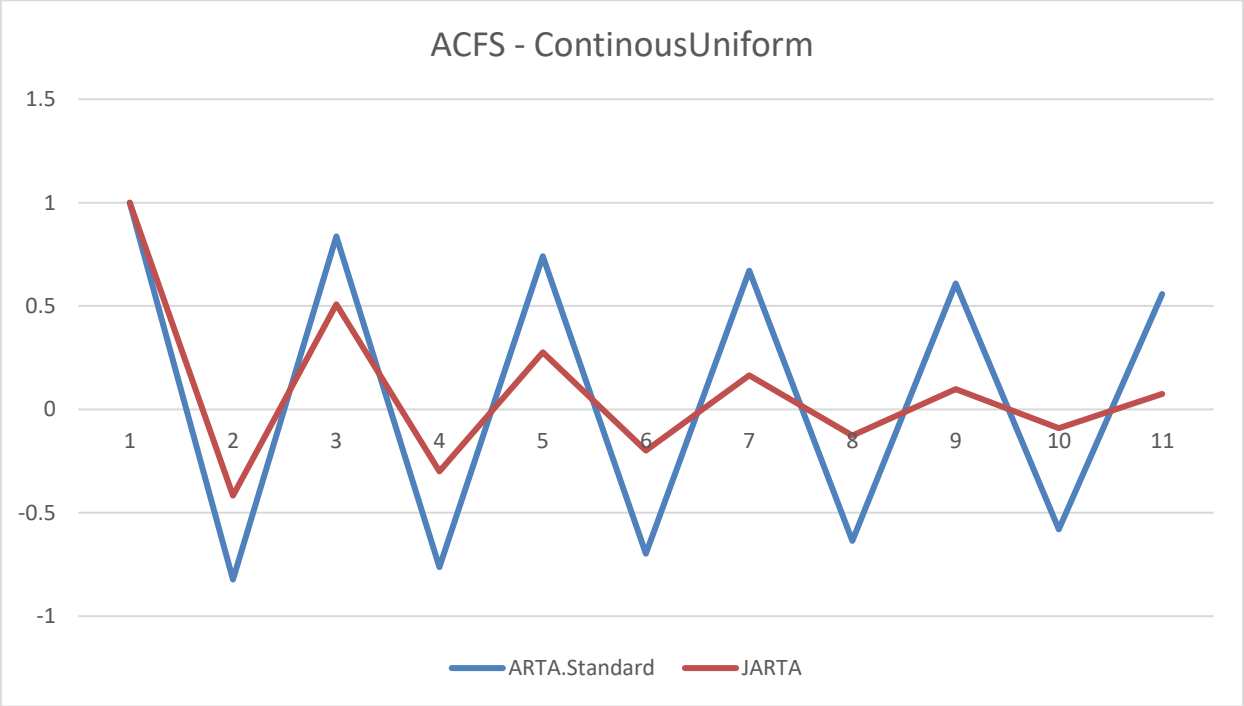


Abbildung 17: ACFS im Vergleich mit ContinousUniform (-1, 1)

Arta.Standard	JARTA
1	1
-0.822630072	-0.4165822
0.836934856	0.5090079
-0.762535905	-0.2992634
0.741310287	0.27696673
-0.698079634	-0.1987881
0.670680655	0.16508882
-0.635117809	-0.1258744
0.609157037	0.09810887
-0.580379504	-0.0899578
0.558146278	0.07513454

8.2.2 Normal

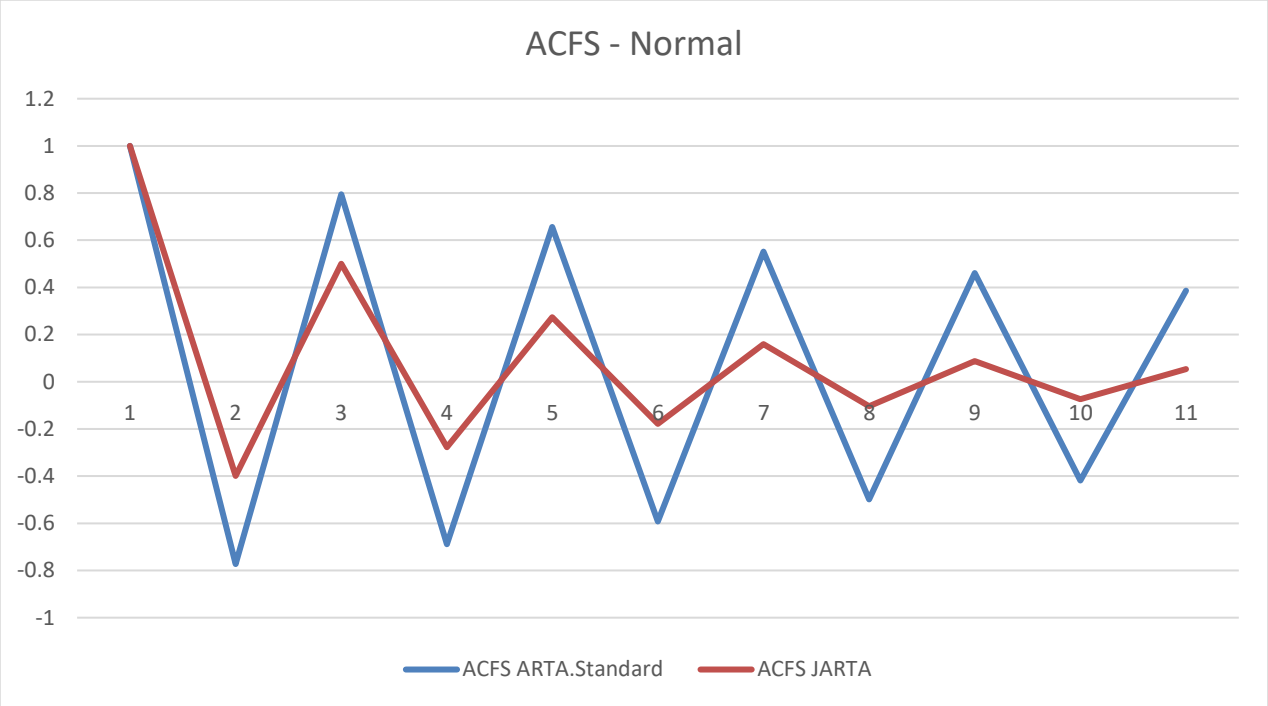


Abbildung 18: ACFS Vergleich Arta.Standard und JARTA, mit N (0,1)

Arta.Standard	JARTA
1	1
-0.772725676	-0.399101088
0.794123716	0.499673667
-0.688091837	-0.276768674
0.656300037	0.273625941
-0.592528152	-0.178230151
0.551392556	0.159867604
-0.499081516	-0.103811582
0.460924491	0.087651036
-0.418321714	-0.07322328
0.385498154	0.053287197

