

# Studienarbeit

## Wissensgenerierung aus Daten - Simulation Verfahren zur Analyse von binären Sequenzen

des Studienganges Angewandte Informatik an der  
Dualen Hochschule Baden-Württemberg Mosbach



von

André Berberich (5894619)

Tobias Bloch (4048683)

Jarno Wagner (3880065)

TINF16A

Betreuer:

Prof. Dr. Dirk Saller

13.06.19

## Aufteilung der verfassten Kapitel:

Kapitel	Inhalte	Verfasser
1-3, 11	Begriffsherleitung und -definition	Jarno Wagner
4	Kontvertierungstool	Tobias Bloch
5-10	Analyseskripte	André Berberich

# Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema „Wissensgenerierung aus Daten - Simulation: Verfahren zur Analyse von binären Sequenzen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Diese Arbeit wurde bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mosbach, 13.06.19

Ort, Datum



Unterschrift André Berberich

Mosbach, 13.06.19

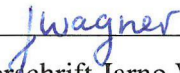
Ort, Datum



Unterschrift Tobias Bloch

Mosbach, 13.06.19

Ort, Datum



Unterschrift Jarno Wagner

# Kurzfassung

Die Arbeit befasst sich mit der Analyse von Sequenzen, die Zustände über einen zeitlichen Verlauf darstellen. Zu Beginn der Arbeit werden diese formal eingeführt und ihre Eigenschaften definiert. Auf Basis dieser Grundlagen werden binäre Sequenzen abgeleitet. Für diese wird ein Verfahren zur Analyse vorgestellt.

In der Regel handelt es sich bei Daten von Produktionslinien nicht um binäre Sequenzen, sondern um maschinen-spezifische Daten, die mehrere Zustände annehmen können. Die Menge der mögliche Zustände muss auf ein in der Arbeit entwickeltes Format konvertiert werden. Dabei findet eine Abbildung der Zustände auf einen binären Wertebereich statt. Ein entsprechendes Tool wurde im Zuge der Arbeit entwickelt.

Die konvertierten Daten werden von dem im Voraus genannten Verfahren analysiert. Dieses wurde durch automatisierte und anpassbare Skripte umgesetzt und wird in dieser Arbeit vorgestellt. Dabei werden PDF-Dateien generiert, die zu weiteren Analysen verwendet werden können.

# Abstract

The work deals with the analysis of sequences representing states over time. At the beginning of the work, these are formally introduced and their properties defined. Binary sequences are derived on the basis of these fundamentals. For these, a method for analysis is presented.

Generally, data from production lines are not binary, but machine-specific data that can assume several states. The set of possible states must be converted to a format developed in the thesis. The states are mapped to a binary value range. A corresponding tool was developed in the course of the work.

The converted data are analyzed by the previously developed procedure. This was implemented by automated and adaptable scripts and is presented in this thesis. PDF files are generated, which can be used for further analysis.

# Inhaltsverzeichnis

Inhaltsverzeichnis	vi
Abbildungsverzeichnis	viii
Listingverzeichnis	x
<b>1 Einleitung</b>	<b>1</b>
<b>2 Sequenzen</b>	<b>2</b>
2.1 Stand der Forschung . . . . .	2
2.2 Allgemeine Definitionen . . . . .	3
2.3 Binäre Sequenzen . . . . .	4
2.4 Eigenschaften . . . . .	5
2.5 Korrelation . . . . .	10
2.6 Klassifizierung . . . . .	11
<b>3 Wissensgenerierung - Vorgehen</b>	<b>14</b>
3.1 Datenaufbereitung . . . . .	15
3.2 Analyse . . . . .	17
<b>4 Daten Bereitstellung</b>	<b>21</b>
4.1 Datenformat . . . . .	21
4.2 Daten Generierung . . . . .	26
4.3 Daten Transformation . . . . .	30
<b>5 Implementierung des Analyseprojekts</b>	<b>32</b>
5.1 Warum Python? . . . . .	32
5.2 Verwendete Bibliotheken . . . . .	33
5.3 Installationsanleitung . . . . .	33
5.4 Verwendung von Docker . . . . .	34
5.5 Aufbau des Python Projekts . . . . .	34

<b>6</b>	<b>Kategorisierung von Sequenzen</b>	<b>37</b>
6.1	Kategorisierte Eigenschaften . . . . .	37
6.2	Beispiel-Skripte . . . . .	47
<b>7</b>	<b>Grundlagen: Kreuzkorrelation und Faltung</b>	<b>48</b>
7.1	Einführung Kreuzkorrelation . . . . .	48
7.2	Beispiele Kreuzkorrelation, Faltung und Autokorrelation . . . . .	49
7.3	Bedeutung der unterschiedlichen Modi . . . . .	51
<b>8</b>	<b>Implementierte Funktion zur Berechnung der Kreuz-Korrelation</b>	<b>53</b>
8.1	Beschreibung der Parameter . . . . .	53
8.2	Beispiel Ergebnisse . . . . .	55
8.3	Beispiel-Skripte . . . . .	59
<b>9</b>	<b>Suche nach Pattern mit Hilfe der Kreuz-Korrelation</b>	<b>60</b>
9.1	Angebotene Funktionen . . . . .	60
<b>10</b>	<b>Ausführung der automatisierten Skripte</b>	<b>64</b>
10.1	Anleitung zur Ausführung der Skripte . . . . .	64
10.2	Hinweis zur zusätzlichen Anzeige der generierten Daten . . . . .	65
10.3	Ablauf der automatisierten Ausführung . . . . .	68
10.4	Benennung der erzeugten Dateien . . . . .	69
10.5	Übernommene Metadaten . . . . .	69
10.6	Ergebnis der Ausführung . . . . .	70
10.7	Verbesserungsmöglichkeiten und Erweiterungen . . . . .	70
<b>11</b>	<b>Fazit</b>	<b>71</b>
	<b>Literatur</b>	<b>72</b>
<b>12</b>	<b>Anhang: Implementierung</b>	<b>74</b>
12.1	Beispiel: Ergebnisse im Quellverzeichnis . . . . .	74
12.2	Log-Meldungen des automatisierten Skripts . . . . .	79
12.3	Verwendung von Docker . . . . .	80
<b>13</b>	<b>Anhang: Literaturrecherche</b>	<b>82</b>

# Abbildungsverzeichnis

1	Sequenz mit sechs Zuständen . . . . .	4
2	Sequenz mit zwei Zuständen . . . . .	5
3	Balance ohne und mit fünf Teilsequenzen . . . . .	8
4	Frequenz ohne und mit fünf Teilsequenzen . . . . .	8
5	Zeitliche Korrelation zwischen zwei Sequenzen . . . . .	10
6	Sequenz mit einem Ausreißer . . . . .	11
7	Sequenz mit einem Zittern . . . . .	12
8	Sequenz mit einem Wechsel . . . . .	13
9	Fertigungslinie mit zwei Maschinen . . . . .	14
10	Produktionsdaten Fertigungslinie (roh) . . . . .	15
11	Produktionsdaten Fertigungslinie (reduziert) . . . . .	16
12	Maschine 1: Balance der 10er Subsequenzen . . . . .	17
13	Maschine 2: Balance der 10er Subsequenzen . . . . .	18
14	Maschine 1: Frequenz der 10er Subsequenzen . . . . .	18
15	Maschine 2: Frequenz der 10er Subsequenzen . . . . .	19
16	Kreuzkorrelation . . . . .	20
17	Use-Case-Diagramm der Generationsanwendung . . . . .	26
18	Aufruf des Programmes mit Kommandozeilenparameter . . . . .	26
19	Klassendiagramm für die Datenspeicherung . . . . .	31
20	Darstellung einer ausgeglichenen Sequenz . . . . .	38
21	Darstellung einer unausgeglichenen Sequenz . . . . .	39
22	Darstellung einer niederfrequenten Sequenz . . . . .	40
23	Darstellung einer mittelfrequenten Sequenz . . . . .	40
24	Darstellung einer hochfrequenten Sequenz . . . . .	41
25	Darstellung einer zu analysierenden Sequenz . . . . .	43
26	Darstellung der Sub-Sequenzen im Bezug auf die Balance . . . . .	44



---

27	Darstellung der Sub-Sequenzen im Bezug auf die Frequenz . . . . .	45
28	Darstellung der Sub-Sequenzen im Bezug auf die Frequenz einer mittelfre- quenten Sequenz . . . . .	46
29	Darstellung der Sub-Sequenzen im Bezug auf die Frequenz einer hochfrequen- ten Sequenz . . . . .	47
30	Vergleich zwischen Faltung, Kreuzkorrelation und Autokorrelation . . . . .	49
31	Beispiel 1 für die Faltung . . . . .	50
32	Beispiel 2 für die Faltung . . . . .	51
33	Ergebnis: Default-Parameter . . . . .	56
34	Ergebnis: plotNormalizedData . . . . .	56
35	Ergebnis: plotCorrelations . . . . .	57
36	Ergebnis: plotNonNormalizedResults . . . . .	57
37	Ergebnis: subtractMeanFromResult = False . . . . .	58
38	Ergebnis: subtractMeanFromResult = True . . . . .	58
39	Patternsuche: Kreuz-Korrelation . . . . .	62
40	Zusätzliche Anzeige der Ergebnisse . . . . .	67
41	Anhang: Quellverzeichnis vor der Ausführung . . . . .	74
42	Anhang: Quellverzeichnis nach der Ausführung . . . . .	74
43	Anhang: Ergebnis der Kategorisierung . . . . .	75
44	Anhang: Frequenz der Sub-Sequenzen . . . . .	76
45	Anhang: Balance der Sub-Sequenzen . . . . .	77
46	Anhang: Ergebnis der Kreuzkorrelation . . . . .	78

# Listingverzeichnis

1	Standard CSV mit Binär-Werten . . . . .	22
2	CSV mit gruppierten Binär-Werten . . . . .	23
3	CSV mit Headereintrag . . . . .	24
4	CSV mit mehreren Intervallreihen . . . . .	25
5	CSV mit Rohdaten . . . . .	30
6	Kommando zur Installation der notwendigen Pakete unter Windows . . . . .	33
7	Kommando zur Installation der notwendigen Pakete unter Linux/MacOS . . . . .	34
8	Kommando zur Installation der notwendigen Pakete unter Linux (Verwendung von pip3) . . . . .	34
9	Pseudo-Code zur Festlegung der Länge der Sub-Sequenzen . . . . .	42
10	Kommando zur Ausführung des Skripts unter Windows . . . . .	65
11	Kommando zur Ausführung des Skripts unter Linux . . . . .	65
12	Quelldatei mit Sequenzen (mit und ohne Metadaten) . . . . .	69
13	Anhang: Beispiel für generierte Log-Meldungen . . . . .	79
14	Anhang: Beispiel für die Verwendung von Docker . . . . .	80

# 1 Einleitung

Industriemaschinen erheben heutzutage eine große Menge an Daten. Zum einen versprechen sich Unternehmen einen Gewinn an Produktivität durch eine Entwicklung zur Industrie 4.0, zum anderen ist es durch günstige Sensorik überhaupt erst möglich geworden Maschinendaten in diesem Umfang zu erfassen<sup>1</sup>.

Mit Hilfe von Methoden und Werkzeugen des Data-Mining lassen sich diese Daten analysieren und so neue Zusammenhänge erkennen. Dies soll Unternehmen bei der zukünftigen Planung und Ausführung von Produktionsprozessen helfen.

Diese Arbeit beschäftigt sich, parallel zu weiteren Arbeiten des Studienganges Angewandte Informatik der DHBW Mosbach, mit der Grundlagenforschung in diesem Bereich. Mit Hilfe simulierter Daten soll der Aufbau sowie Eigenschaften von Produktionsdaten erforscht werden. Ziel dabei ist es, Sequenzen zu Charakterisieren und Klassifizieren.

Zu Beginn werden Sequenzen als binäre Datenstrukturen eingeführt. Diese bestehen nur aus den Zuständen *Fehler* und *kein Fehler*. Neben einer formalen Beschreibung werden die Eigenschaften solcher Sequenzen vorgestellt. Auf Grundlage dieser Eigenschaften soll es möglich sein, Sequenzen zu klassifizieren oder untereinander zu vergleichen. In einem beispielhaften Vorgehen werden die Möglichkeiten der Analyse auf Grundlage der theoretischen Inhalte herausgearbeitet.

Im zweiten Teil dieser Arbeit wird auf die technische Umsetzung einer automatisierten Auswertung von Sequenzen eingegangen. Dazu wird als erstes Beschrieben wie sich Daten in ein einheitliches und vergleichbares Format konvertieren lassen. Zudem wird ein Tool zur Generierung von Simulationsdaten, welches im Rahmen dieser Studienarbeit implementiert wurde, vorgestellt. Daran anschließend wird die Umsetzung eines weiteren Tools, zur automatisierten Analyse von simulierten Daten oder Produktionsdaten, erläutert.

---

<sup>1</sup> Gillhuber 2019

## 2 Sequenzen

Bei den in dieser Arbeit verwendeten Datenstrukturen handelt es sich um *Sequenzen*. Im Folgenden werden Sequenzen formal definiert und Begriffe zum Beschreiben dieser eingeführt. Daneben wird die binäre Sequenz als spezielle Abstraktion vorgestellt. Anschließend wird auf die zeitliche Korrelation zwischen Sequenzen eingegangen sowie Muster für eine Klassifizierung von Teilsequenzen definiert.

Zuerst wird allerdings noch ein Blick auf bestehende Definitionen für *Sequenz* geworfen. Das Wort stammt aus dem Lateinischen und bedeutet Folge<sup>2</sup>. Allgemein versteht man darunter eine Folge von gleichartigen oder ähnlichen Dingen<sup>3</sup>. Der Duden definiert im Kontext der Computerwissenschaften zudem eine Sequenz als Folge von hintereinander gespeicherten Daten<sup>4</sup>. Da es sich bei Produktionsdaten um eben dies handelt, lässt sich der Begriff *Sequenz* in dieser Arbeit verwenden.

Bei den hier verwendeten Sequenzen handelt es sich um nominalskalierte Daten<sup>5</sup>. Diese können weder in eine natürliche Reihenfolge gebracht werden, noch lassen sich Rechenoperationen wie Addition oder Multiplikation auf ihnen anwenden. Dies muss bei der weiteren Arbeit mit Sequenzen bedacht werden.

### 2.1 Stand der Forschung

Den aktuellen Stand der Forschung und bisher geleistete Arbeit in diesem Gebiet sind nicht einfach herauszuarbeiten. Besonders im Bereich der Wissensgenerierung aus Produktionsdaten gestaltet es sich schwer Grundlagenliteratur zu finden. Ein Großteil der Veröffentlichungen beschäftigt sich auf einem höheren Level mit diesem Thema. Dagegen gibt es mehr wissen-

---

<sup>2</sup> Duden 2019

<sup>3</sup> Wiktionary 2019

<sup>4</sup> Duden 2019

<sup>5</sup> Stevens 1946

schaftliche Grundlagenliteratur im Bereich der Bioinformatik, die sich mit diesem Thema auseinandersetzt. Unter dem Stichwort *Sequenzanalyse* findet man bspw. Algorithmen zur Analyse von DNA/RNA oder grundlegende Definitionen zum Umgang mit Sequenzen<sup>6</sup>.

Die Ergebnisse der Literaturrecherche befindet sich im Anhang 13.

## 2.2 Allgemeine Definitionen

Zur Simulation von Realdaten werden Sequenzen als eine endliche Menge von zeitdiskreten Ereignissen beschrieben. Ein Ereignis wird durch ein *Symbol* repräsentiert. Die Zusammenfassung aller Symbole einer Sequenz nennt man *Alphabet* und wird durch  $\Sigma$  dargestellt.

**Beispiel 2.2.1.** Alphabet mit den Symbolen 0, 1 und 2:  $\Sigma_1 = [0, 1, 2]$  oder mit den Symbolen a und b:  $\Sigma_2 = [a, b]$ .

Eine Sequenz besitzt zudem eine Indexmenge  $I$ . Diese ist eine endliche linear geordnete Menge von 0 bis  $n - 1$  mit der Mächtigkeit  $n$ . Diese entspricht der Länge der Sequenz.

**Beispiel 2.2.2.** Indexmenge  $I = [0, 1, 2, \dots, n - 1]$  mit der Mächtigkeit  $|I| = n$ .

Somit lässt sich eine Sequenz als Funktion  $s : I \rightarrow \Sigma$  schreiben. Jedem Element der Menge  $I$  (Index) wird ein Symbol des Alphabets  $\Sigma$  zugeordnet.

**Beispiel 2.2.3.** Aus  $\Sigma = [a, b]$  und  $n = 5$  folgt  $s = [a, b, a, a, b]$ .

Einen zusammenhängenden Bereich einer Sequenz bezeichnet man mit *Teilsequenz*  $t_{i,j}(s) = s[i, \dots, j]$ . Mit Hilfe von Teilsequenzen lassen sich Sequenzen detaillierter betrachten. Bei zeitdiskreten Datenströmen lassen sich so bestimmte Zeiträume ohne den Einfluss vorheriger oder folgender Ereignisse betrachten.

**Beispiel 2.2.4.** Aus  $s = [0, 0, 0, 0, 1, 0, 1, 0, 1]$  folgt  $t_{4,8}(s) = [1, 0, 1, 0, 1]$ .

Weiterhin lassen sich Sequenzen anhand ihrer Dimensionen beschreiben. Während eindimensionale Sequenzen an einer Stelle im Index nur einen Wert haben, haben mehrdimensionale Sequenzen mehr als einen Wert. In der Realität würde dies bspw. bedeuten, dass zu einem Zeitpunkt mehrere Informationen zum Zustand einer Maschine erfasst werden. Deshalb bietet es sich an, für jede Dimension ein eigenes Alphabet aufzustellen.

---

<sup>6</sup> Siehe: Rahmann 2014

**Beispiel 2.2.5.** Eine Sequenz mit den Alphabeten  $\Sigma_1 = [a, b]$  für die erste und  $\Sigma_2 = [A, B, C]$  für die zweite Dimension:  $s = [\{a, C\}, \{b, B\}, \{a, B\}, \{a, A\}]$ .

## 2.3 Binäre Sequenzen

Auf Grundlage der formalen Beschreibung wird für die folgende Arbeit die binäre Sequenz eingeführt. Eine binäre Sequenz ist eindimensional und kann zwei mögliche Zustände besitzen. Dazu wird das Alphabet  $\Sigma = [0, 1]$  mit 0 für *keinen Fehlerzustand* und 1 für *Fehlerzustand* eingeführt.

Ziel dieses Ansatzes ist eine größtmögliche Vereinfachung von Realdaten zu erzielen. Es wird angenommen, dass der Betrieb einer Produktionsmaschine sich entweder in einem gewünschten oder unerwünschten Zustand befindet. Unter letzterem versteht man jegliches ungeplante Verhalten, weshalb Vorgänge, wie bspw. das Umrüsten einer Maschine als erwünschten und somit fehlerfreien Zustand gezählt werden.

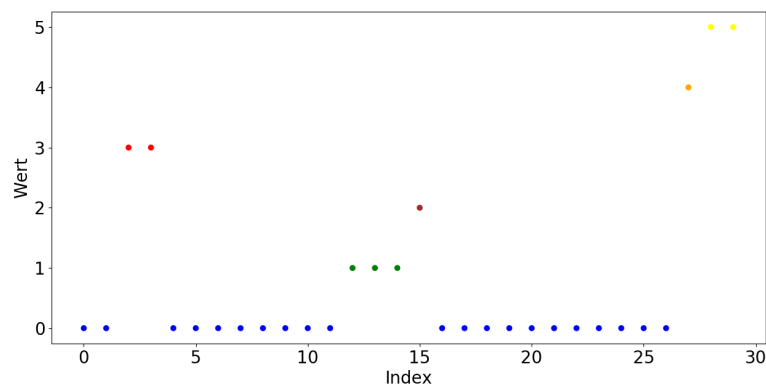


Abbildung 1: Sequenz mit sechs Zuständen

In Abbildung 1 sieht man eine Sequenz mit sechs möglichen Zuständen. Die Werte 0, 1, 2 stehen dabei für ein fehlerfreies und die Werte 3, 4, 5 für ein fehlerhaftes Verhalten der Produktionsmaschine. Durch das Zusammenfassen in diese beiden Kategorien, wird eine Vereinfachung der Sequenz erzielt (vgl. Abbildung 2). Das die Reduktion auf zwei Zustände mit einem gewissen Informationsverlust einhergeht, muss bei der weiteren Arbeit mit den Sequenzen beachtet werden.

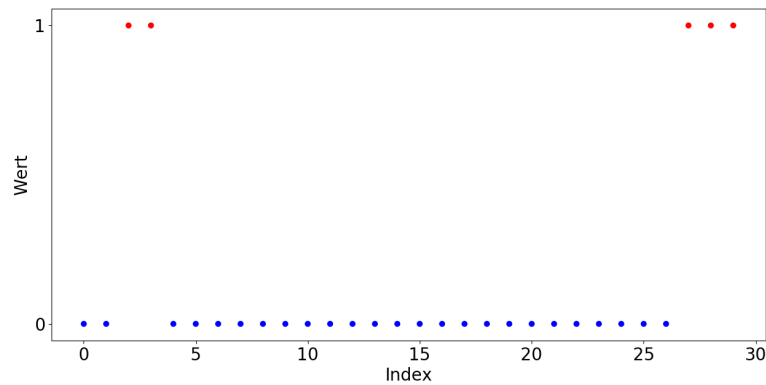


Abbildung 2: Sequenz mit zwei Zuständen

## 2.4 Eigenschaften

Um Sequenzen oder Teilsequenzen zu klassifizieren, müssen Eigenschaften definiert und voneinander abgegrenzt werden. Dazu werden die *Balance* und *Frequenz* einer Sequenz eingeführt.

### Balance

Mit der Balance wird das Verhältnis des Vorkommens eines Symbols zur Länge der Sequenz beschrieben.

$$Balance(Symbol) = \frac{Anzahl(Symbol)}{Gesamtlänge\ der\ Sequenz} \quad (2.1)$$

$$Balance(0) + Balance(1) = 1; \text{ für } \Sigma = [0, 1] \quad (2.2)$$

Man kann die Balance somit auch als Maß für die Ausrichtung einer Sequenz zu einem Symbol hin bezeichnen. Die Balance wird nach der Formel 2.1 berechnet. Für eine binäre Sequenz gilt zudem Formel 2.2. Mit ihr ist es möglich  $Balance(1)$  aus  $Balance(0)$  und anders herum zu berechnen.

**Beispiel 2.4.1.** Gegeben sind:

- Sequenz:  $s = [1, 0, 0, 0, 1, 0, 1, 1, 1, 1]$
- Alphabet:  $\Sigma = [0, 1]$
- Länge:  $|s| = 10$

Daraus folgt:

$$\text{Balance}(0) = \frac{\text{Anzahl}(0)}{|s|} = \frac{4}{10} = 0.4$$

$$\text{Balance}(1) = 1 - \text{Balance}(0) = 1 - 0.4 = 0.6$$

## Frequenz

Daneben ist die Frequenz ein Maß für die Aktivität an Wertewechseln innerhalb einer Sequenz. Sie ist das Verhältnis zwischen der Anzahl an Wertewechsel und der Sequenzlänge. Sie wird nach Formel 2.3 berechnet.

$$\text{Frequenz} = \frac{\text{Anzahl der Wertewechsel}}{\text{Gesamtlänge der Sequenz}} \quad (2.3)$$

Ein Wertewechsel tritt auf, wenn ein Element der Sequenz sich von seinem Nachfolger unterscheidet, also  $s_i \neq s_{i+1}$  ist. In einem Produktionsprozess könnte dies der Zeitpunkt sein, bei dem es zu einer Störung kommt, bzw. diese behoben wurde. Bei einer binären Sequenz handelt es sich um den Wechsel von 0 auf 1 oder von 1 auf 0.

**Beispiel 2.4.2.** Gegeben sind:

- Sequenz:  $s = [1, 0, 0, 0, 0, 0, 1, 1, 1, 1]$
- Alphabet:  $\Sigma = [0, 1]$
- Länge:  $|s| = 10$

Daraus folgt:

$$\text{Frequenz} = \frac{\text{Anzahl der Wertewechsel}}{|s|} = \frac{2}{10} = 0.2$$

Somit lassen sich mit Hilfe der Balance und Frequenz Aussagen darüber treffen, wie oft ein Fehlerzustand in einer Sequenz auftritt und ob es häufig zu Wertewechseln kommt.



## Teilsequenzen

Dabei bietet es sich an, besonders bei langen Sequenzen, sich auch die Eigenschaften von Teilsequenzen anzuschauen. Das folgende - etwas größere - Beispiel soll dies verdeutlichen.

**Beispiel 2.4.3.** Gegeben sind:

- Sequenz:  $s = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]$
- Alphabet:  $\Sigma = [0, 1]$
- Länge:  $|s| = 30$
- Teilsequenzen:  $t_{0,5}(s)$ ,  $t_{6,11}(s)$ ,  $t_{12,17}(s)$ ,  $t_{18,23}(s)$  und  $t_{24,29}(s)$

Daraus folgt mit den Formeln 2.1 und 2.3 für jede Teilsequenz sowie die gesamte Sequenz:

Sequenz	Balance(0)	Frequenz
s	$0.4\bar{3}$	$0.3\bar{6}$
$t_{0,5}(s)$	1	0
$t_{6,11}(s)$	$0.\bar{3}$	$0.1\bar{6}$
$t_{12,17}(s)$	0	0
$t_{18,23}(s)$	$0.\bar{3}$	$0.\bar{6}$
$t_{24,29}(s)$	0.5	$0.8\bar{3}$

Es zeigt sich, dass Balance und Frequenz in den Teilsequenzen teils stark variieren. Damit ergibt sich auch die Wichtigkeit, Sequenzen nicht nur als Ganzes, sondern auch deren Verhalten über die Zeit zu betrachten. So treten bis Index 12 immer mehr Fehlerzustände auf und werden ab Index 17 wieder weniger (vgl. Abbildung 3). Bei der Frequenz zeigen sich zu Beginn noch keine Wertewechsel, während zum Ende hin die Sequenz sehr aktiv wird (vgl. Abbildung 4).

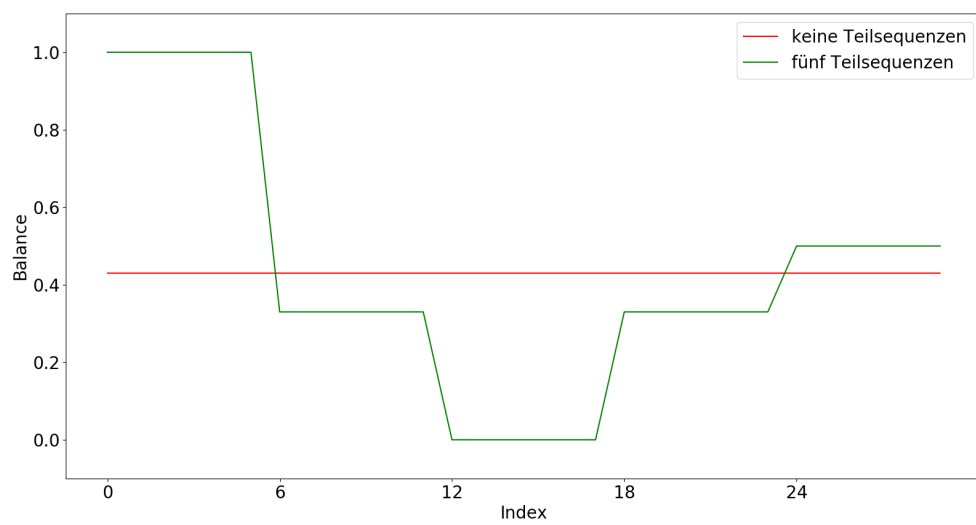


Abbildung 3: Balance ohne und mit fünf Teilsequenzen

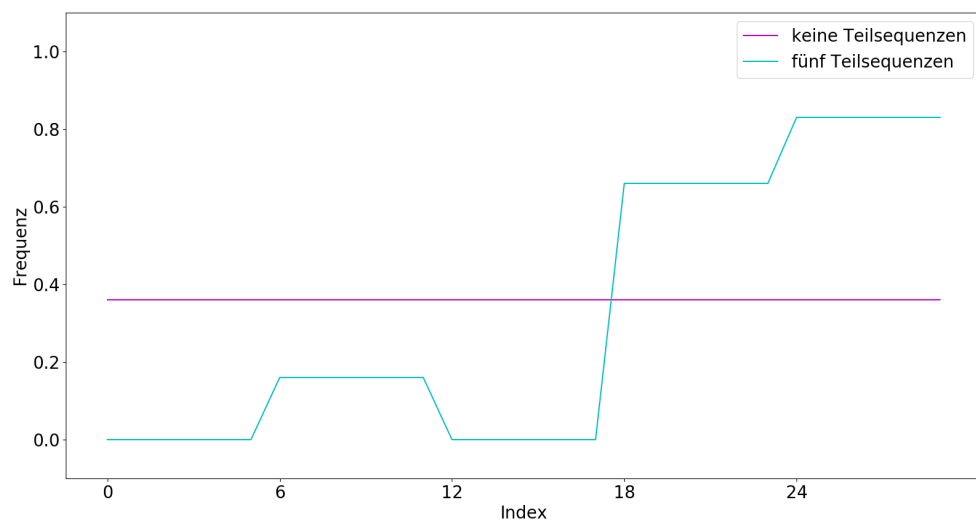


Abbildung 4: Frequenz ohne und mit fünf Teilsequenzen

## Wertebereich

Der kleinste Wert für die Balance oder Frequenz ist 0 und der größte kleiner gleich 1. Eine Sequenz mit einer niedrigen, bzw. hohen Balance wird als unausgeglichen beschrieben, während ein Wert um die 0.5 als ausgeglichen gilt. Dieser Ausgleich bezieht sich letztlich auf die Verteilung der Nullen und Einsen. Eine detaillierte Auflistung der unterschiedlichen Wertebereiche findet sich in Tabelle 1. Da die Balance für zwei Zustände berechnet werden kann, wird der Einfachheit halber - wenn von Balance gesprochen wird - implizit die Balance von *Fehlerzustand*, bzw.  $B(1)$  angenommen.

Balance	Beschreibung
$0 \leq B < 0.4$	unausgeglichen
$0.4 \leq B \leq 0.6$	ausgeglichen
$0.6 < B \leq 1$	unausgeglichen

Tabelle 1: Balance

Da mit der Frequenz ein Maß für die Wertewechsel innerhalb einer Sequenz bildet, entspricht ein Wert gegen 0 einer sehr inaktiven Sequenz mit nur sehr wenigen bis keinen Wertewechseln. Ein Wert von 1 steht hingegen für eine hochfrequente Sequenz, in der sehr viel Wertewechsel stattfinden. Eine detaillierte Auflistung der unterschiedlichen Wertebereiche findet sich in Tabelle 2.

Frequenz	Beschreibung
$0 \leq F \leq 0.25$	niederfrequent
$0.25 < F < 0.75$	mittelfrequent
$0.75 \leq F < 1$	hochfrequent

Tabelle 2: Frequenz

Mit der Balance und Frequenz sind nun zwei Eigenschaften definiert, mit denen sich grundsätzliche Aussagen über Sequenzen treffen lassen. Aus betriebswirtschaftlicher Sicht ist es erstrebenswert, die Balance von *Fehlerzustand* möglichst niedrig zu halten. Eine Balance von 0 wäre der Idealzustand, da somit kein Fehlerzustand in der Sequenz vorhanden ist.

Wenn man die Frequenz einer Sequenz betrachtet, lässt sich grundsätzlich sagen, dass eine niederfrequente Sequenz ein wünschenswerter Zustand ist, denn wenige Wertewechsel sprechen für eine gewisse Kontinuität und Stabilität. Allerdings sollte man die Frequenz zusammen mit der Balance und nicht alleine betrachten. Denn eine niedrige Frequenz im Zusammenspiel mit einer hohen Balance würde bedeuten, dass sich eine Maschine konstant im Fehlerzustand befindet.

## 2.5 Korrelation

Bisher wurden Sequenzen nur für sich alleine betrachtet und nicht im Zusammenhang mit Anderen. Dabei scheint dies sogar notwendig, wenn man sich vergegenwärtigt, dass Fertigungslinien oft aus mehreren unterschiedlichen Produktionsmaschinen bestehen. Diese bilden untereinander oft Abhängigkeiten, weshalb das Auftreten zeitlicher Korrelation erwartbar ist. Eine solche zeitliche Verschiebung ist in Abbildung 5 dargestellt. Hier lässt sich das Muster von Sequenz 1 in Sequenz 2 mit einer Verschiebung von 3 beobachten.

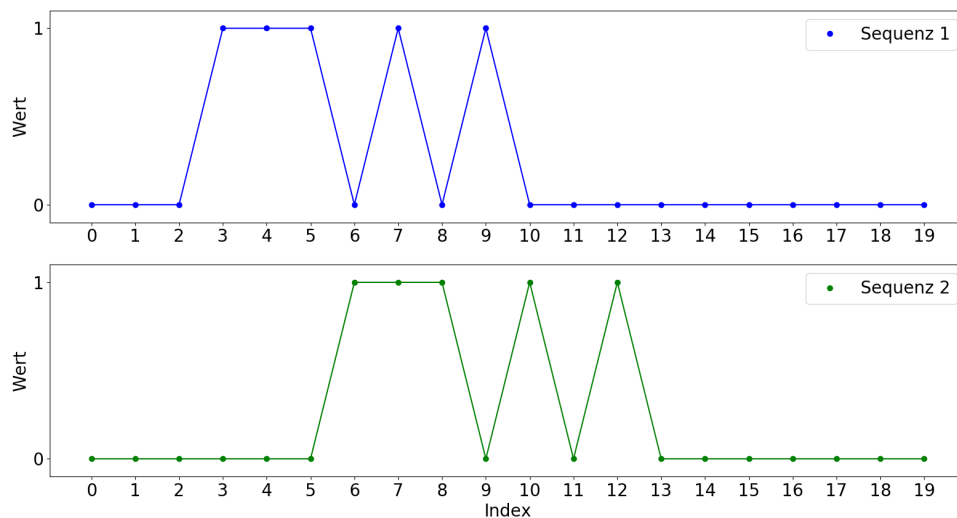


Abbildung 5: Zeitliche Korrelation zwischen zwei Sequenzen

Mit Hilfe der Korrelation als Maß für den Zusammenhang zwischen Sequenzen, wird eine solche Betrachtung möglich. In der Signalanalyse wird dafür die Kreuzkorrelationsfunktion verwendet. Auch wenn eine zeitliche Korrelation zwischen Sequenzen so nachgewiesen werden kann, ist dies noch kein Nachweis für einen kausalen Zusammenhang. Die Implementierung dieser Funktion sowie deren beispielhafte Verwendung findet sich in Kapitel 8.