8.2.3 Exponential

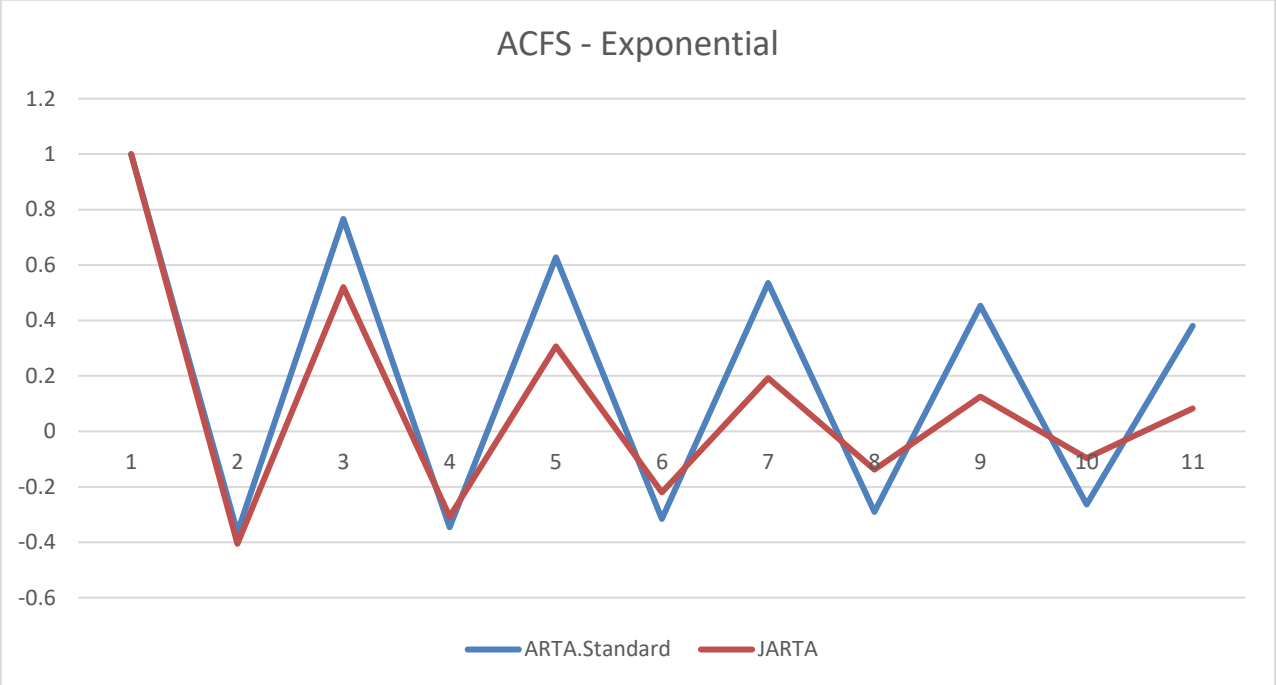


Abbildung 19: Die ACFS im Vergleich, Exponentialverteilung

Arta.Standard	JARTA
1	1
-0.352431384	-0.4057
0.769765957	0.519637
-0.333929006	-0.30831
0.61893006	0.306799
-0.310477583	-0.22037
0.518574645	0.192266
-0.284221243	-0.13807
0.431806635	0.124674
-0.256415253	-0.09608
0.358656033	0.083002

8.3 Vergleich PACFS

Bei den PACFS der ContinousUniform und Normalverteilung sind die Werte beinahe Deckungsgleich, lediglich die ersten vier Werte weichen stärker voneinander ab, anschliessend verlaufen sie beinahe kongruent und nähern sich immer weiter Null an. Bei der PACFS der Exponentialverteilung können grössere Abweichungen festgestellt werden, auch während der Annäherungsphase gegen das Ende. Beim Vergleich der effektiven Werte können wir in unseren die gleiche Struktur wie in den JARTA-Werten wiederfinden.

8.3.1 Continous Uniform

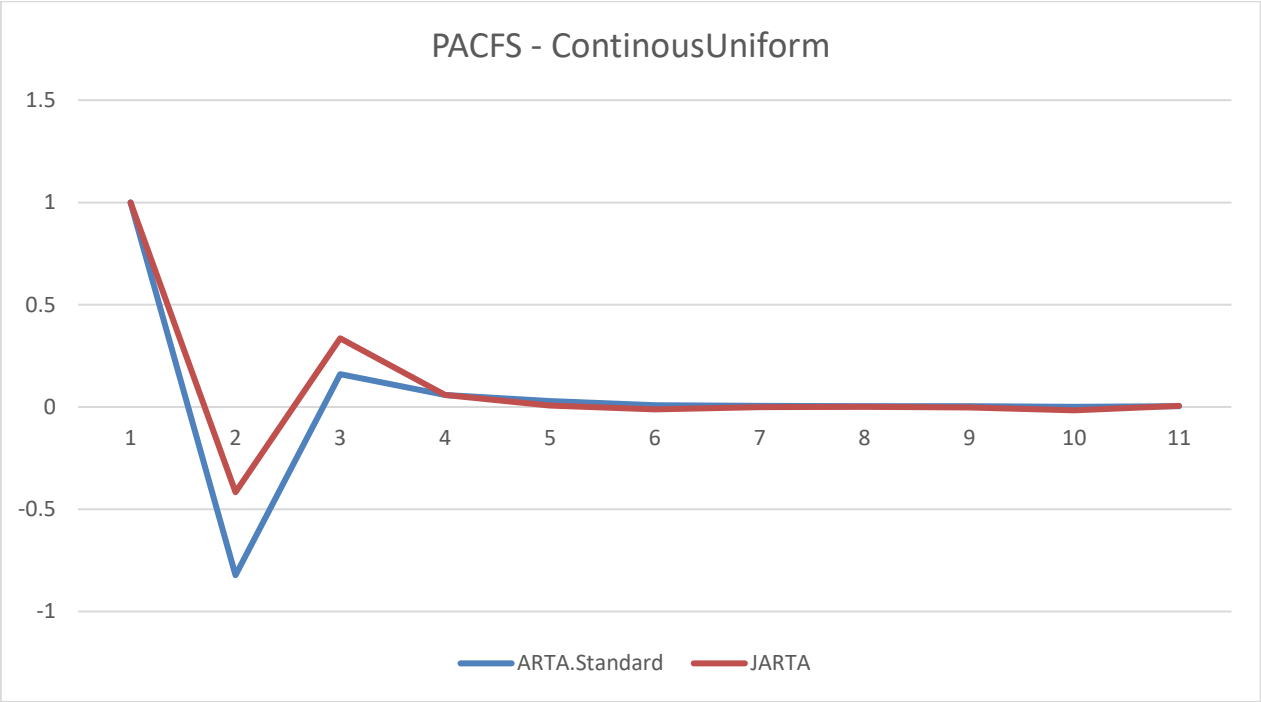


Abbildung 20: PACFS im Vergleich mit ContinousUniform (-1, 1)

Arta.Standard	JARTA
1	1
-0.822630072	-0.416582159
0.160214621	0.335467202
0.059270643	0.059191148
0.029553697	0.007012443
0.00888604	-0.011560657
0.00558979	-0.001478902
0.00433623	6.32E-04
0.004504524	-0.00263461
0.001151885	-0.015718683
0.003586952	0.004676503