## 2.6 Klassifizierung

Wenn man Sequenzen graphisch darstellt ergeben sich unterschiedliche Muster. Um die Klassifikation von Sequenzen zu erleichtern, wurden in Kapitel 2.4 die Wertebereiche für Balance und Frequenz definiert. Im Folgenden werden markante Muster, welche in Teilsequenzen auftreten können, beschrieben.

Das erste Muster nennt sich *Ausreißer* und ist in Abbildung 6 dargestellt. Eine solche Teilsequenz befindet sich fast ausschließlich in einem Zustand. Änderungen treten fast gar nicht und wenn doch nur kurz auf. Bei Produktionsprozessen kann man dieses Muster bei einer Maschine, welche im Regelbetrieb arbeitet und kurz einen Fehler wirft, beobachten. Wichtig dabei ist, dass schnell wieder ein Sprung in den Ausgangszustand stattfindet. Die Balance einer solchen Sequenz ist unausgeglichen und die Frequenz sehr niedrig.

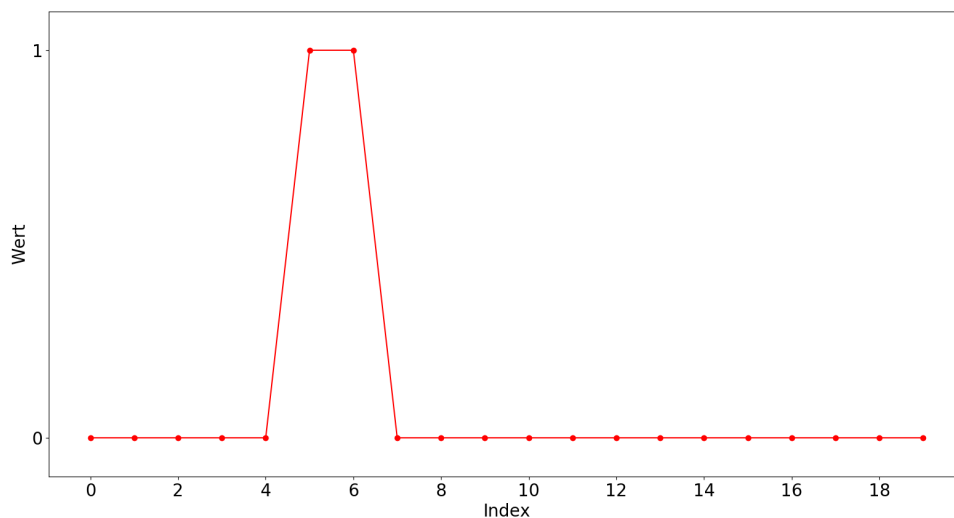


Abbildung 6: Sequenz mit einem Ausreißer

Ein weiteres Grundmuster ist in Abbildung 7 dargestellt. Dieses zeigt einen regelmäßigen Wechsel zwischen den zwei möglichen Zuständen und wird als *Zittern* bezeichnet. Dieses Verhalten ist eher untypisch für Produktionsmaschinen und dürfte nicht oft in der Realität beobachtet werden. Hier wäre ein Rückschluss auf eine defekte Sensorik naheliegend. Solche Sequenzen haben eine Balance von ca. 0.5 sowie eine hohe Frequenz.

Bei dem dritten Muster befindet sich eine Sequenz zuerst in einem Zustand und verharrt nach einem Wertewechsel im Anderen. Abbildung 8 zeigt dieses Muster. In der Praxis kann so ein Muster auftreten, wenn eine Maschine im Regelbetrieb arbeitet und auf Grundlage eines Fehlers längere Zeit ausfällt. Durch diesen *Wechsel* ergibt sich zudem der Name dieses Musters. Die Balance ist ausgeglichen und die Frequenz sehr niedrig.

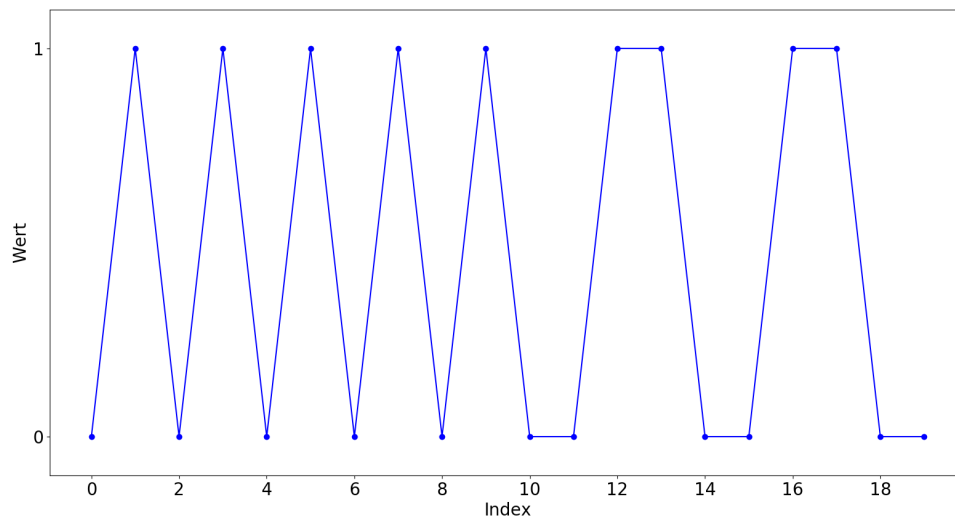


Abbildung 7: Sequenz mit einem Zittern

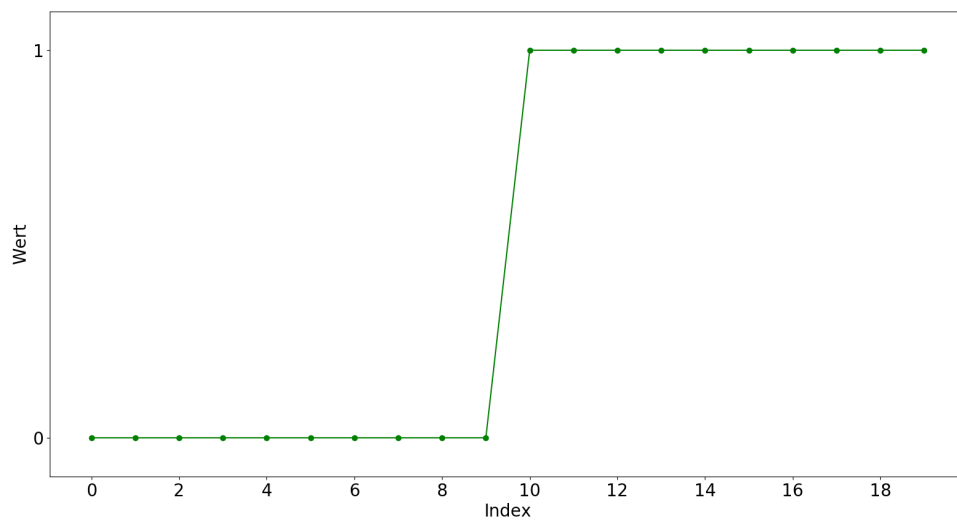


Abbildung 8: Sequenz mit einem Wechsel

Die bereits erwähnte Kreuzkorrelation findet auch Anwendung bei der Mustersuche in Sequenzen. Die Implementierung dieser Funktion und ihre Verwendung sind in Kapitel 9 dargestellt.

### 3 Wissensgenerierung - Vorgehen

In Kapitel 2 wurden Sequenzen und ihre Eigenschaften vorgestellt. Im Folgenden wird anhand eines fiktiven Beispiels das Vorgehen zur Wissensgenerierung aus Produktionsdaten - auf Grundlage der vorgenommenen Definitionen - beschrieben. Es werden die Daten von zwei Produktionsmaschinen, welche in einer Fertigungsline hintereinander angeordnet sind (vgl. Abbildung 9), miteinander verglichen.

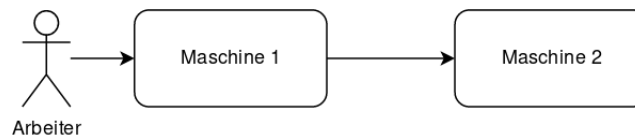


Abbildung 9: Fertigungsline mit zwei Maschinen

In diesem Beispiel wird Maschine 1 mit Stückteilen befüllt. Diese werden dort vorverarbeitet und an Maschine 2 zur Endverarbeitung weitergegeben. Eine Produktionsmaschine kann laufen, warten oder stehen. Bei der Datenerhebung wird dabei zwischen vier möglichen Zuständen unterschieden (vgl. Tabelle 3).

Symbol	Zustand	Beschreibung
0	läuft	Maschine läuft im Standardbetrieb
1	steht	Maschine wurde durch Mitarbeiter angehalten
2	steht	Maschine hat von alleine angehalten
3	wartet	Maschine wartet auf Vorgänger in Fertigungsline

Tabelle 3: Zustände in Fertigungsline



In diesem Beispiel werden die erhobenen Daten zum aktuellen Zustand der beiden Maschinen für einen Tag betrachtet. Dabei wird von 16 Arbeitsstunden und einer Datenerhebung pro Minute ausgegangen, weshalb für jede Maschine 960 Datenpunkte generiert werden (vgl. Abbildung 10).

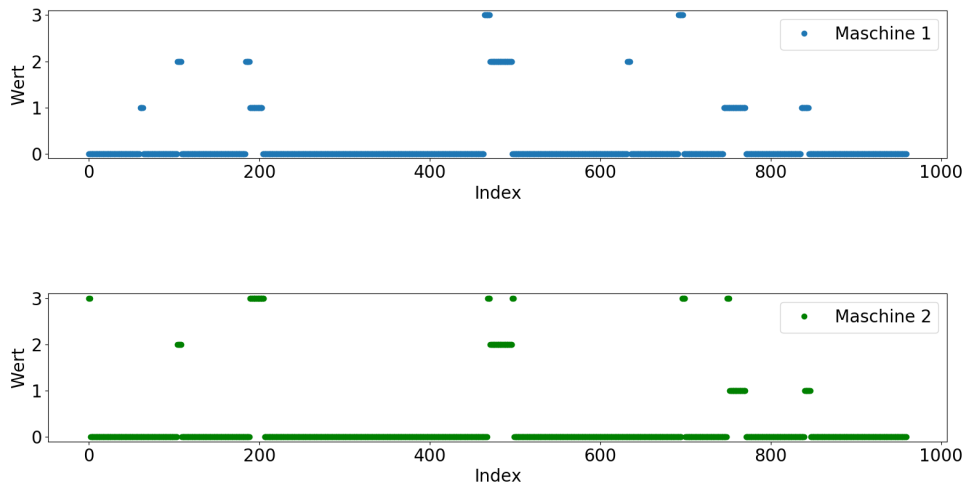


Abbildung 10: Produktionsdaten Fertigungslinie (roh)

### 3.1 Datenaufbereitung

Anders als bei simulierten Daten, muss man bei Produktionsdaten mit Fehlern rechnen. So kann es schon durch einen verdreckten Sensor zu Messfehlern oder Lücken in den Messreihen kommen<sup>7</sup>. Fehler müssen somit entfernt oder korrigiert werden. Die im Folgenden beschriebenen Schritte haben daher zur Annahme, dass eine umfassende Aufbereitung der Daten vorgenommen wurde.

Als erstes werden die Produktionsdaten auf binäre Sequenzen reduziert. Dazu müssen die Zustände (Tabelle 3) in die Kategorien *Fehlerzustand* und *kein Fehlerzustand* aufgeteilt werden. Zur ersten Kategorie gehören dabei die Zustände 1,2 und 3, da es sich bei diesen um ein unerwünschtes Verhalten im Sinne eines reibungslosen Produktionsprozesses handelt. Der einzige fehlerfreie Zustand bleibt die 0 (vgl. Tabelle 4).

<sup>7</sup> Runkler 2015

Symbol	Zustand	Beschreibung
0	kein Fehlerzustand	Maschine läuft im Standardbetrieb
1	Fehlerzustand	Maschine wartet oder wurde durch Mitarbeiter / von selbst angehalten

Tabelle 4: Reduzierte Zustände in Fertigungslinie

Bei diesem Arbeitsschritt zeigt sich, dass es bereits schwierig sein kann die ursprünglichen Zustände auf zwei zu reduzieren. In diesem Beispiel könnte der Zustand *wartet* (Tabelle 3) auch als *kein Fehlerzustand* gewertet werden, da das Warten einer Maschine an sich kein Fehlverhalten ist. Hier wird er allerdings als *Fehlerzustand* gewertet, da es in dieser Fertigungslinie Abhängigkeiten der Maschinen untereinander gibt. Daher kann bspw. das Warten von Maschine 2 auf einen Fehler von Maschine 1 zurückgeführt werden.

Wenn man das Ergebnis der Datenaufbereitung visualisiert (Abbildung 11), zeigt sich die gewünschte Vereinfachung auf zwei Zustände.

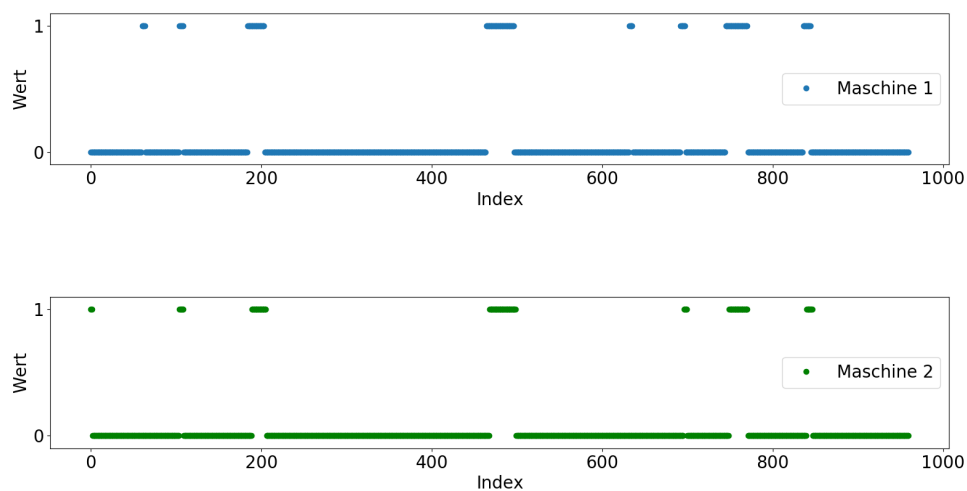


Abbildung 11: Produktionsdaten Fertigungslinie (reduziert)

## 3.2 Analyse

### Balance und Frequenz

Die Klassifizierung beginnt damit, dass die Balance ( $B(1)$ ) und Frequenz für beide Sequenzen nach 2.1 und 2.3 berechnet werden.

- $Balance_{M1} = 0.11$
- $Balance_{M2} = 0.09$
- $Frequenz_{M1} = 0.02$
- $Frequenz_{M2} = 0.01$

Auch wenn diese Werte noch keine große Aussagekraft haben, lässt sich durch sie zumindest der Eindruck einer visuellen Betrachtung (Abbildung 11) bestätigen. Die Balance ist für beide Maschinen unausgeglichen. Maschine 1 befand sich über den ganzen Tag zu 11% und Maschine 2 zu 9% der Zeit in einem fehlerhaften Zustand. Beide Maschinen sind zudem mit 0.02 und 0.01 sehr niederfrequent.

Der nächste Schritt ist eine feinere Betrachtung der bekannten Eigenschaften. Dazu werden die Balance und Frequenz für Teilsequenzen der Länge 10 berechnet. Somit lassen sich aus den 960 Datenpunkten 96 Werte für Balance und Frequenz erzielen. Die Ergebnisse sind in den Abbildungen 12 und 13 für die Balance sowie in 14 und 15 für die Frequenz dargestellt.