8.3.2 Normal

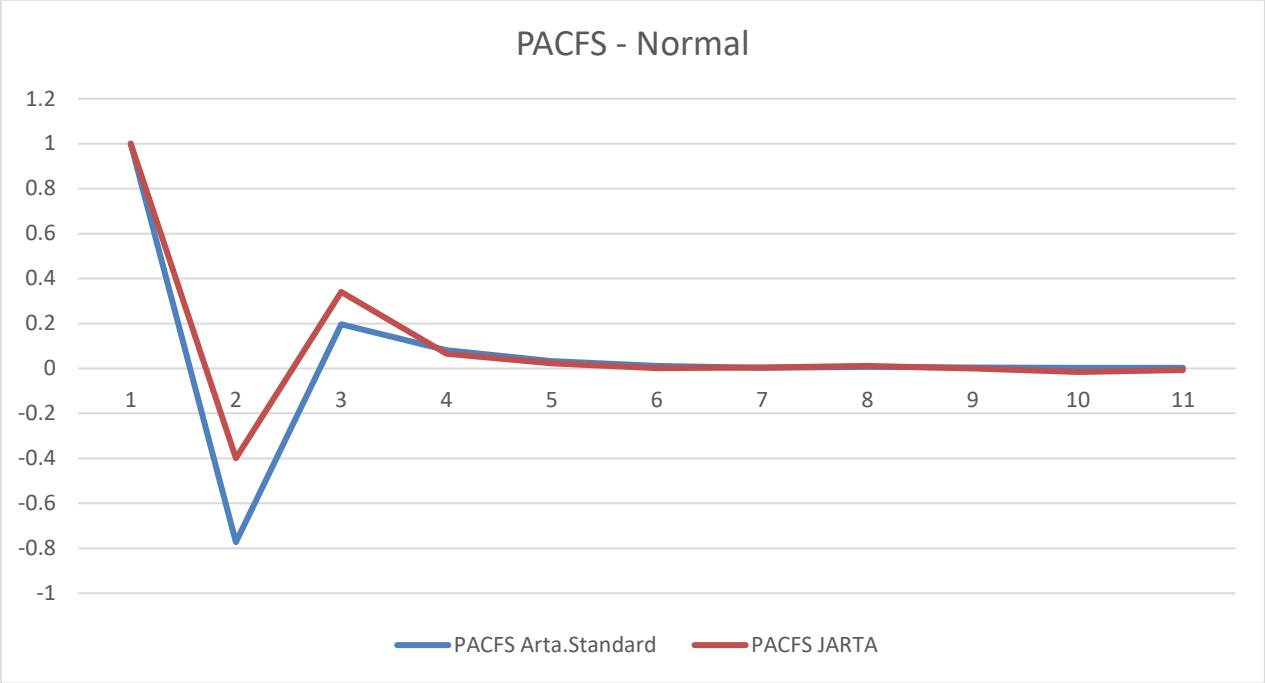


Abbildung 21: PACFS Vergleich Arta.Standard und JARTA, mit N (0,1)

Arta.Standard	JARTA
1	1
-0.772725676	-0.39910109
0.197018745	0.34039199
0.080930834	0.06616926
0.032123014	0.02214178
0.01125053	0.0010887
0.003249382	0.0037281
0.006603324	0.01094123
0.00395471	3.89E-04
0.003405857	-0.016045
0.002817221	-0.00777538

8.3.3 Exponential

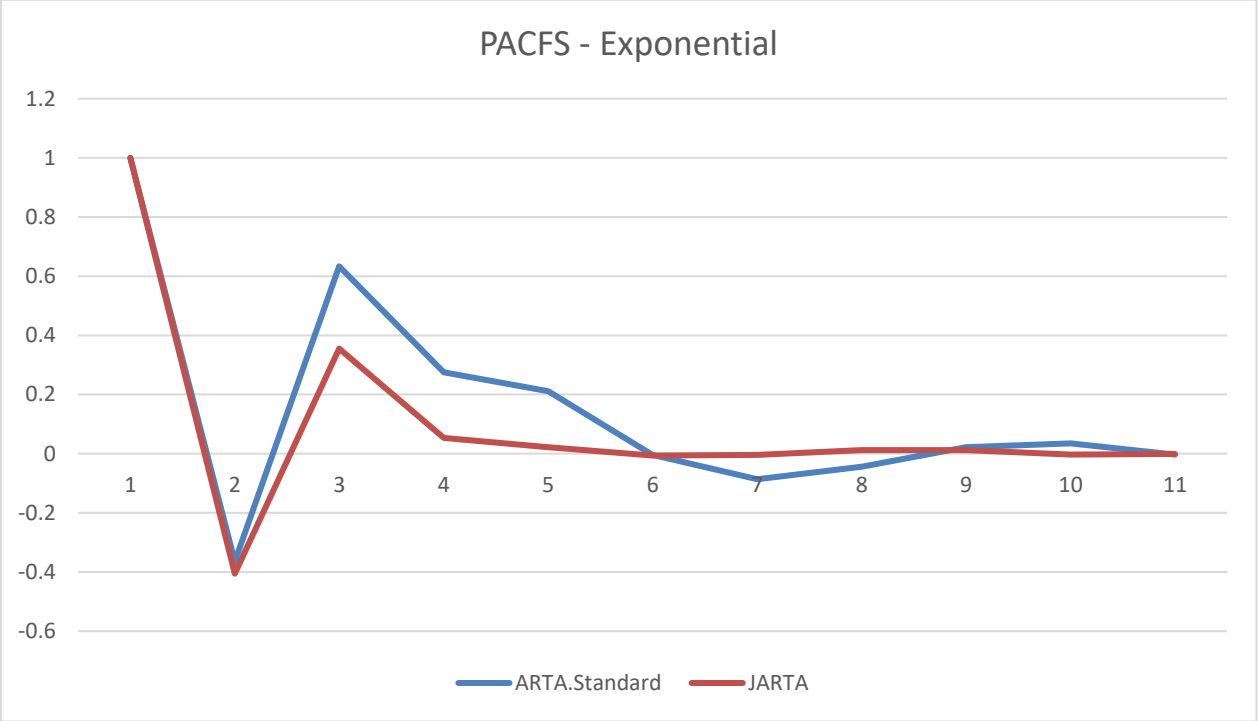


Abbildung 22: Vergleich der PACFS zwischen JARTA und Arta.Standard, in einer Exponentialverteilung

Arta.Standard	JARTA
1	1
-0.352431384	-0.4057
0.645558076	0.355045
0.282681961	0.053268
0.193684645	0.021615
-0.050560009	-0.00614
-0.1221241	-0.00387
-0.030407946	0.011617
0.061462457	0.011863
0.056167963	-0.00286
-0.016576183	-0.00109

8.4 Vergleich Arta.Standard und ContinousUniform

Ein weiterer Vergleich machen wir zwischen autokorrelierten Zufallszahlen und reinen Zufallszahlen. Dabei beschränken wir uns lediglich auf einen Vergleich. Wir erzeugen je 1000 ARTA-Zahlen und Zufallszahlen mit einer ContinousUniform-Verteilung. Als Basis dieser Zufallszahlen nutzen wir die Mathnet.Numerics Library. Die Verteilung ist durch ihre Boundaries von -1 und 1 beschränkt. Zur Analyse beschränken wir uns wiederrum auf die ersten 100 Zahlen.

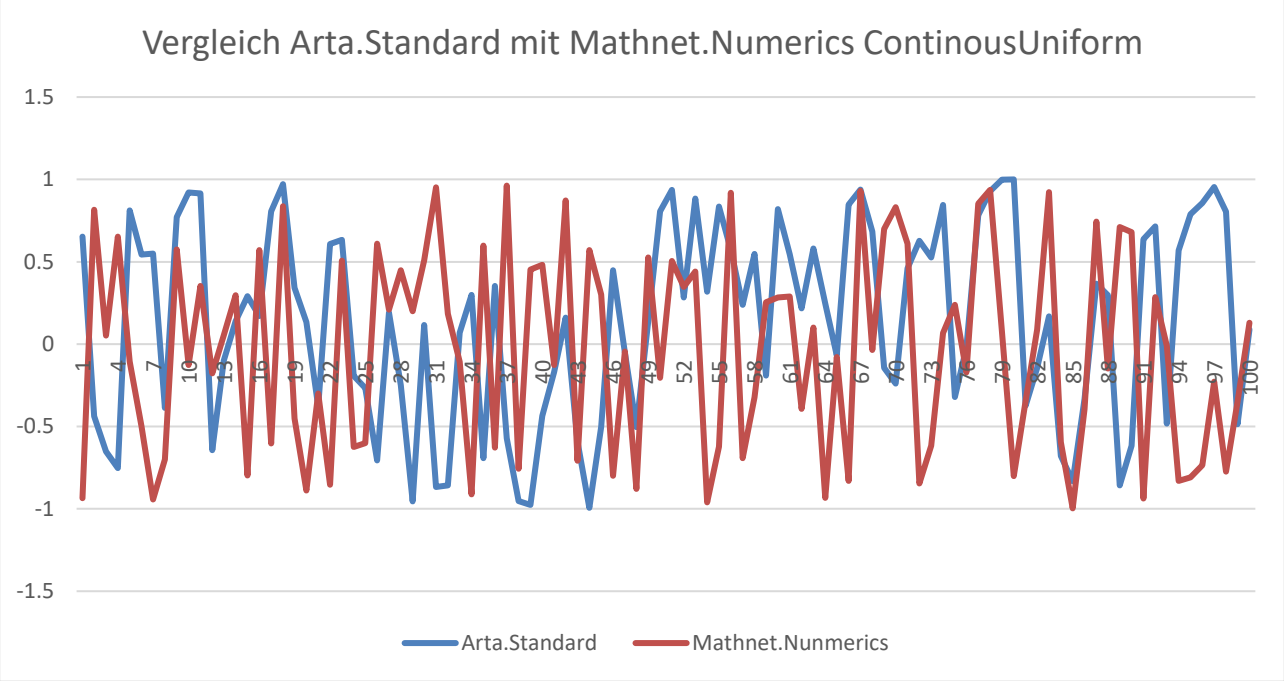


Abbildung 23: Vergleich zwischen Arta.Standard und Mathnet.Numerics

Die Zufallszahlen von Mathnet.Numerics zeigen häufigere und gleichverteiltere Sprünge zwischen negativen und positiven Zahlen, welche oft einen hohen numerischen Wert aufweisen. Solche Sprünge sind auch bei den Arta.Standard-Zahlen ersichtlich, jedoch erkennt man, dass sie immer durch ruhigere, flachere Phasen unterbrochen werden. Vergleich man nun diese Muster kann hier von Autokorrelation gesprochen werden, da sie sich immer wieder, mit nur leichten Veränderungen wiederholen.

9. Anwendungsfall und Simulation [bis 13.12.2017]

Als letzte Phase der vorliegenden Arbeit, nutzen wir die Klassenbibliothek bzw. die Simio-Erweiterung, um Simulationsprojekte mit autokorrelierten Zufallszahlen zu speisen. In einem ersten Schritt nutzen wir Arta.Simio in einer eigenen Simulation, anschliessend stellen wir das Beispiel eines Lagerhauses von Uhlig und Rank nach. Die zweite Simulation gibt uns die Möglichkeit, einen direkten Vergleich durchzuführen.

9.1 Vorbereitung

Als Vorbereitung zur Simulation haben wir ein ArtaElement und eine Source in ein eigenes Model gekapselt. Anschliessend haben wir zwei Referenzproperties, welche auf das ArtaElement, bzw. auf dessen Korrelationskoeffizienten zeigen, hinzugefügt. Daher können die Einstellungen direkt auf Stufe des Models getätigt werden. Weitere Referenzproperties wurden auf die wichtigsten Controls der Source gebunden.

9.2 Eigene Simulation

Das Ziel dieser Simulation ist es, die Unterschiede von den Simio-generierten Zufallszahlen und den ARTA-Zahlen zu beobachten und anschliessend auszuwerten. Dazu erstellen wir ein kleines Model, welches einmal mit ARTA-Zahlen und einmal mit Simiozufallszahlen gespiesen wird. Das Model unterscheidet sich lediglich in den Parametern der Source, welche die Entities erzeugt.
Unser Model soll einen vereinfachten Prozess eines Flughafens abbilden. Konkret simulieren wir die Gepäckabgabe. Dabei wird ein Entity als jeweils ein Gepäckstück angesehen, welches am Check-in abgegeben wird, eine Security-Prüfung durchläuft und anschliessend via Fahrzeug in ein Flugzeug verladen wird.

9.2.1 Simulationsaufbau

Das Model besteht aus fünf Elementen. Die ModelEntity „BaggageGenerator“ stellt dabei ein Gepäckstück dar, welche laufend von der Source entsprechend erzeugt werden. Von dort werden sie per Fliessband an einen Server weitergeleitet, welcher die Security-Prüfung darstellen soll. Anschliessend werden sie an einen weiteren Server geleitet. Dort werden die Gepäckstücke auf ein Fahrzeug verladen und anschliessend zum Flugzeug transportiert.

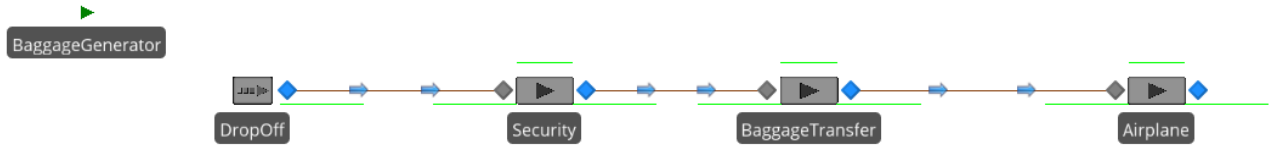


Abbildung 24: Produktionslinie

Das Model enthält den obenstehenden Aufbau zweimal, einmal wird die Source mit ARTA-Zahlen bestückt, die andere mit „normalen“ Zufallszahlen. Weiter wird beiden Output-Knoten der Sources ein Diagramm angefügt. Dieses zeichnet den Zeitpunkt des Verlassens eines Entities auf. Dies ermöglicht es uns, bereits zur Simulationszeit erste Vergleiche anzustellen.
Um dem Simulationsaufbau einen grösseren Bezug zur Realität zu geben, bestückten wir die einzelnen Elemente mit entsprechenden 3D-Modellen.