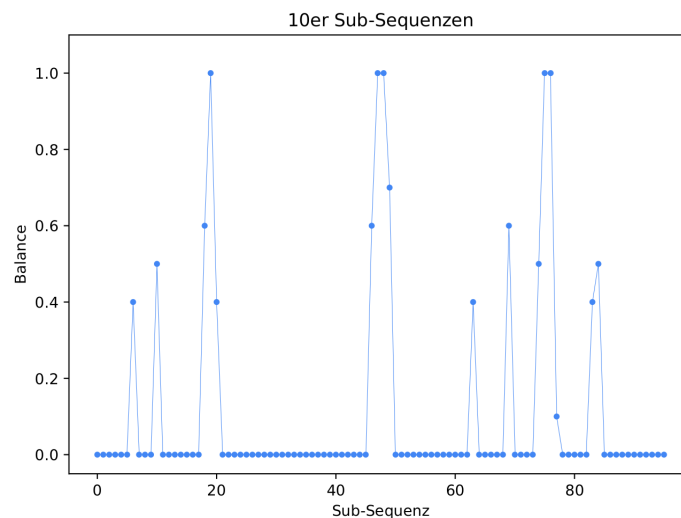


Abbildung 12: Maschine 1: Balance der 10er Subsequenzen

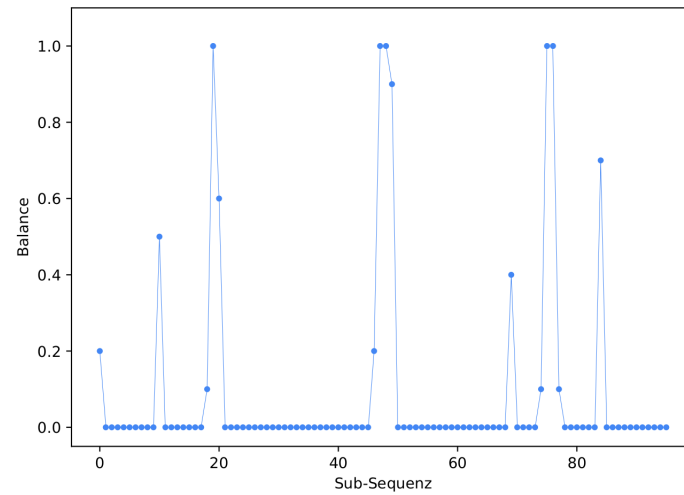


Abbildung 13: Maschine 2: Balance der 10er Subsequenzen

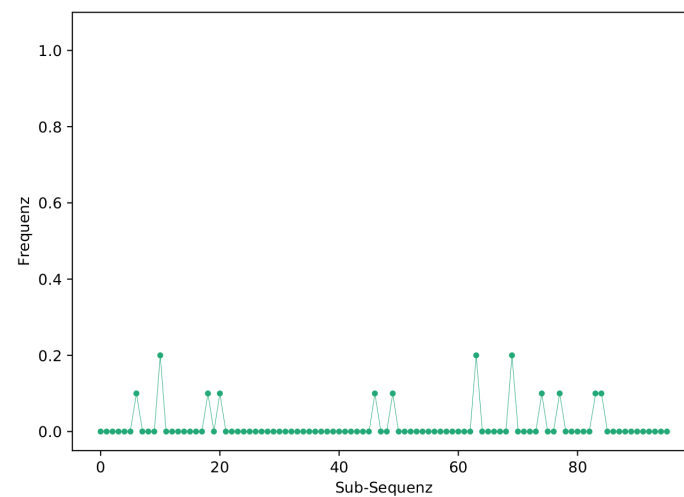


Abbildung 14: Maschine 1: Frequenz der 10er Subsequenzen

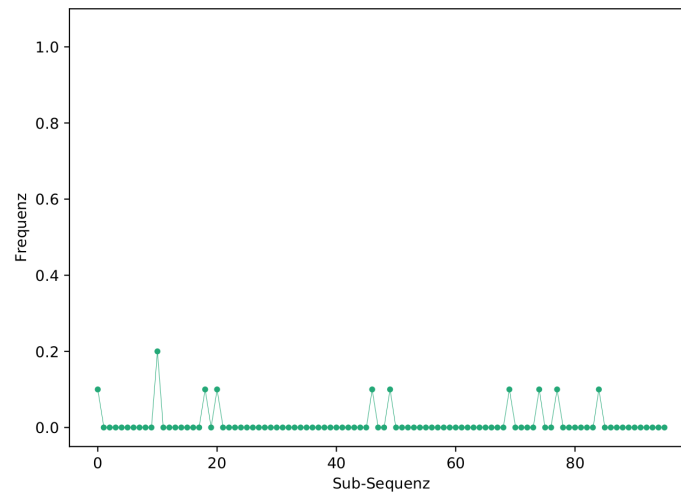


Abbildung 15: Maschine 2: Frequenz der 10er Subsequenzen

Beim Vergleichen der Balance und Frequenz der beiden Maschinen untereinander zeigen sich die Ähnlichkeiten der Plots zueinander. Dabei auffällig ist ein auftretendes Muster bei der Balance von Maschine 1 (vgl. Abbildung 12), dass beides Mal für mindestens 10 Minuten ausschließlich in einem Fehlerzustand ist. Das Muster findet sich von Index 6 bis 9 und 62 bis 77. In Abbildung 13 zeigt sich ebenfalls ein Muster von 10 bis 19 und 68 bis 77, welches sich jeweils mindestens 10 Minuten ausschließlich im Fehlerzustand befindet.

Die Frequenz beider Maschinen weist ebenfalls ein repetitives Muster auf. So ist dieses bei beiden Maschinen von 17 bis 21, 45 bis 50 und 73 bis 78 zu beobachten (vgl. Abbildungen 14 und 15).

Zusammenfassend lässt sich sagen, dass eine zeitliche Korrelation zwischen den Maschinen zu beobachten ist. Entsprechend würde sich hier die Anwendung der Kreuzkorrelationsfunktion zur Verifikation empfehlen.

## Korrelation

Mit dem Tool aus Kapitel 8 erhalten wir das Ergebnis, welches in Abbildung 16 abgebildet ist.

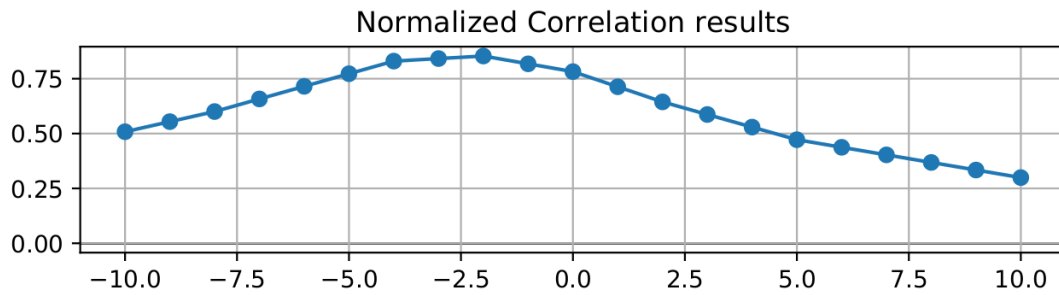


Abbildung 16: Kreuzkorrelation

Das Ergebnis der Auswertung bestätigt die erste Vermutung, dass eine zeitliche Korrelation zwischen beiden Maschinen besteht. Das Maximum von ca. 0.8 der normalisierten Kreuzkorrelation liegt ungefähr zwischen  $-2.5$  und  $-2.0$  und stellt die Verschiebung dar. Dieses Ergebnis lässt sich so deuten, dass Maschine 2 um den Verschiebungswert auf der x-Achse nach rechts verschoben ist. Das bedeutet, dass Maschine 1 meistens vor Maschine 2 in den *Fehlerzustand* springt. Somit lässt sich der erste Eindruck mit der Kreuzkorrelationsfunktion bestätigen.

## 4 Daten Bereitstellung

Wenn Daten von automatisierten Skripten untersucht werden sollen, können diese Daten unter Umständen nicht miteinander verglichen werden. Es spielt hierbei eine wichtige Rolle, dass alle Daten auf einem gleichen Stand sind, falls Zeitangaben eine Rolle spielen, sowie in einem gleichen Format gespeichert sind. Dies betrifft zum einen bereits vorhandene Daten, jedoch auch Daten, welche rein fiktiv erstellt wurden um spezielle Muster erkennen zu können. Im Folgenden werden die dazu verwendeten Methoden von Tobias Bloch genauer erklärt.

### 4.1 Datenformat

Da verschiedene Technologien während des Prozesses der Datenanalyse zum Einsatz kommen, muss ein einheitliches Format festgelegt werden, mit welchem die Daten gespeichert und weiter gegeben werden. Dabei waren die maßgeblichen Punkte, dass die Daten jederzeit von Menschen gelesen und überprüft werden können, sowie eine möglichst kompakte Speicherung.

Das Format wurde daher auf eine CSV (Comma Separated Value) festgelegt. Dabei werden die einzelnen Daten hintereinander geschrieben und lediglich durch ein Komma (,) getrennt. Wie in Listing 1 zu erkennen ist, kann auf den ersten Blick die Datenstruktur nachvollzogen werden.

Bei den zu speichernden Daten handelt es sich um Binärdaten, welche nur zwei Zustände kennen: Funktionierend (0) und Störung (1). Dabei steht jedes Zeichen für einen Intervall, welchen wir standardmäßig auf eine Sekunde definiert haben.

### Listing 1: Standard CSV mit Binär-Werten

1	1,
2	0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,
3	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,
4	0,
5	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
6	1,0,0,1,1,0,0,1,1,0,0,1,1,
7	1,0,0,1,1,0,0,1,1,0,0,1,1,1,1,1,1,0,0,1,1,0,0,1,1,0,0,1,1,1,1,1,1,1,1,

Ein Problem an dieser Form tritt auf, wenn er sich um eine große Menge an Daten in einer Datei handelt. Bei zu vielen Zeilen verliert ein Mensch schnell den Überblick. Außerdem haben große Datenmengen einen hohen Speicherbedarf auf der Festplatte und kann Größen von 100 Megabyte erreichen.

Daher wurde das Format ein wenig abgeändert. Aufgrund der beschränkten Analyse, welche sich lediglich auf Binärdaten bezieht, kommen häufig Wiederholungen vor. Als Resultat werden aufeinander folgende Intervalle mit denselben Werten in einem Eintrag gespeichert. Dabei wird zuerst der Wert und nach einem Trennstrich die Anzahl der Wiederholungen angegeben. In Listing 2 kann erkannt werden, dass selbst eine große Datenmenge von 8000 Intervallen einfach überblickt werden kann.



## Listing 2: CSV mit gruppierten Binär-Werten

```
1 118|0,1|1,215|0,1|1,58|0,1|1,219|0,1|1,95|0,1|1,168|0,1|1,181|0,1|1,67|0,
2 1|1,183|0,1|1,69|0,1|1,208|0,1|1,104|0,1|1,258|0,1|1,176|0,1|1,137|0,1|1,
3 145|0,1|1,213|0,1|1,78|0,1|1,155|0,1|1,99|0,1|1,182|0,1|1,220|0,1|1,99|0,
4 1|1,141|0,1|1,42|0,1|1,114|0,1|1,91|0,1|1,61|0,1|1,184|0,1|1,237|0,1|1,70|0,
5 1|1,89|0,1|1,245|0,1|1,211|0,1|1,77|0,1|1,194|0,1|1,150|0,1|1,257|0,1|1,62|0,
6 1|1,117|0,1|1,39|0,1|1,207|0,1|1,207|0,1|1,243|0,1|1,20|0,1|1,210|0,1|1,244|0,
7 1|1,123|0,1|1,191|0,1|1,137|0,1|1,136|0,1|1,232|0,1|1,182|0,1|1,56|0
```

Zusätzlich zu den gespeicherten Intervalldaten müssen auch allgemeine Informationen über den Inhalt der Datei gespeichert werden. Aus diesem Grund haben wir in der ersten Zeile einen „Header“ mit allen dafür benötigten Informationen eingeführt. Zudem wird der Header durch das Dollarsymbol (\$) markiert, damit es beim maschinellen Auslesen leichter erfasst werden kann und nicht mit in die Auslese von Intervalldaten geraten kann.

In Listing 3 ist der Header in einer Datei mit Maschinendaten enthalten. Die wichtigen Informationen sind, jeweils durch Komma getrennt, in vier sogenannten Key-Value-Paaren gespeichert. Das erste mit dem Namen „Machine“ gibt den Namen der Maschine an, von welcher die Intervalle stammen. „Start“ gibt den genauen Startpunkt des ersten Intervalls an, während „End“ den letzten angibt. Unsere Dateien enthalten nur Werte von einem Tag, mehr dazu in Abschnitt 4.3. Unter „Intervall“ wird der Abstand der einzelnen Intervalle in Millisekunden angegeben. Dieser Wert kann variiert werden und liegt im Standardfall bei einer Sekunde oder 1000 Millisekunden.

Listing 3: CSV mit Headereintrag

```
1 $Machine,BUP1FUE1,Start,2015-07-22-00-00-00,End,2015-07-22-24-00-00,
   Interval,1000ms$
2 60|1,3539|0,19|1,34|0,3324|1,2577|0,22|1,15|0,2824|1,168|0,1|1,181|0,1|1
3 118|0,1|1,215|0,1|1,58|0,1|1,219|0,1|1,95|0,1|1,168|0,1|1,181|0,1|1,67|0
4 1|1,183|0,1|1,69|0,1|1,208|0,1|1,104|0,1|1,258|0,1|1,176|0,1|1,137|0,1|1
5 145|0,1|1,213|0,1|1,78|0,1|1,155|0,1|1,99|0,1|1,182|0,1|1,220|0,1|1,99|0
6 1|1,141|0,1|1,42|0,1|1,114|0,1|1,91|0,1|1,61|0,1|1,184|0,1|1,237|0,1|1
7 1|1,89|0,1|1,245|0,1|1,211|0,1|1,77|0,1|1,194|0,1|1,150|0,1|1,257|0,1|1
8 1|1,117|0,1|1,39|0,1|1,207|0,1|1,207|0,1|1,243|0,1|1,20|0,1|1,210|0,1|1
9 1|1,123|0,1|1,191|0,1|1,137|0,1|1,136|0,1|1,232|0,1|1,182|0,1|1,56|0
```

Für den Fall, dass zwei oder mehrere Intervallreihen miteinander verglichen werden sollen, können mehrere Reihen in einer Datei angegeben werden. Dabei gilt, dass ein Absatz den Beginn einer neuen Reihe einleitet. Wie in Listing 4 zu erkennen, enthält eine Zeile entweder einen Header, der für die darunter stehende Zeile Informationen enthält. Die Speicherung in einer Datei hilft bei einer gezielten Untersuchung der Intervall untereinander, da auf diese Weise besser nachvollzogen werden kann, welche Werte miteinander verglichen wurden.

Listing 4: CSV mit mehreren Intervallreihen

```

1 $Machine,BUP1FUE1,Start,2015-07-22-00-00-00,End,2015-07-22-24-00-00,
   Interval,1000ms$
2 60|1,3539|0,19|1,34|0,3324|1,2577|0,22|1,15|0,2824|1
3 $Machine,BUP1FUE1,Start,2015-07-23-00-00-00,End,2015-07-22-24-00-00,
   Interval,1000ms$
4 69|1,2249|0,22|1,60|0,1165|1,4688|0,21|1,11|0,1624|1
5 $Machine,BUP1FUE1,Start,2015-07-24-00-00-00,End,2015-07-22-24-00-00,
   Interval,1000ms$
6 225|1,23|0,1745|1,634|0,2324|1,2897|0,24|1,235|0,74|1
7 $Machine,BUP1FUE1,Start,2015-07-25-00-00-00,End,2015-07-22-24-00-00,
   Interval,1000ms$
8 215|1,687|0,164|1,4765|0,3324|1,2784|0,46|1,15|0,164|1

```

Die Erstellung einer Datei mit verschiedenen Daten geschieht nach aktuellem Stand nicht automatisch, sondern muss manuell ausgeführt werden. Ein Algorithmus zur Automatischen Untersuchung könnte in Zukunft angesetzt werden, wenn ein sinnvolles Vergleichsmuster ermittelt wurde. Unsere Untersuchung hat sich auf spezielle Vergleiche bezogen, weshalb manuelles Eingreifen gerechtfertigt war.

Ein anderer Ansatz für die Speicherung der Intervallreihen hätte in einer Datenbank geschehen können. Dies hätte jedoch größeren Aufwand zur Folge und wäre problematisch beim Austausch der Daten untereinander gewesen. Da es im Umfang dieser Arbeit mit unserer Variante angemessen funktioniert hat, fanden wir den Einsatz einer Datenbank für unnötig.

## 4.2 Daten Generierung

Für die Analyse von Datenintervallen werden Testdaten benötigt, welche auf die verschiedenen Muster untersucht werden können. Aus diesem Zusammenhang wurde im Rahmen dieser Arbeit eine Anwendung programmiert, welche je nach Parameterangaben verschiedene Testdateien generieren kann. Dabei gilt, dass 0 der Normalzustand ist und 1 einen Problemzustand darstellt.

Bei dieser Anwendung handelt es sich um eine Konsolenapplikation, welche in C# mit dem .NET Core Framework geschrieben wurde. Durch die Verwendung von .NET Core wird sichergestellt, dass das Programm neben Windows auch auf Mac und Linux mit vollem Umfang lauffähig ist.