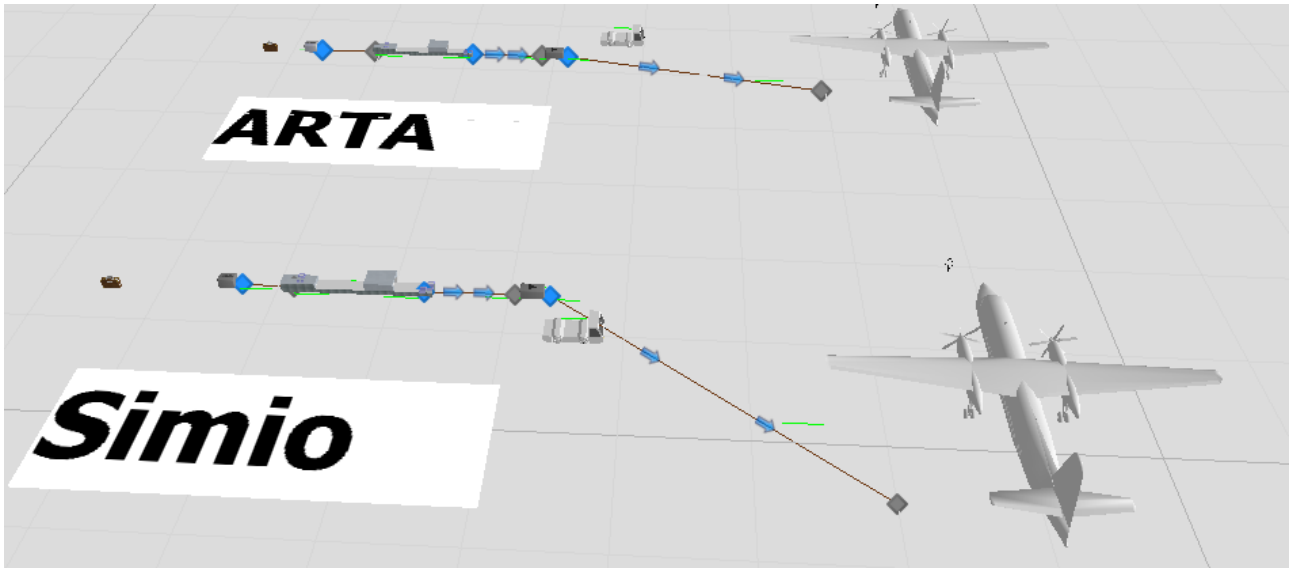


Abbildung 25: Komplettes Simulationsmodel

Durchlauf	Werte ArtaSource	Werte SimioSource
A	0.1	Random.Exponential(.25)
B	0.5	Random.Exponential(.25)
C	0.9	Random.Exponential(.25)
D	-0.5	Random.Exponential(.25)

Während dieser vier Durchläufe wollen wir folgendes Verhalten des Systems beobachten und anschliessend auswerten.

- Gesamtbild der InterarrivalTime
- Durchsatz über die gesamte Simulationsdauer
- Auswirkungen der verschiedenen Korrelationskoeffizienten

Diese Aspekte decken wir zum Teil als Experiment und als Graphen zur Simulationszeit ab.

9.2.2 Resultate

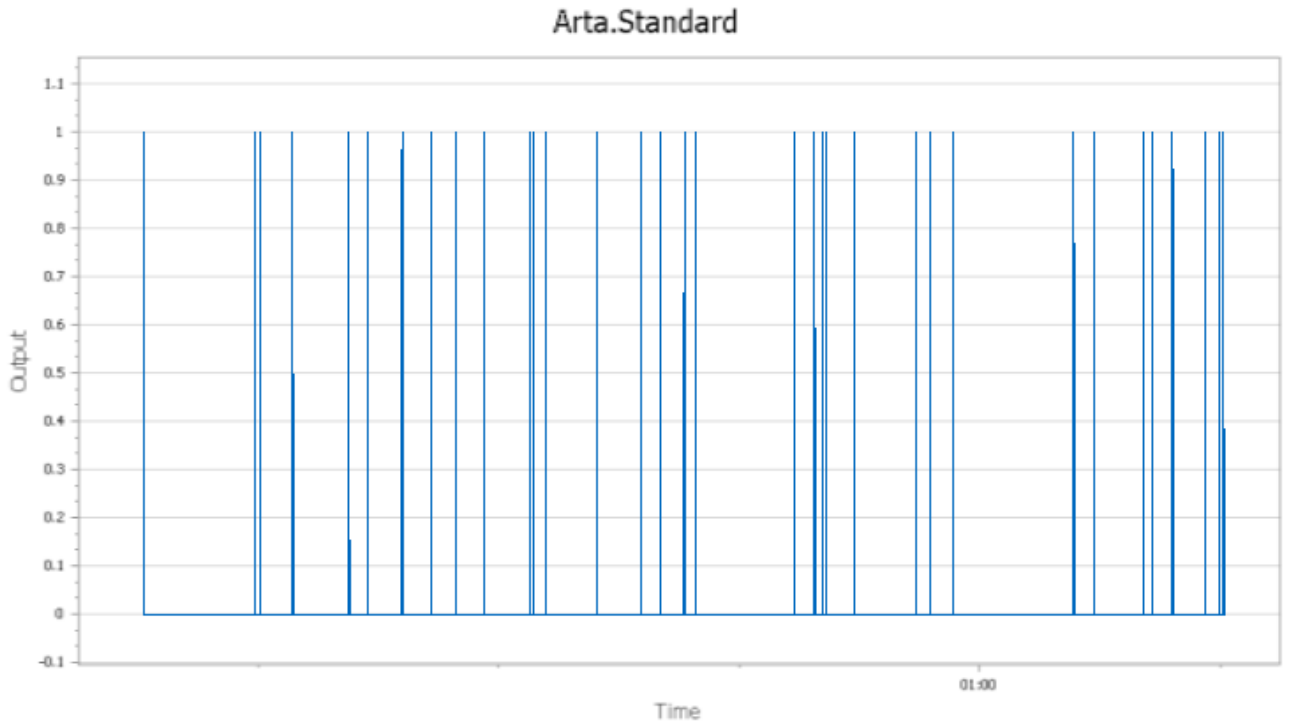
Folgende Resultate haben wir bezüglich der Anzahl der generierten Entities erhalten.

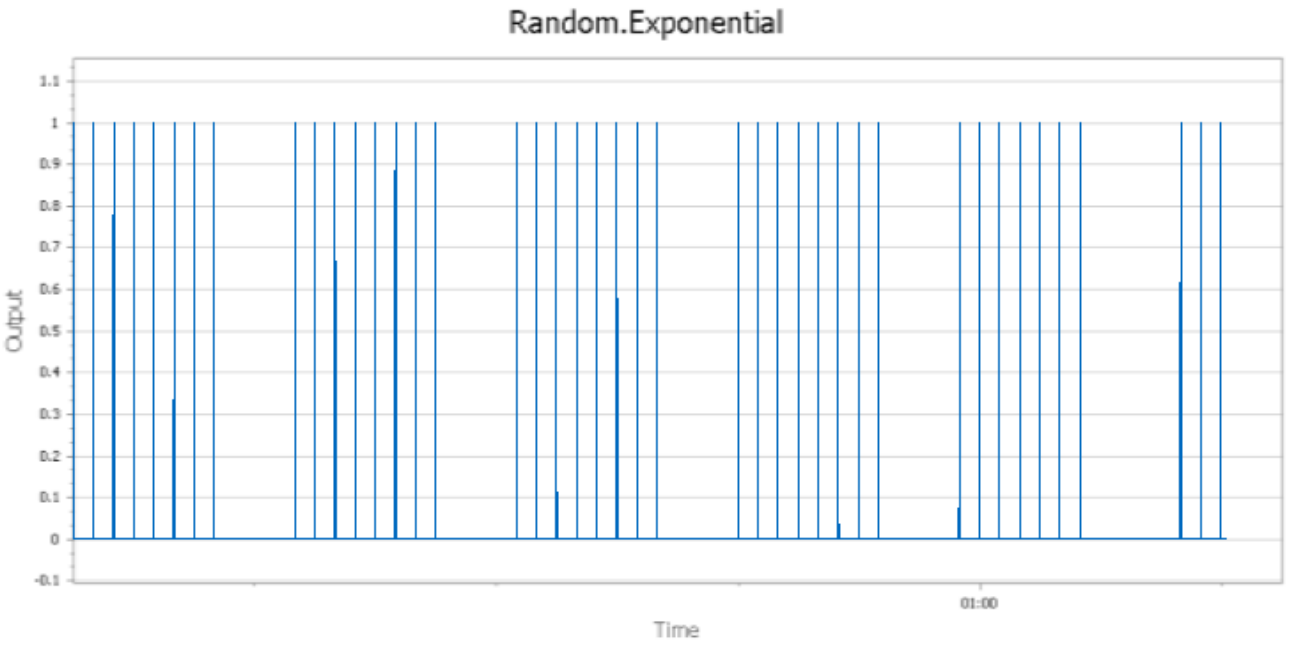
Durchlauf	Element	Anzahl erzeugte Entities
A	ArtaModel	1335
	SimioSource	1440
B	ArtaModel	1299
	SimioSource	1440
C	ArtaModel	1433
	SimioSource	1441
D	ArtaModel	1291
	SimioSource	1441