Der Benutzer gibt der Anwendung seine Einstellungen entweder über Kommandozeilenparameter (siehe Abbildung 18) mit oder über die Konsolenanwendung, welche nach den nicht angegebenen Parametern fragt. Das Ergebnis wird in einer CSV-Datei in unserem Format (siehe Abschnitt 4.1) abgespeichert. Der Ablaufprozess wird in Abbildung 17 verdeutlicht.

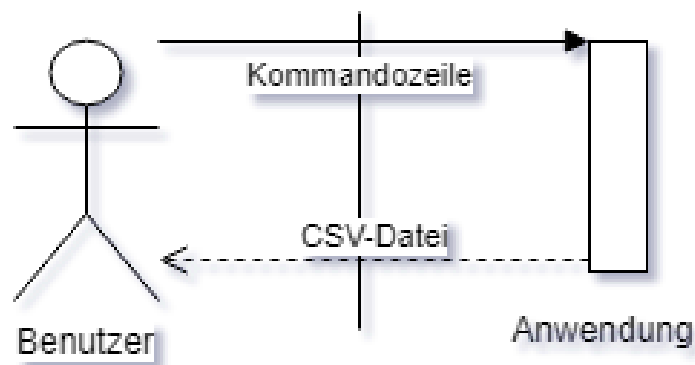


Abbildung 17: Use-Case-Diagramm der Generationsanwendung

Open:

Abbildung 18: Aufruf des Programmes mit Kommandozeilenparameter

Als Parameter für die Anwendung gibt es verschieden Optionen, welche das Programm zum Start benötigt. Dabei handelt es sich um folgende:

**IntervalsMaximum**

Integer, gibt die Anzahl der Intervalle an, welche generiert werden. Der Standardwert liegt hier bei 10000.

**IsHighfrequent**

Nullable Boolean, kann die Generation beeinflussen, ob eher hochfrequente oder niederfrequente Sequenzen herauskommen. Kann auf null gesetzt werden um die Generation dem Zufall zu überlassen.

**UsePattern**

Boolean, gibt an, ob bei der Generation willkürlich generiert werden soll oder bestimmte Muster („Patterns“) genutzt werden sollen.

**UseComplexRandom**

Boolean, der die Nutzung einer komplexeren Zufallsmethode angibt. Dies führt jedoch zu längeren Berechnungszeiten, weshalb er standardmäßig aus ist.

**UseShortSaveForm**

Boolean, gibt an, ob die Kurzform des Speichersystemes genutzt werden soll (wie in Abschnitt 4.1 beschrieben)

### 4.2.1 Algorithmus zur Datengenerierung

Für die eigentliche Generierung der Daten wird ein Algorithmus verwendet, welcher im Zweck dieser Arbeit entstanden ist. Zuerst werden die mitgegebenen Parameter ausgelesen und analysiert. Dadurch können Optionen gesetzt werden, wie die maximale Länge eines Pattern (dt. Muster), die Häufigkeit von Patterns in der Sequenz und die möglichen Patterns.

Ein Pattern ist die Abfolge bestimmter 0 und 1 Kombinationen, welche wiederholt auftreten. Zwischen diesen Patterns befinden sich eine große Ansammlung von 0, da angenommen wird, dass die theoretische Maschine in diesen Zeiträumen ordnungsgemäß funktioniert. Durch den Parameter „IsHighfrequent“ werden die Parameter auf die Frequenz beeinflusst, wodurch die Chance höher oder tiefer ist, dass entsprechende Frequenzen vorliegen.

Diese Patterns wurden im Zuge der Studienarbeit erstellt, damit eine Ähnlichkeit zwischen den generierten Intervallen und den maschinellen Daten besteht. Ein Pattern stellt hierbei eine gewisse Art von Problem dar, welche bei einer realen Maschine hätte auftreten können. Diese helfen zudem, dass in den computergenerierten Daten eine Form der Korrelation festgestellt werden kann, was bei zufällig generierten Daten nicht unbedingt gegeben wäre.

Grundlegend geben die Patterns die Intervallmöglichkeiten an, welche bereits im vorangegangenen Kapitel erklärt wurden. Im Folgenden ist eine Liste aller möglichen Patterns:

**Peak**

Einzelner Ausschlag von 1-5 Fehlerzuständen. Hierbei kann es sich um einen Messfehler oder ein kleines Problem handeln, welches sich sofort selbst behebt.

**Block**

Langer Ausschlag von 20-500 Fehlerzuständen. Symbolisiert ein größeres Problem, bei dem die Maschine stehen geblieben ist und nicht mehr funktioniert.

**Shiver**

Schneller Wechsel von Funktionsfähigkeit und Fehlerzustand in 1-3er Abständen. Zeigt ein Problem, bei dem die Maschine mit Behinderung läuft.

**Long Shiver**

Langsamer Wechsel von Funktionsfähigkeit und Fehlerzustand in 5-20er Abständen. Ähnlich dem normalen Shiver, aber mit mehr Abstand um die Variation zu erhöhen.

**Random**

Zufallsgenerierte Ansammlung von 0 und 1. Stellt eine nicht sofort erklärbare Situation dar, welche mit geringer Wahrscheinlichkeit jederzeit auftreten kann.

Der grundsätzliche Algorithmus funktioniert in drei Einzelabschnitten. Zuerst wird die Anzahl der Patterns ermittelt, welche für die maximale Intervalllänge angemessen sind. Die genaue Zahl ist zufällig, wird allerdings von der Option „IsHighfrequent“ erhöht, falls diese gesetzt wurde. Durch die zufällige Anzahl an Patterns ergibt sich eine größere Vielfalt an unterschiedlichen Intervallen.

Sobald die Anzahl der Patterns klar ist, wird deren Länge und Abstand geklärt. Die Länge gibt die Anzahl an Werten an, die ein Intervall beinhaltet, während der Abstand die Anzahl von 0 zwischen zwei Patterns angibt. Diese Anzahlen werden ebenfalls zufällig generiert um Varianz zu schaffen.

Der letzte Schritt ist das Zuordnen der verschiedenen Patterns auf die entstandenen Patternplätze. Die Patterns werden zufällig ausgelost, allerdings mit unterschiedlicher Wahrscheinlichkeit. Dabei wird vor allem auf die Option „IsHighfrequent“ Rücksicht genommen, wodurch viele Shiver und wenige Blöcke oder Peaks ausgewählt werden.

Im Anschluss werden die Patterns mit entsprechender Länge und Abstand eingefüllt und das Intervall wird vollständig gebildet. Falls „UseShortSaveForm“ gesetzt wurde, wird das Intervall auf die kurze Schreibweise aus Abschnitt 4.1 umgewandelt. Das Ergebnis wird in einer CSV-Datei gespeichert.

## 4.3 Daten Transformation

Nach der Untersuchung von simulierten Daten wurde auf reale Daten übergegangen. Dafür lagen uns Maschinendaten der Firma MPDV vor, welche über den Zeitraum 2016-2018 gesammelt wurden.

Wenn neben den simulierten Daten auch reale Daten untersucht werden sollen, dann müssen diese auf einen gemeinsamen Nenner gebracht werden, sodass die selben Methoden zur Untersuchung verwendet werden können. Daher lag der Entschluss nahe, die Maschinendaten der MPDV auf unser Format umzuwandeln.

Die Rohdaten wurden uns als CSV-Dateien gegeben, welche die Störmeldungen mehrerer Maschinen über mehrere Tage beinhalten. Als Ansatz wurde gewählt, ein Programm zu schreiben, welches die Rohdaten in unser Format umwandelt, wie es in Abschnitt 4.1 beschrieben wurde. Hierfür wurde ebenfalls .NET Core verwendet, damit die Anwendung auf verschiedenen Betriebssystemen zum Einsatz kommen kann.

Bei einer Untersuchung der Rohdaten wurde klar, dass diese sehr viele Datensätze enthielt und relativ unübersichtlich aufgebaut war. Die Störmeldungen wurden einzeln aufgeführt, bei denen jeweils Start und Endzeit, sowie die entsprechende Meldung angegeben wurden. Die Zeiten sind jeweils in Sekunden angegeben und auf den gesamten Tag ausgelegt. Ein Ausschnitt wird in Listing 5 gezeigt. Bei dem jeweils ersten Ausdruck handelt es sich um den Maschinennamen, danach folgt Startzeit, Datum, Endzeit sowie die Störmeldung als Zahlen-code.

Listing 5: CSV mit Rohdaten

```

1 "BFD1ERF1","16745","08/09/2013","16753","105"
2 "BFD1FUE1","16753","08/09/2013","16773","2"
3 "BFD1FUE1","16773","08/09/2013","16773","9011"
4 "BFD1ERF1","16753","08/09/2013","16773","2"
5 "BFD1AR01","10719","08/09/2013","10932","2"
6 "BFD1AR01","10932","08/09/2013","10934","105"
7 "BFD1FUE1","12813","08/09/2013","12910","105"
8 "BFD1ERF1","12813","08/09/2013","12910","105"
9 "BFD1FUE1","13890","08/09/2013","14181","7"
10 "BFD1ERF1","13890","08/09/2013","14181","7"

```



Die MPDV Dateien waren mit ihrer Größe bis zu 20 Megabyte relativ voll gepackt und mussten reduziert werden. Der Ansatz war, für jede Maschine und jeweils jeden Tag eine eigene Datei zu erstellen. Dies sollte die Möglichkeit verbessern gezielt dieselbe Maschine an unterschiedlichen Tagen oder verschiedene Maschinen miteinander zu vergleichen.

Die Konsolenanwendung zur Transformation ist relativ simpel aufgebaut und besitzt keinerlei Einstellungsmöglichkeiten. Als einziger Input ist der Pfad zu einer MPDV-Datei erforderlich. Der Algorithmus sortiert zuerst alle Einträge, welche in den Rohdaten zu finden sind. Dabei wird die Nutzung von Objektorientierter Programmierung genutzt. Ein Klassendiagramm ist unter Abbildung 19 zu sehen.

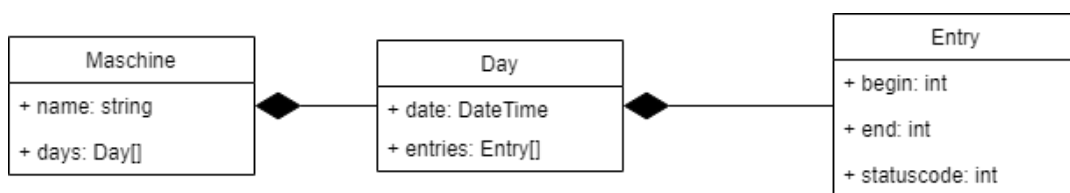


Abbildung 19: Klassendiagramm für die Datenspeicherung

Für jede neue Maschine wird ein neues Objekt angelegt mit dem jeweiligen Maschinennamen. Für jeden Tag, an dem Daten für die entsprechende Maschine vorliegen, wird ein „Day“-Objekt mit dem entsprechenden Datum der Maschine hinzugefügt. In diesem Objekt werden alle Einträge an diesem Tag gespeichert. Diese strukturierte Speicherung ermöglicht eine bessere Übersicht und verbessert die Weiterverarbeitung der Daten im nächsten Schritt.

Da in dem unter Abschnitt 4.1 definiertem Format die Statusmeldungen in Intervallen gespeichert sind, muss nun eine Transformation stattfinden. Deswegen werden für jeden Tag die Sekunden fortlaufend hochgezählt und auf vorliegende Störmeldungen überprüft. Dabei gilt, dass jede Störmeldung mit einer „1“ gekennzeichnet wird und der Normalzustand mit „0“.

Jede dieser umgewandelten Tage wird danach in einer eigenen Datei gespeichert und mit einem entsprechenden Header versehen. Der Header ist in diesem Fall wichtig, da ansonsten die Hintergrunddaten, von welcher Maschine und welchem Tag die Daten stammen, verloren gehen würden. Diese Dateien können im Anschluss ausgewertet werden.

# 5 Implementierung des Analyseprojekts

In diesem Kapitel wird das implementierte Projekt zur Analyse der Daten vorgestellt. In den folgenden Kapiteln werden die implementierte Skripte mit den angebotenen Funktionen dargestellt. Dabei wird auch auf die Bedeutung und Wirkung der Parameter eingegangen. Für die Analysen wird empfohlen, die automatisierten Skripte zu verwenden, deren Ausführung in Kapitel 10 beschrieben wird. Beispiele für das Ergebnis einer Ausführung ist im Anhang unter Kapitel 12.1 dargestellt.

## 5.1 Warum Python?

Python ist eine Skriptsprache, die auch im wissenschaftlichen Umfeld und für maschinelles Lernen verwendet wird<sup>8</sup>. Dadurch existieren im Umfeld der Skriptsprache viele Bibliotheken die eingebunden und verwendet werden können. Aufgrund der großen Nutzerbasis und Dokumentation der verwendeten Bibliotheken sind notwendige Informationen leichter zu recherchieren.

Ein weiterer Vorteil von Python kommt daher, dass es sich um eine Skriptsprache handelt, die interpretiert wird. Anpassungen können schnell umgesetzt und getestet werden, ohne die komplette Anwendung neu zu kompilieren und zu veröffentlichen. Dadurch ist Python allerdings auch weniger für schnelle Berechnungen geeignet. Deshalb wurde für numerische Berechnungen die Bibliothek NumPy entwickelt, die Berechnungen erheblich schneller ausführen kann<sup>9</sup>. Im Bereich der Grafikgenerierung bietet sich das Paket Matplotlib an, da es das meistverbreitete und eines der umfangreichsten Paket zur Erstellung von Plots ist<sup>10</sup>.

---

<sup>8</sup> Siehe: Fuxjäger 2017.

<sup>9</sup> vgl. Woyand 2017, S.173f.

<sup>10</sup> vgl. Rossant 2013, S.45.

## 5.2 Verwendete Bibliotheken

Zwei externe Bibliotheken werden für das Projekt benötigt:

### NumPy

Eine Bibliothek für mathematische Operationen auf multi-dimensionalen Arrays<sup>11</sup>. Aus dieser Bibliothek werden mathematische Funktionen verwendet.

### Matplotlib

Diese Bibliothek bietet Plotting-Funktionalitäten zur Darstellung von Grafiken<sup>12</sup>. Sie wird verwendet um PDF-Dateien aus der Analyse zu generieren.

## 5.3 Installationsanleitung

Damit die Skripte lauffähig sind, müssen sowohl Python als auch die zusätzlichen Bibliotheken installiert werden. Benötigt wird Python 3 (in der Entwicklung wurde Version 3.7.0 verwendet). Die zusätzlichen Bibliotheken können über „pip“<sup>13</sup>, das Paketverwaltungsprogramm für Python, installiert werden. Die Installation der Pakete ist nur einmalig notwendig. Folgende Befehle können zur Installation der Pakete verwendet werden.

### 5.3.1 Paketinstallation unter Windows

Unter Windows können die Pakete über das folgende Kommando (in einer Konsole) installiert werden (vorausgesetzt, Python 3 ist installiert. Pip sollte bei der Installation von Python bereits installiert werden):

```
1 pip install numpy matplotlib
```

Listing 6: Kommando zur Installation der notwendigen Pakete unter Windows

---

<sup>11</sup> Siehe: Athanasias 2014.

<sup>12</sup> Siehe: team 2019.

<sup>13</sup> Siehe: Foundation 2019.

### 5.3.2 Paketinstallation unter Linux / MacOS

Unter Linux/MacOS kann es notwendig werden, die Pakete explizit für Python 3 zu installieren (nach der Installation von Python 3). Im Folgenden sind die dafür notwendigen Kommandos abgebildet.

```
1 pip install numpy matplotlib
```

Listing 7: Kommando zur Installation der notwendigen Pakete unter Linux/MacOS

Hinweis: Je nach Konfiguration des Systems kann auch folgendes Kommando notwendig sein, um die Pakete explizit unter Python 3 zu installieren:

```
1 pip3 install numpy matplotlib
```

Listing 8: Kommando zur Installation der notwendigen Pakete unter Linux (Verwendung von pip3)

## 5.4 Verwendung von Docker

Es besteht die Möglichkeit, die Skripte auch mit Hilfe von Docker-Container auszuführen. Die Verwendung von Docker hat Vor- und Nachteile, die allerdings nicht Bestandteil dieser Arbeit sind. Durch die Verwendung von Docker ist die lokale Installation von Python und den benötigten Bibliotheken nicht mehr notwendig. Allerdings ist dafür eine lauffähige Installation von Docker notwendig.

Für die Verwendung von Docker existieren Beispiele im Anhang unter Kapitel 12.3.

## 5.5 Aufbau des Python Projekts

Der Quellcode ist nach der jeweiligen Funktionalität in eigene Module aufgeteilt. Dadurch ist es auch möglich, nur einzelne Funktionalitäten in eigenen Skripten zu verwenden. Die Module selbst sind wiederum in einzelne Dateien gegliedert. Dadurch werden die Aufgaben und Funktionalitäten noch weiter aufgeteilt, um die Lesbarkeit des Codes zu erhöhen und die Komplexität zu reduzieren. Zusätzlich zur Dokumentation in dieser Arbeit, wurde der Code kommentiert.

### 5.5.1 Top-Level Ansicht

Im Oberverzeichnis befinden sich die folgenden Dateien und Ordner. Dieses Verzeichnis dient als Einstiegspunkt zur Ausführung der Skripte.

#### Skriptdateien

Der Einstiegspunkt zur automatisierten Ausführung bildet das Skript in „automated\_script.py“. Zum Zweck der Dokumentation sind weitere Beispielskripte abgelegt, die das Prefix „example“ im Dateinamen tragen.

#### Unterordner

In den Unterordnern „analysisrequest“, „categorization“ und „crosscorrelation“ sind die Skripte für den jeweiligen Bereich abgelegt. Diese werden in den folgenden Sektionen genauer beschrieben. Eine Ausnahme bildet der „sourceFiles“ Ordner. Dieser Ordner wird für die Quelldateien verwendet, wenn bei der Ausführung kein alternativer Pfad angegeben wird. Sollte der Ordner nicht existieren, so wird dieser automatisch erstellt.

#### Docker-File

Diese Datei kann verwendet werden, um einen Docker-Container zu erstellen. Siehe dazu die notwendigen Befehle im Anhang unter 12.3.

#### Dependencies.txt

In dieser Datei ist der Befehl zur Installation der notwendigen Pakete hinterlegt.

### 5.5.2 Analysis-Request

Das Modul „AnalysisRequest“ bildet die Grundlage der Analysen. Für jede Quelldatei wird ein Request angelegt, der anschließend durch die weiteren Module verarbeitet wird. Die Ergebnisse der jeweiligen Module werden ebenfalls im Request festgehalten. Dieses Modul beinhaltet auch die Logik und Funktionalität zum Einlesen der Quelldateien, sowie zur Expansion der komprimierten Quelldaten.

## Validierung des Datenformats

Eingelesen werden können Dateien, die dem in Kapitel 4 beschriebenen komprimierten Datenformat entsprechen. Dabei werden die angegebenen Werte überprüft. Das Format schreibt vor, dass ein einzelner Eintrag die Anzahl und den Wert der Sequenz für diesen Block hat. Wenn die Anzahl nicht größer 0 ist, wird eine entsprechende Fehlermeldung ausgegeben. Auch der Wert wird dahingehend überprüft, dass dieser nur 0 oder 1 sein kann. Angaben die keine Zahlen sind werden nicht akzeptiert.

### 5.5.3 Categorization

Dieses Modul bietet die Funktionalität zur Kategorisierung von Sequenzen. Die Berechnung der Balance, Frequenz und der restlichen Sequenzwerte wird durch dieses Modul durchgeführt. Die Logik zur Erstellung der PDF-Dateien um die berechneten Werte zu plotten, befindet sich ebenfalls in diesem Modul. Die Funktionalität ist in Kapitel 6 näher beschrieben.

### 5.5.4 Cross-Correlation

Dieses Modul dient der Berechnung der Kreuzkorrelation sowie der Suche nach Mustern in einer Sequenz. Dafür werden unterschiedliche Funktionen sowie Einstellungsmöglichkeiten geboten, die dem Anwendungsfall angepasst werden können. Die Logik zur Erstellung von PDF-Dateien um die in diesem Modul gewonnen Daten darzustellen, ist ebenfalls im Modul selbst enthalten. Die Funktionalität ist in den Kapiteln 8 und 9 näher beschrieben.

# 6 Kategorisierung von Sequenzen

Im folgenden Kapitel wird die Implementierung der Skripte vorgestellt, die unterschiedliche Eigenschaften von Sequenzen berechnen und ausgeben. Die erstellten Auswertungen basieren auf den in Kapitel 3 vorgestellten Eigenschaften und Vorgehen. Die Skripte befinden sich im „categorization“ Ordner und werden von den automatisierten Skripten verwendet, deren Ausführung in Kapitel 10 beschrieben ist. Die gezeigten Bilder werden in dieser Form durch die automatisierten Skripte erstellt.

## 6.1 Kategorisierte Eigenschaften

Folgende Eigenschaften werden für Sequenzen ermittelt:

**Balance** Die Balance ist ein Maß für die „Richtung“ in die eine Sequenz ausgerichtet ist.

Damit wird der Tendenz zu 0 oder 1 ein Wert zugeordnet. Die Balance kann Werte zwischen 0 und 1 annehmen. Die beiden Grenzen sind ebenfalls im Wertebereich vorhanden. Eine Sequenz mit einer Balance von 0.5 gilt als ausgeglichen, das bedeutet, dass die Sequenz so viele 0en wie 1en enthält.

**Frequenz** Die Frequenz ist ein Maß für die Aktivität an Wertewechseln innerhalb einer Sequenz. Ein Wert gegen 0 entspricht einer sehr inaktiven Sequenz mit nur sehr wenigen bis keinen Wertewechsel. Ein Wert von 1 steht für eine hochfrequente Sequenz, in der sehr viel Wertewechsel stattfinden.

**Sub-Sequenz** Die zu analysierenden Sequenzen werden anhand ihrer Länge in kleinere Sub-Sequenzen aufgeteilt, für die ebenfalls die Balance und Frequenz berechnet werden.

### 6.1.1 Berechnung Balance

#### Vorgehen

Die Balance wird nach folgender Formel berechnet:

$$Balance = \frac{\text{Anzahl Wert} = 1}{\text{Gesamtlänge der Sequenz}}$$

Bei einer leeren Sequenz ist der Standardrückgabewert 0. Der berechnete Wert entspricht der im Abschnitt 2.4 vorgestellten Balance(1).

#### Beispiel für eine ausgeglichene Sequenz

Die folgende Sequenz mit einer Länge von 10000 Einträgen folgt der folgenden Vorschrift und ist in Abbildung 20 abgebildet:

$$a_n = \begin{cases} 1 & \text{für } n < 5000 \\ 0 & \text{für } n > 5000 \end{cases}$$

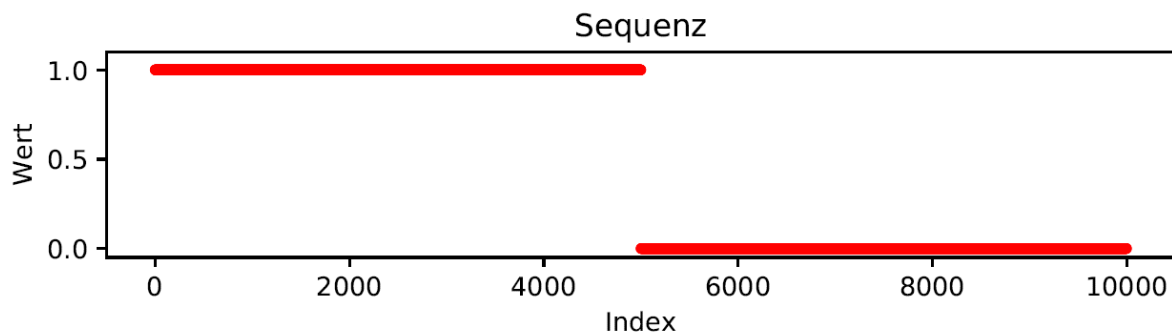


Abbildung 20: Stellt eine ausgeglichene (Balance = 0.5) Sequenz dar<sup>14</sup>.

Für diese Sequenz wird eine Balance von 0.5 berechnet, da es genau so viele Werte mit 1 als auch 0 gibt.

<sup>14</sup> Quelle: Eigene Darstellung



### Beispiel für eine unausgeglichene Sequenz

Die folgende Sequenz mit einer Länge von 10000 Einträgen folgt der folgenden Vorschrift und ist in Abbildung 21 abgebildet:

$$a_n = \begin{cases} 1 & \text{für } n < 9000 \\ 0 & \text{für } n > 1000 \end{cases}$$

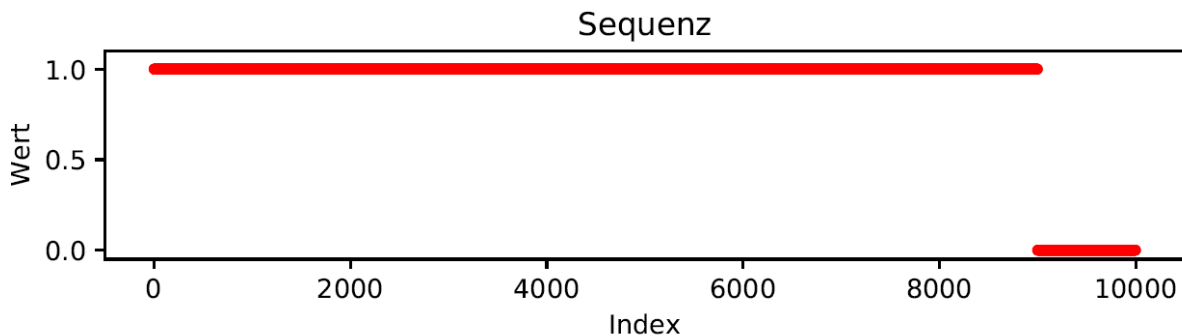


Abbildung 21: Stellt eine unausgeglichene (Balance = 0.9) Sequenz dar<sup>15</sup>.

Für diese Sequenz wird eine Balance von 0.9 berechnet, da diese deutlich Richtung 1 tendiert.

### 6.1.2 Berechnung der Frequenz

#### Vorgehen

Die Frequenz einer Sequenz wird nach folgender Formel berechnet:

$$\text{Frequenz} = \frac{\text{Anzahl der Wertänderungen}}{\text{Gesamtlänge der Sequenz}}$$

Ein Wert  $\leq 0.25$  gilt als „Niederfrequent“, ein Wert  $x > 0.25$  und  $< 0.75$  gilt als „Mittelfrequent“ und Werte  $\geq 0.75$  als „Hochfrequent“. Der Standardrückgabewert einer leeren Sequenz ist 0. Der berechnete Wert entspricht der im Abschnitt 2.4 vorgestellten Frequenz.

<sup>15</sup> Quelle: Eigene Darstellung

### Beispiel einer niederfrequenten Sequenz

Eine niederfrequente Sequenz mit einer Länge von 10000 Einträgen ist in Abbildung 22 abgebildet:

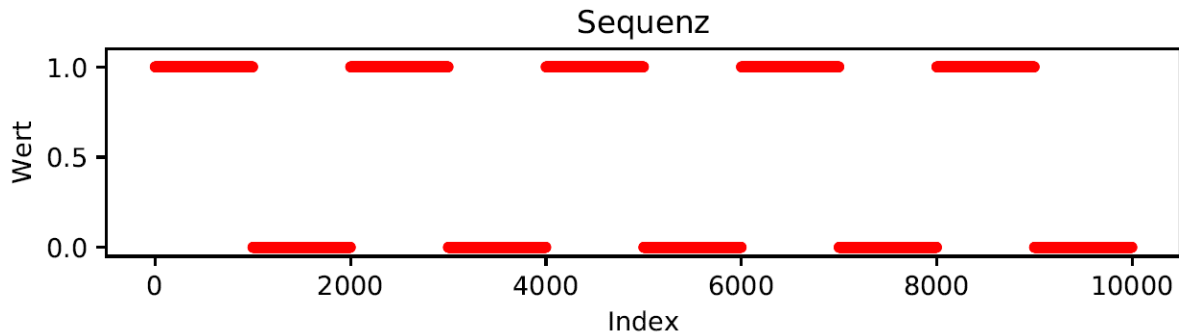


Abbildung 22: Stellt eine niederfrequente Sequenz dar<sup>16</sup>.

Diese Sequenz hat einen berechneten Frequenzwert von 0. Die Anzahl an Wertewechsel im Verhältnis zur Länge der Sequenz ist sehr niedrig.

### Beispiel einer mittelfrequenten Sequenz

Eine mittelfrequente Sequenz mit einer Länge von 10000 Einträgen ist in Abbildung 23 abgebildet:

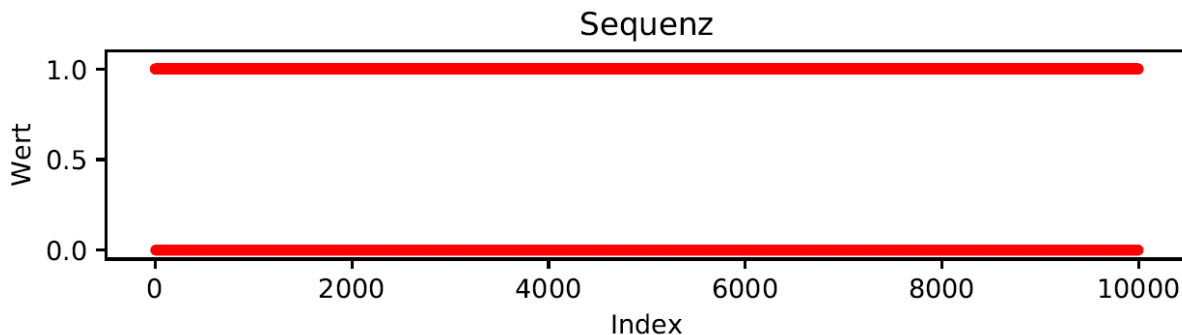


Abbildung 23: Stellt eine mittelfrequente Sequenz dar<sup>17</sup>.

Diese Sequenz hat einen berechneten Frequenzwert von 0.4 und gilt damit als mittelfrequent.

<sup>16</sup> Quelle: Eigene Darstellung

<sup>17</sup> Quelle: Eigene Darstellung

### Beispiel einer hochfrequenten Sequenz

Eine hochfrequente Sequenz mit einer Länge von 10000 Einträgen ist in Abbildung 24 abgebildet:

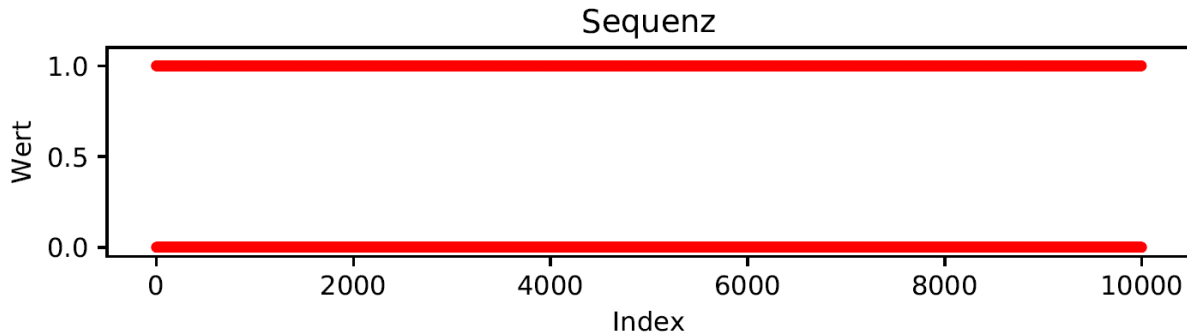


Abbildung 24: Stellt eine hochfrequente Sequenz dar<sup>18</sup>.

Diese Sequenz hat einen berechneten Frequenzwert von 0.9 und gilt damit als hochfrequent.

### 6.1.3 Auswertung der Sub-Sequenzen

Im Zuge des Projekts wurde ersichtlich, dass eine grafische Darstellung von Sequenzen in ihrer gesamten Länge nicht optimal für alle Fälle geeignet ist. Wenn man die unterschiedlichen Sequenzen aus Abbildung 23 und Abbildung 24 vergleicht, so sehen diese auf den ersten Blick identisch aus. Bei der Auswertung der Frequenz sind diese aber unterschiedliche einzuordnen. Dieser Effekt entsteht durch die Länge der Sequenzen und deren Darstellung in den generierten PDF Dateien.

Mit einer Analyse der Sequenzen in ihrer Gesamtlänge gehen auch zusätzliche Informationen verloren. Die Information ob eine Sequenz im ersten Drittel niederfrequent und zum Ende hin hochfrequent wird, kann durch einen einzelnen Zahlenwert nicht abgebildet werden.

Um die Unterschiede zu verdeutlichen, ist es notwendig die Sequenzen in kleinere Teilsequenzen zu unterteilen und diese zu vergleichen.

Die Aufteilung in Sequenzen entspricht dem in Kapitel 3 vorgestellten Vorgehen.

<sup>18</sup> Quelle: Eigene Darstellung

## Vorgehen zur Berechnung der Sub-Sequenzlängen

Eine Sequenz wird je nach deren Länge in unterschiedlich große Teilsequenzen zerlegt. Diese Sub-Sequenzen werden ebenfalls im Bezug auf die Balance und Frequenz analysiert und die Ergebnisse werden in den entsprechenden PDF Dateien dargestellt. Die Länge der Sub-Sequenzen wird dabei iterativ festgelegt. Das Vorgehen ist im Listing 9 als Pseudo-Code dargestellt.

```

1 int sequenceLength = length(sequence)
2 int subSequenceLength = 10
3
4 // Ensure there are at least 10 sub sequences:
5 while (sequenceLength / subSequenceLength ) >= 10
6     SubSequenceLength is valid
7     // Increase the size of the sub sequence by a factor of 10
8     subSequenceLength = subSequenceLength * 10

```

Listing 9: Pseudo-Code zur Festlegung der Länge der Sub-Sequenzen

Die Länge der möglichen Sub-Sequenzen startet bei 10 und wird bei jeder Iteration um den Faktor 10 erhöht. Dies geschieht so lange, bis die Anzahl der entstehenden Sub-Sequenzen 10 unterschreitet.