Die Anzahl der generierten Entities zeigt im Vergleich keine grossen Differenzen. Der spannende Aspekt hierbei sind die jedoch die Zeitabstände zwischen den einzelnen Entities. Diese Zeitabstände wollen wir in den folgenden Diagrammen darstellen und auswerten. Der betrachtete Zeitraum erstreckt sich jeweils über eine Stunde.

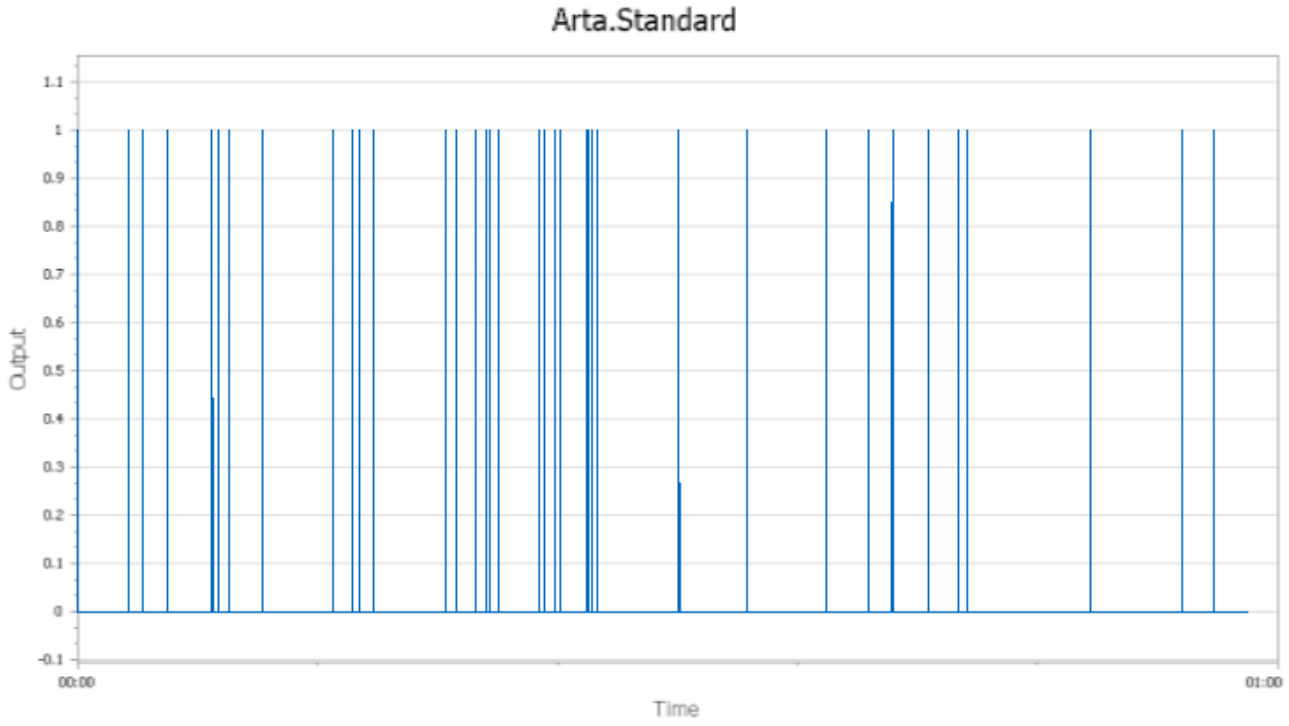
Schnell ersichtlich ist, dass die von Simio-generierten InterarrivalTimes sich stark von den ARTA-generierten Zeiten unterscheiden. Die Simio-Zeiten halten sich streng an das gleiche Muster, welches sich nach bestimmten Zeitabständen zu wiederholen scheint, wobei diese Zeitabstände immer denselben Wert besitzen.
Die mit ARTA-Zahlen gespiesene Source erzeugt die Zahlen ebenfalls in gewissen Intervallen. Jedoch sind diese, im Vergleich mit den Simio-Zahlen, nicht immer gleichlang, sondern werden über die gesamte Simulationszeit immer grösser.

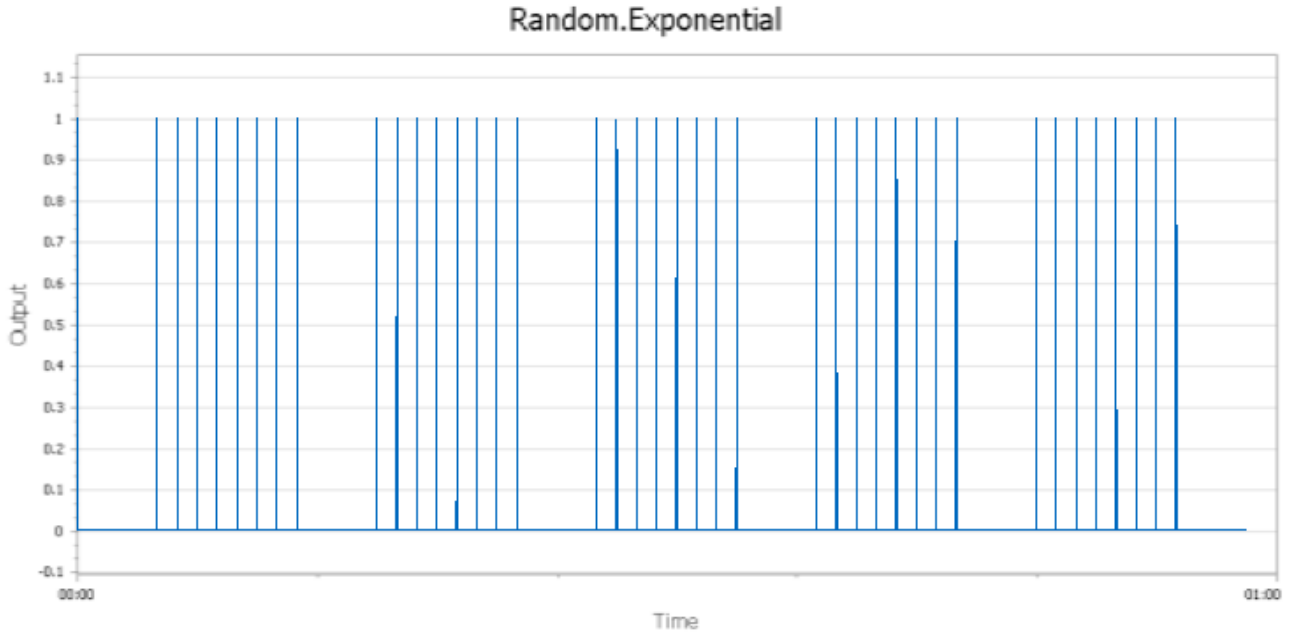
9.2.3 Zeitliches Verhalten – Durchlauf A