Eine Sequenz der Länge 1000 wird dabei in Sub-Sequenzen der Länge 10 und 100 aufgeteilt. Eine Sequenz der Länge 10000 wird in Sub-Sequenzen der Länge 10, 100 und 1000 aufgeteilt.

## Benennung der Sub-Sequenzen

In den Schaubildern und den generierten PDF-Dateien ist von 10er-Sub-Sequenzen, 100er Sub-Sequenzen usw. die Rede. Die Zehnerpotenz ist dabei die Länge der Sub-Sequenz über die der Wert (Balance oder Frequenz) berechnet wurde. Der Index auf der X-Achse entspricht dem Index der Sub-Sequenz. Die erste 10er-Sub-Sequenz startet bei dem ersten Element in der Sequenz und beinhaltet die ersten 10 Elemente. Über diese Elemente wurden die Werte berechnet. Die 10er-Sub-Sequenz mit dem nächst größeren Index enthält die nächsten 10 Elemente. Die Sub-Sequenzen überschneiden sich dabei nicht.

### Beispiel für die analysierte Sub-Sequenzen

Im Folgenden wird ein Beispiel für die generierten Sub-Sequenzen und deren Darstellung gegeben. Als Grundlage dient die in Abbildung 25 dargestellte Sequenz. Diese Sequenz hat eine Länge von 10000, ist mit einem Wert von 0.34 für die Frequenz mittelfrequent und gilt mit einem Balance-Wert von 0.17 als Unausgeglichen. Man erkennt dass die Sequenz einige Intervalle mit einem Wert von 1 besitzt, wobei ein Wert von 0 klar dominiert. Die Sequenz ist in dieser Form graphisch schwer zu analysieren, wenn man beispielsweise eine Aussage über die Länge der Ausschläge oder die Balance in Teilbereichen treffen will.

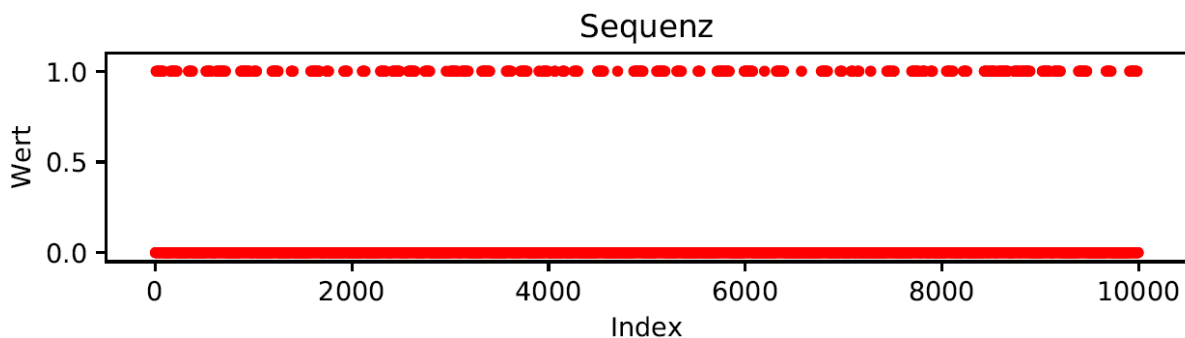


Abbildung 25: Stellt eine zu analysierende Sequenz dar<sup>19</sup>.

Stellt man allerdings die Teilsequenzen dar, erhöht sich der Detailgrad und es wird ein Verlauf der Frequenz und der Balance ersichtlich. Durch den erhöhten Detailgrad können die Sequenzen besser analysiert werden (falls dies notwendig ist).

In Abbildung 26 ist die Balance der Sub-Sequenzen dargestellt. In dieser Darstellung ist der Verlauf der Balance erkennbar, wodurch Intervalle mit gleicher Balance besser erkennbar sind. Die unterschiedliche Länge der Sub-Sequenzen wirkt sich dabei auf die Darstellung aus. Je größer die Sub-Sequenz ist, desto gleichmäßiger ist die Balance. Dadurch sind gleichmäßige Intervalle in größeren Sub-Sequenzlängen besser erkennbar als in kleinen Längen. Kurze Ausschläge sind allerdings in kurzen Sub-Sequenzlängen besser erkennbar.

<sup>19</sup> Quelle: Eigene Darstellung

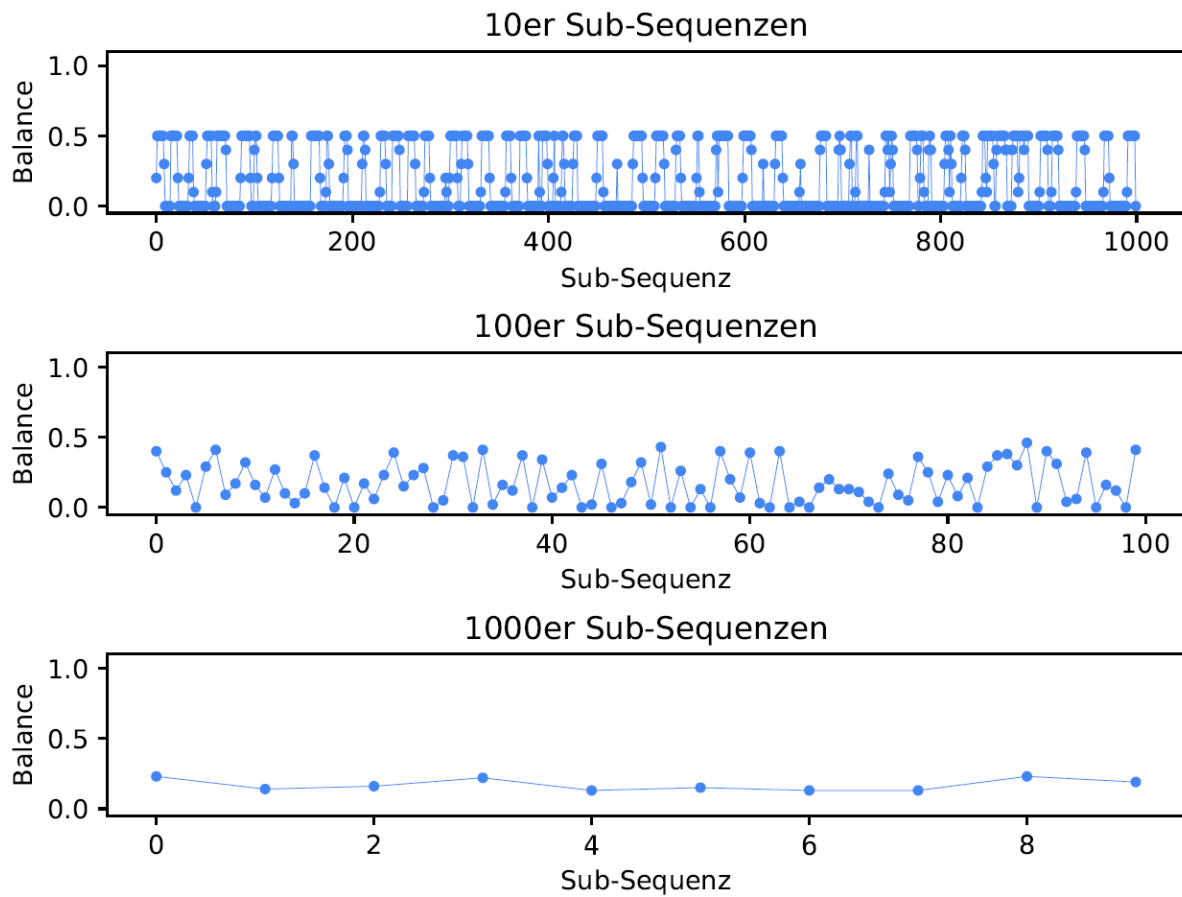


Abbildung 26: Stellt die Balance der Sub-Sequenzen dar<sup>20</sup>.

<sup>20</sup> Quelle: Eigene Darstellung

Analog dazu wird auch die Frequenz der Sub-Sequenzen analysiert. Das Ergebnis ist in Abbildung 27 dargestellt.

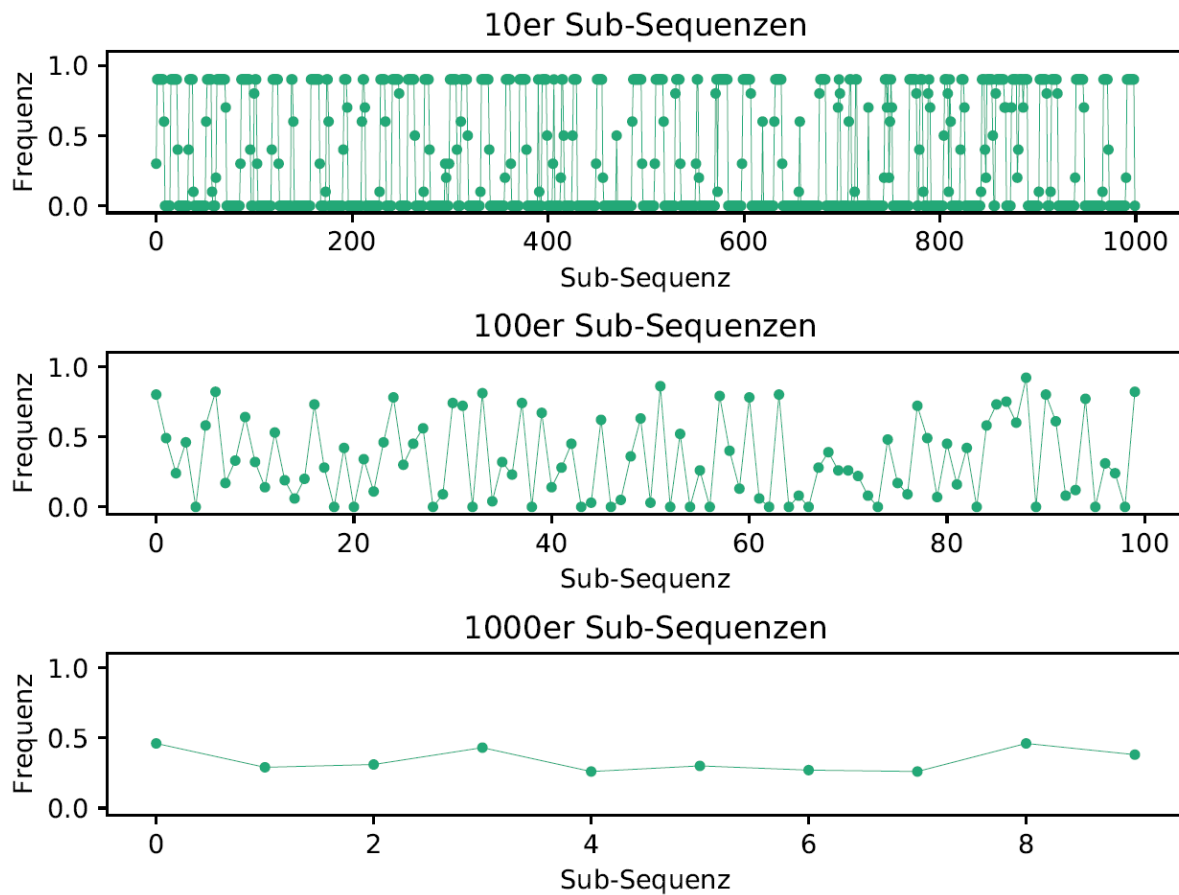


Abbildung 27: Stellt die Frequenz der Sub-Sequenzen dar<sup>21</sup>.

### Vergleich über die Sub-Sequenzen

Über die Sub-Sequenzen können die Sequenzen in manchen Fällen besser verglichen werden. Die in Abbildung 23 und Abbildung 24 dargestellten Sequenzen sehen auf den ersten Blick identisch aus. Allerdings unterscheiden sich diese in ihrem Wert für die Frequenz (0.4 zu 0.9). Dieser Unterschied ist auch deutlich, wenn man sich die Frequenz der entsprechenden Sub-Sequenzen anschaut.

<sup>21</sup> Quelle: Eigene Darstellung

Die Sub-Sequenzen der mittelfrequenten Sequenz aus Abbildung 23 sind in Abbildung 28 dargestellt. Es ist nicht überraschend, dass diese sehr konstant ist und den Frequenzwert von 0.4 wiedergibt.

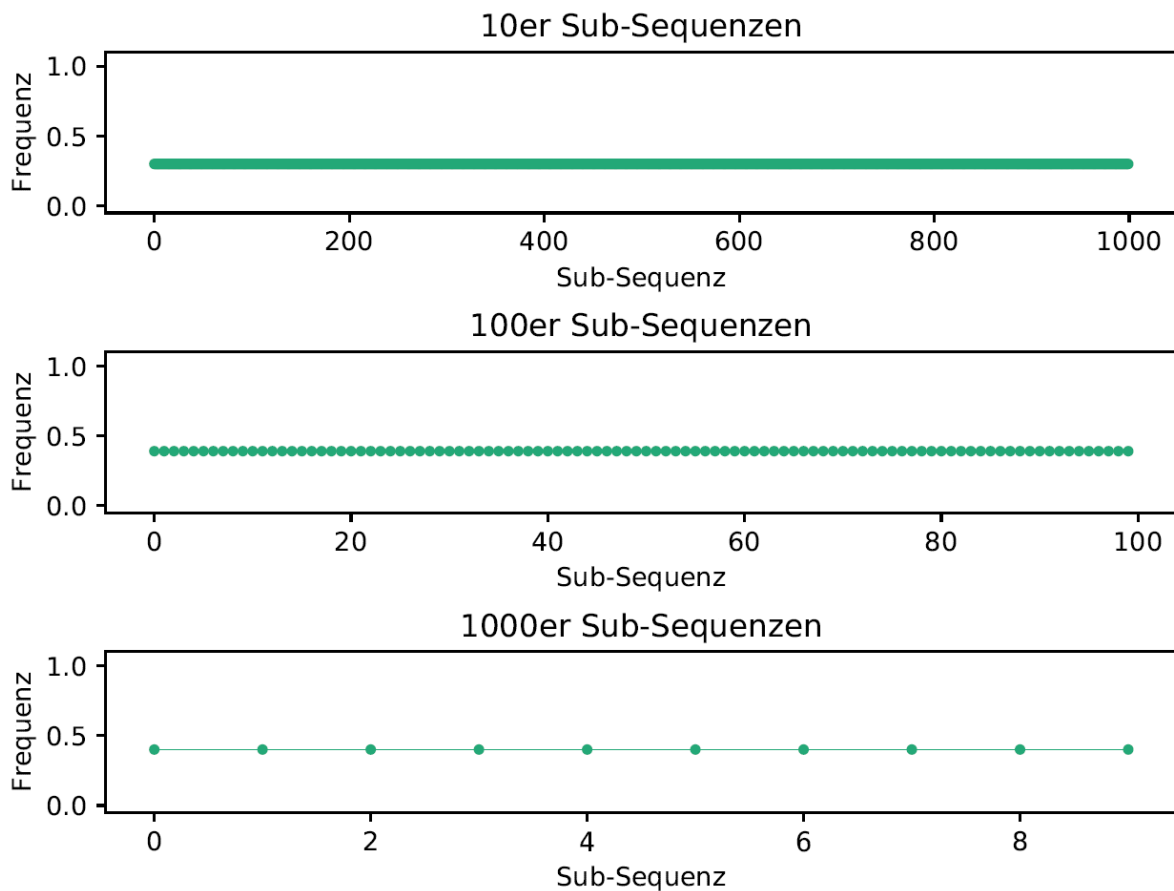


Abbildung 28: Stellt die Frequenz der Sub-Sequenzen einer mittelfrequenten Sequenz dar<sup>22</sup>.

Die Sub-Sequenzen der hochfrequenten Sequenz aus Abbildung 24 zeigen hingegen ein neues Detail, welches nicht in der Gesamtansicht erkennbar ist. Auch hier ist der hohe Wert für die Frequenz erkennbar, allerdings ist dieser nicht so gleichmäßig wie bei der mittelfrequenten Sequenz. Es gibt einzelne Spitzen, die über die gesamte Länge der Sequenz verteilt sind. Die Sub-Sequenzen sind in Abbildung 29 dargestellt.