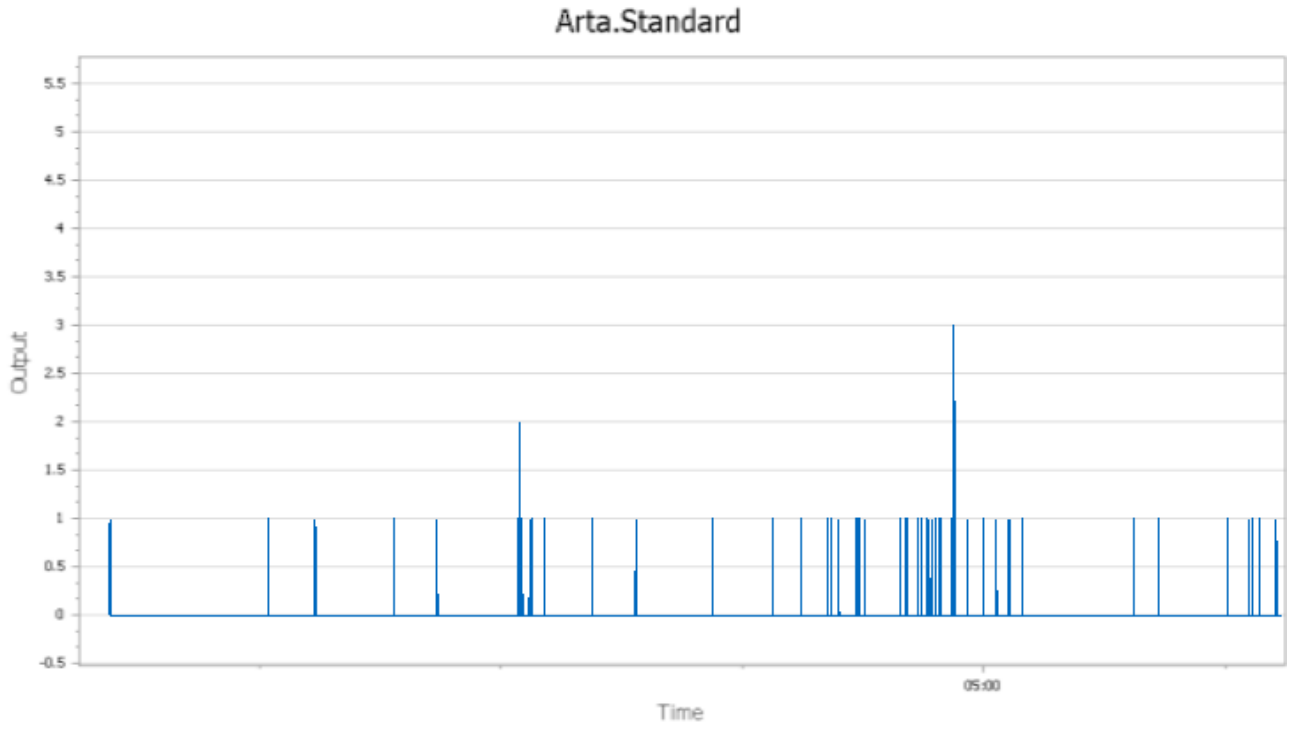


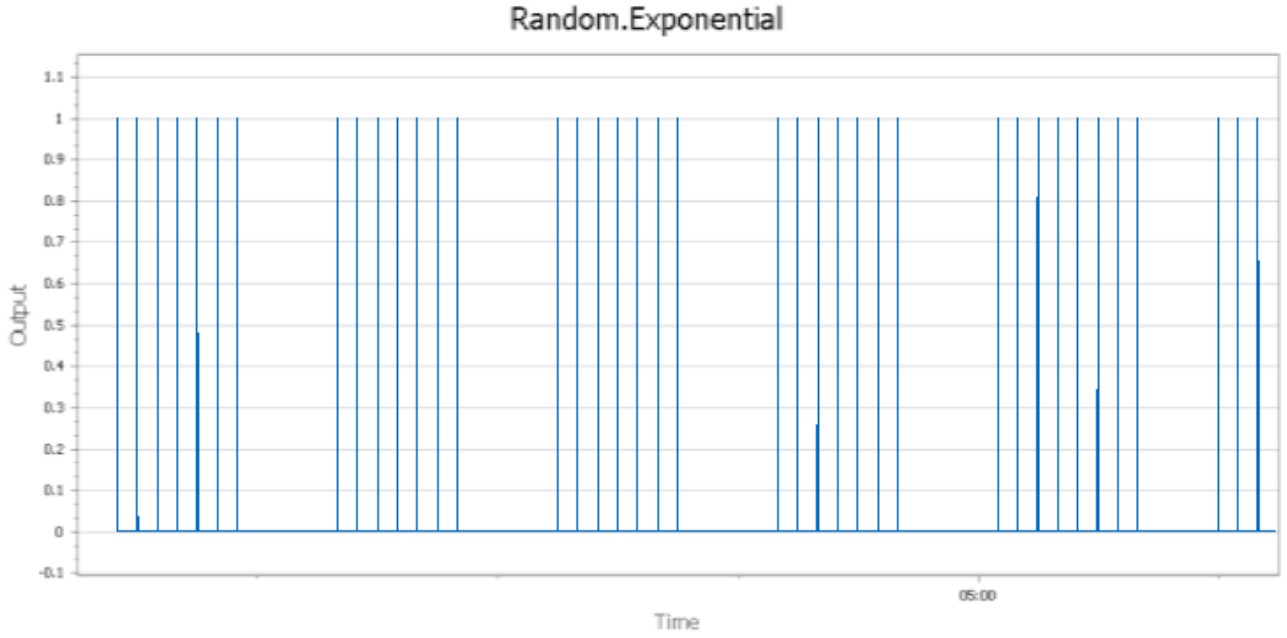
9.2.4 Zeitliches Verhalten - Durchlauf B



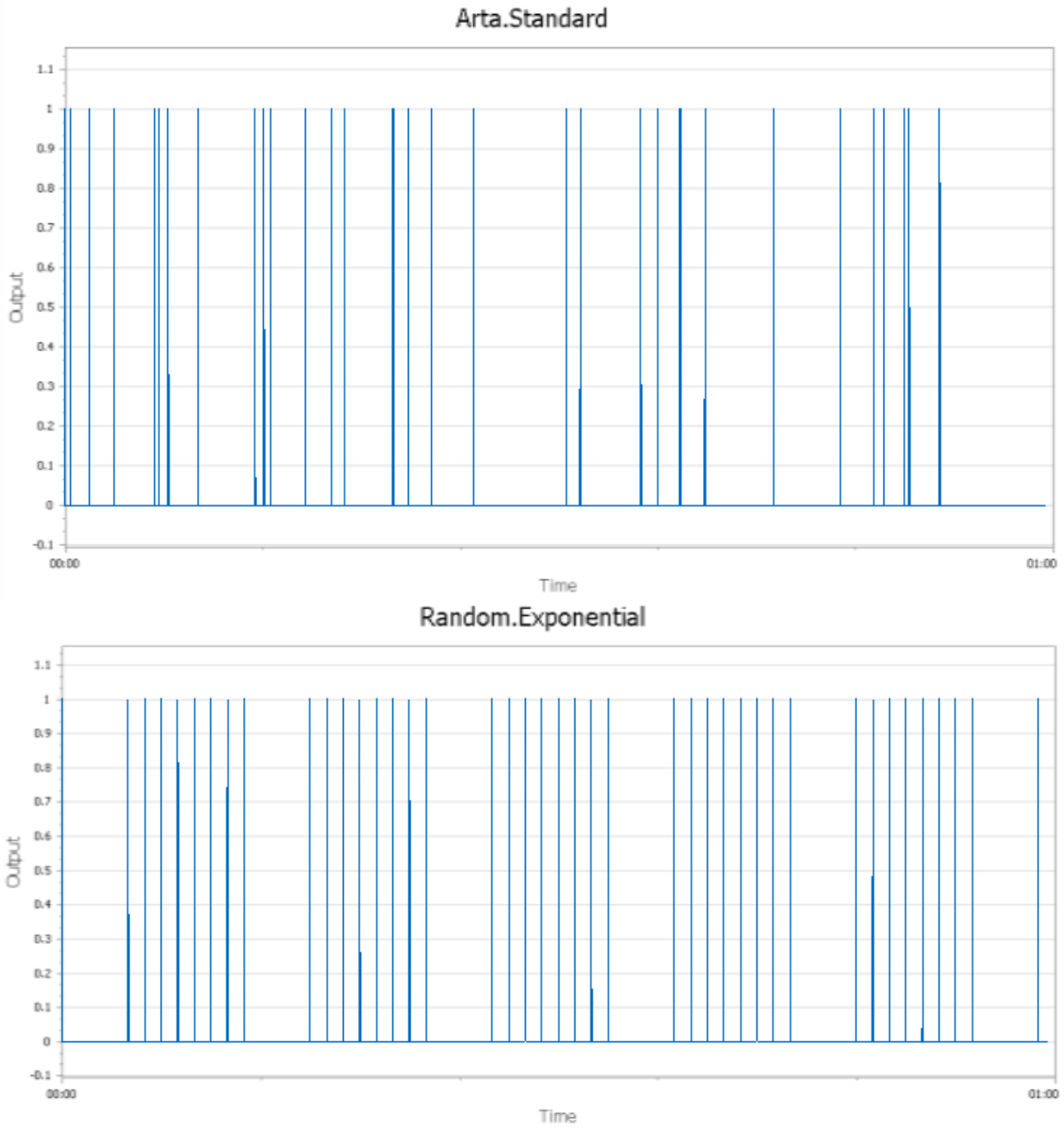


9.2.5 Zeitliches Verhalten – Durchlauf C





9.2.6 Zeitliches Verhalten – Durchlauf D



10. Schlussfolgerung und Ausblick [bis 20.12.2017]

Die angestrebten Ziele wurden erreicht. Einerseits liefert die erzeugte Klassenbibliothek autokorrelierte Zufallszahlen, wobei der Grad der Autokorrelation übe die Korrelationskoeffizienten gesteuert werden kann, andererseits ist ein Simio-AddIn realisiert worden, welches es ermöglicht, die Klassenbibliothek in einer Simulationssoftware zu nutzen.

10.1 Mathematische Grundlagen/Auswertung

Durch unsere Auswertung ist ersichtlich geworden, dass Arta.Standard Zufallszahlen erzeugt, welche der JAVA-Implementation sehr ähnlich sind. Als einziger Unterschied können die ACFS und PACFS gesehen werden, welche sich in unserer Implementation langsamer Null annähern. Die generierten Zahlen liegen jeweils in den von der Verteilung definierten Grenzen und wiesen das angestrebte autokorrelierte Muster auf.

10.2 Simulation

Arta.Standard wurde im Rahmen dieser Arbeit bereits innerhalb eines Simulationsmodells getestet. Als nächsten Schritt sehen wir die Verwendung in komplexeren Modellen. In den Testsimulationen wurde klar ersichtlich, dass unsere Bibliothek autokorrelierte Zufallszahlen liefert und sich dies von den systemgenerierten Zahlen stark unterscheiden. Als nächstes Experiment sehen wir die Verwendung bzw. Gegenüberstellung eines komplexen Systems, welches einen realen Anwendungsfall abbildet. So können die Unterschiede in Bezug auf reale Daten klar ersichtlich gemacht werden.
Einen spannenden Anwendungsfall sehen wir in der Nachbildung des Simulationsaufbaus, welcher zum Test von JARTA erzeugt wurde. Dies würde einen konkreten Vergleich der beiden Implementationen innerhalb einer Simulation zeigen.

10.3 Erweiterungspotential

Von der Implementationsseite her sehen wir ebenfalls Erweiterungspotential. Einerseits kann ArtaStatistics um eine grosse Funktionalität erweitert werden, um so einen umfassenderen, tieferen Einblick in den ARTA-Prozess zu geben. Eine von uns wichtig erachtete Funktionalität sehen wir in der graphischen Darstellung bzw. der Datenausgabe. Durch eine Erweiterung könnten die ARTA-Zahlen entweder direkt visualisiert oder formatiert ausgegeben werden. Weiter kann auch das Simio-AddIn in seiner Form weiterentwickelt werden. Der Ansatz liegt hierbei darin, dass die ARTA-Zahlen nicht durch eine Function in ein System gespiesen werden, sondern dass dies über Simio-Events geschieht.

11. Literaturverzeichnis

Pereira, D, Dez. 2012	Autocorrelation effects in manufacturing systems performance: a simulation analysis	4
T. Uhlig, O. Rose, S. Rank	JARTA — A Java library to model and fit Autoregressive-To-Anything processes	4, 20
Matsumoto, M.; Nishimura, T, 1998	Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator	13
Peter J. Brockwell, Richard A. Davis, 2006	*Time Series. Theory and Methods*	16

12. Quellen

Regression/Residuuen	https://de.wikipedia.org/wiki/Residuum_(Statistik)	7
Cryptool	https://www.cryptool.org/de/cryptool1	9
Giraffentext	https://de.wikipedia.org/wiki/Giraffen	11
Zeitreihenanalyse	Zeitreihenanalyse- Einstieg und Aufgaben von Thomas Mazzoni, FernUniversität in Hagen	15
Autokorrelation	Der Einfluss von Autokorrelation in komplexen Materialflusssystemen, Rank, Schmidt, Uhlig	15
Yule-Walker-Gleichungen	https://de.wikipedia.org/wiki/Yule-Walker-Gleichungen	16
Verteilungsfunktionen	https://de.wikipedia.org/wiki/Verteilungsfunktion	16
Zentraler Grenzwertsatz	https://de.wikipedia.org/wiki/Zentraler_Grenzwertsatz	16
Mathnet.Numerics	https://numerics.mathdotnet.com https://github.com/mathnet/mathnet-numerics	21

13. Abbildungsverzeichnis

14. Codefragmente