<sup>22</sup> Quelle: Eigene Darstellung



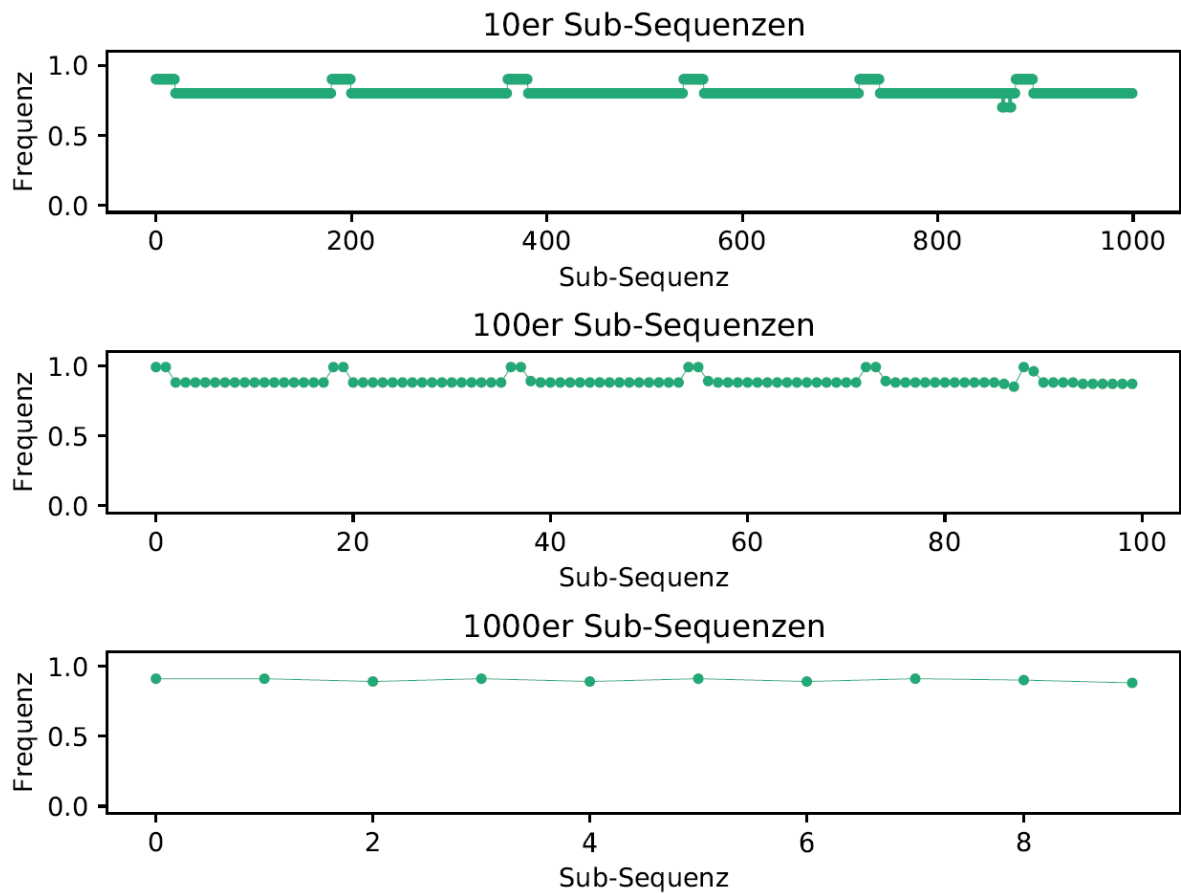


Abbildung 29: Stellt die Frequenz der Sub-Sequenzen einer hochfrequenten Sequenz dar<sup>23</sup>.

## 6.2 Beispiel-Skripte

Ein Skript welches die Funktion aus diesem Kapitel nutzt ist als Beispiel im Projekt unter „example\_script\_automated\_crosscorrelation.py“ abgelegt.

Das Skript das im Zuge der Automatisierung verwendet wird, ist im Unterverzeichnis „categorization“ unter „automated\_categorization.py“ zu finden.

<sup>23</sup> Quelle: Eigene Darstellung

# 7 Grundlagen: Kreuzkorrelation und Faltung

Dieses Kapitel beinhaltet eine kurze Einführung zur Kreuzkorrelation und der Faltung. Die Kreuzkorrelation wird im Projekt verwendet (siehe Kapitel 8), weshalb eine Einarbeitung in das Thema zum besseren Verständnis notwendig war. Für den Fall, dass das Projekt in zukünftigen Arbeiten verwendet wird, wurde dieses Kapitel hinzugefügt um den Einstieg zu erleichtern.

## 7.1 Einführung Kreuzkorrelation

Die Kreuzkorrelation wird verwendet, um die Korrelation zwischen zwei Signalen oder Sequenzen zu berechnen. Dabei werden unterschiedliche Zeitverschiebungen zwischen den Sequenzen eingesetzt. NumPy verwendet dabei folgende Formel für die Kreuzkorrelation<sup>24</sup>.

$$f \star g \equiv \int_{-\infty}^{+\infty} \bar{f}(-t) * g(t)$$

Die Formel kann in eine Form umgewandelt werden, aus der die Verschiebung einer Sequenz gegenüber einer anderen besser ersichtlich ist<sup>25</sup>:

$$f \star g \equiv \int_{-\infty}^{+\infty} \bar{f}(-\tau) * g(t + \tau) \, d\tau$$

---

<sup>24</sup> Siehe: Dokumentation NumPy: Community 2018 und Definition Kreuzkorrelation: MathWorld 2018.

<sup>25</sup> Umformung: siehe: MathWorld 2018.

NumPy bietet dabei unterschiedliche Modi bei der Berechnung der Kreuzkorrelation an. Diese entsprechen den Modi der mathematischen Faltung ( zu Englisch „Convolution“). Die Unterschiede der Kreuzkorrelation und Faltung sowie der Autokorrelation sind in den folgenden Abbildungen dargestellt. Abbildung 30 bietet einen Überblick, während die Abbildungen 31 und 32 zwei Beispiele darstellen. Über die jeweiligen Quellenangaben können diese in animierter Form betrachtet werden.

## 7.2 Beispiele Kreuzkorrelation, Faltung und Autokorrelation

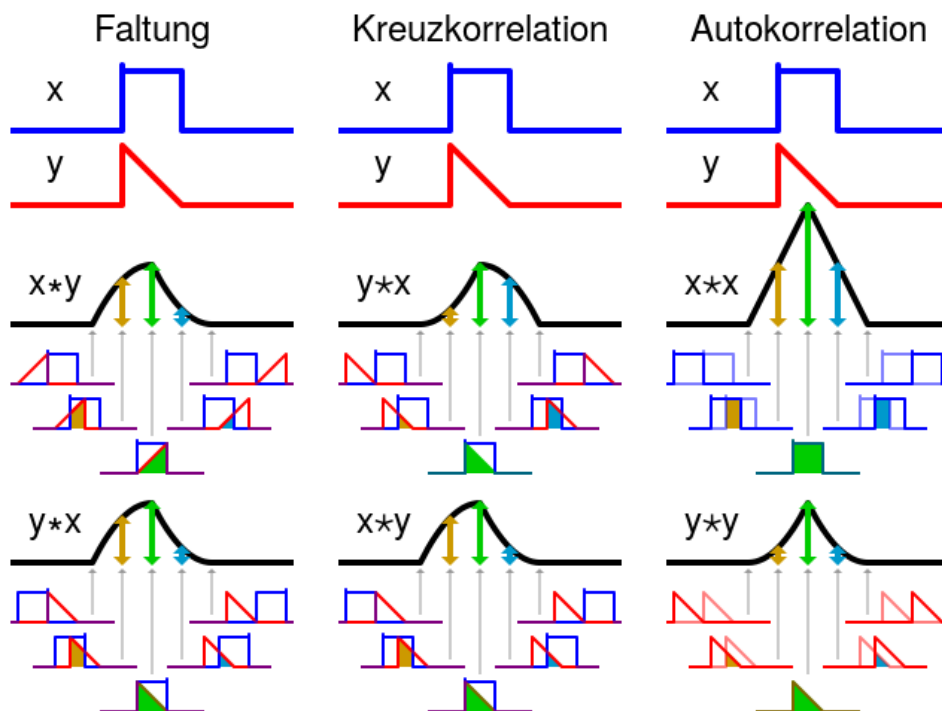


Abbildung 30: Vergleicht die Faltung, Kreuzkorrelation und Autokorrelation<sup>26</sup>

<sup>26</sup> Quelle: Cmglee 2013

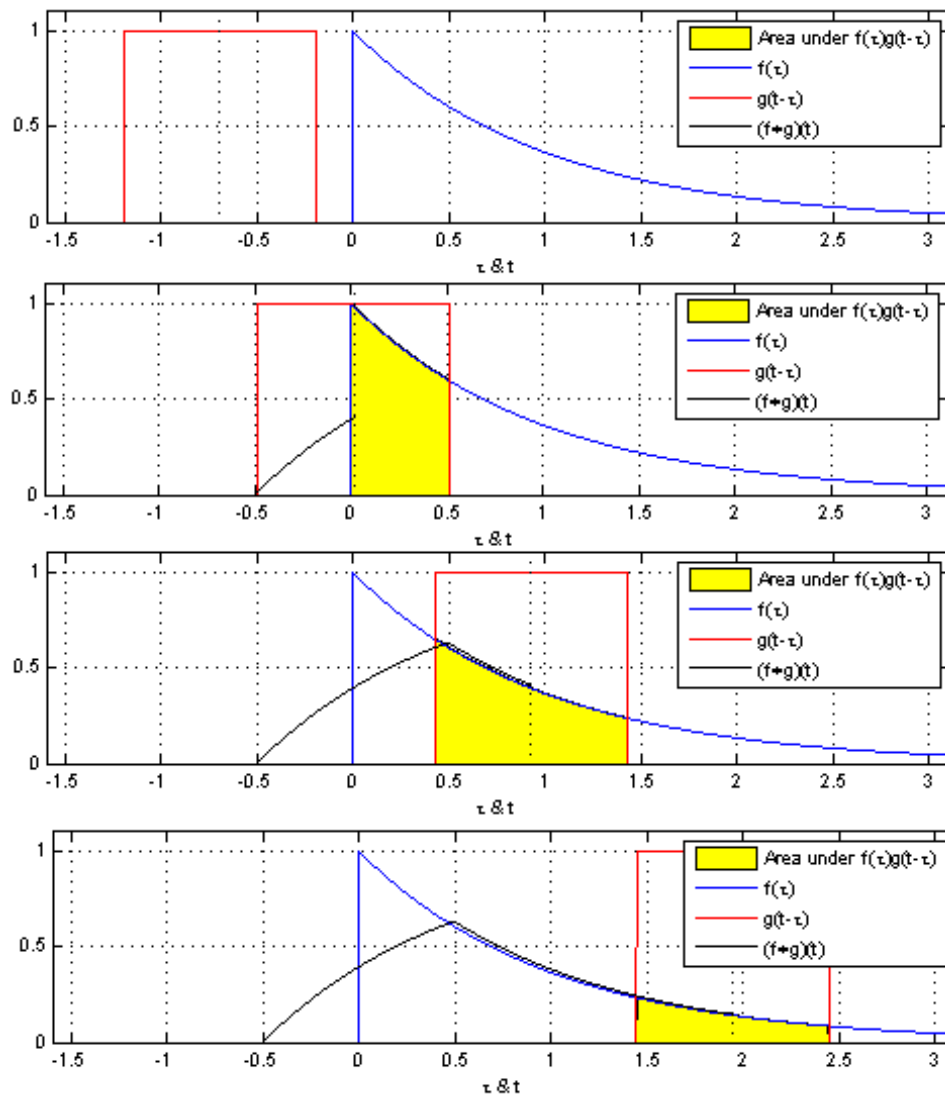
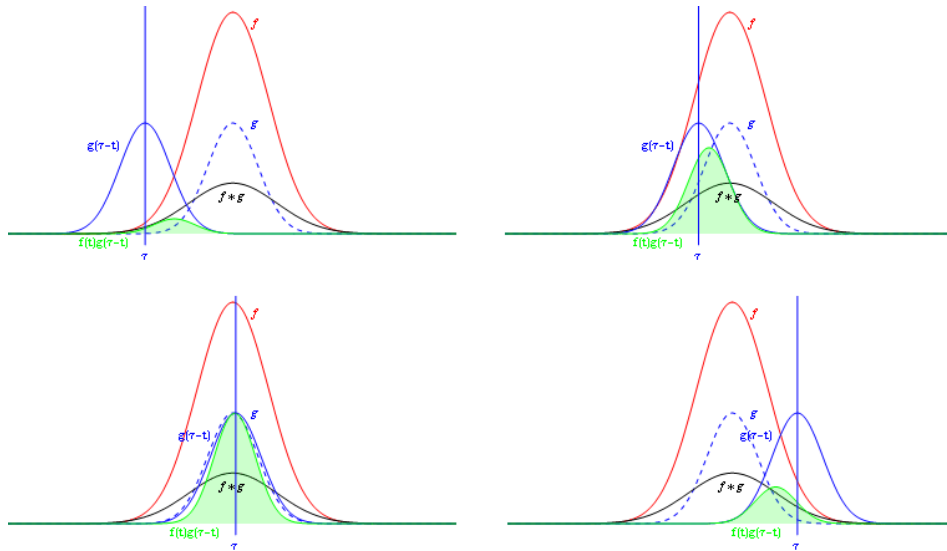


Abbildung 31: Darstellung einer eindimensionalen Faltung<sup>27</sup>

<sup>27</sup> Quelle: Amberg 2010

Abbildung 32: Faltung zweier Gaussfunktionen<sup>28</sup>

## 7.3 Bedeutung der unterschiedlichen Modi

In den Beschreibungen gelten folgende Voraussetzungen:

Sequenz A: von  $a[0]$  bis  $a[L_A - 1]$  mit  $L_A = \text{Länge von Sequenz A}$

Sequenz B: von  $b[0]$  bis  $b[L_B - 1]$  mit  $L_B = \text{Länge von Sequenz B}$

Als Quelle dient die Dokumentation<sup>29</sup> sowie eine Beschreibung der Modi<sup>30</sup>.

### 7.3.1 Modus: Valid

Dieser Modus wird verwendet, wenn der Modus nicht explizit angegeben wird.

Dabei wird eine Ausgabe der Länge  $\max(L_A, L_B) - \min(L_A, L_B) + 1$  erzeugt. Die Verschiebung findet dabei im Bereich von  $\min(L_A, L_B) - 1$  bis  $-\max(L_A, L_B) - 1$  statt.

<sup>28</sup> Quelle: Inductiveload 2009

<sup>29</sup> Dokumentation NumPy Correlate: Community 2018.

<sup>30</sup> Beschreibung der Modi: StackOverflow 2017.

### 7.3.2 Modus: Full

In diesem Modus wird die Berechnung im Bereich von 0 bis  $L_A + L_B - 2$  durchgeführt. Dadurch ergibt sich eine Ausgabe von  $L_A + L_B + 1$  Elementen. Dabei können Effekte an den Rändern der Sequenzen auftreten, da diese sich dort nicht mehr voll überlappen.

### 7.3.3 Modus: Same

Liefert ein Ergebnis der Länge  $\max(L_A, L_B)$ . Dabei können ebenfalls Effekte an den Rändern auftreten.

Die Berechnung findet im Bereich von  $\frac{(L_B-1)}{2}$  bis  $L_A - 1 + \frac{L_B-1}{2}$  statt.

Sollte  $L_A < L_B$  gelten, werden die beiden Sequenzen vor der Berechnung ausgetauscht.

# 8 Implementierte Funktion zur Berechnung der Kreuz-Korrelation

Die Skripte zur Berechnung der Kreuz-Korrelation befinden sich im „crosscorrelation“-Verzeichnis. Diese Skripte werden von den automatisierten Skripten verwendet. Ein Aufrufen des hier beschriebenen Codes ist nur notwendig, wenn man eigene Skripte verfasst oder nur eine einzelne Funktion nutzen will. Das Python Skript „functions\_crosscorrelation.py“ bietet als Einstiegspunkt die Funktion „crossCorrelation(...)“ an. Im Folgenden werden die Parameter und deren Bedeutung beschrieben. Im Anschluss werden die Auswirkungen der Parameter dargestellt. Die in diesem Kapitel dargestellten Abbildungen werden durch die Skripte erzeugt.

## 8.1 Beschreibung der Parameter

Es gibt drei Parameter, die beim Aufruf der Funktion mitgegeben werden müssen. Dabei handelt es sich um die beiden Sequenzen, die verarbeitet werden sollen sowie um die Einstellungen. Aufgrund der vielen Einstellungsmöglichkeiten, wurden diese in eine eigene Klasse ausgelagert, die beim Aufruf als dritter Parameter übergeben werden muss.

### **seqA**

Die erste Datensequenz in Form eines eindimensionalen Arrays. Die enthaltenen Zahlenwerte müssen in Float konvertierbar sein.

### **seqB**

Die zweite Datensequenz in Form eines eindimensionalen Arrays. Die enthaltenen Zahlenwerte müssen in Float konvertierbar sein.

### **settings**

Die Einstellungen die verwendet werden sollen.

### 8.1.1 Einstellungsparameter

Folgende Parameter stehen für die Konfiguration zur Verfügung. Die standardmäßig gesetzten Werte sind mit angegeben:

#### **plotNormalizedData**

[Default: False] Auf True setzen, wenn die beiden Sequenzen auch normalisiert dargestellt werden sollen.

#### **plotCorrelations**

[Default: False] Auf True setzen, wenn die berechnete Kreuz-Korrelation dargestellt werden soll. Dabei werden alle drei Berechnungsmethoden „Valid“, „Same“ und „Full“ dargestellt.

#### **plotNonNormalizedResults**

[Default: False] Auf True setzen, wenn auch die unnormalisierten Ergebnisse dargestellt werden sollen.

#### **plotNormalizedResults**

[Default: True] Auf False setzen, wenn auch die normalisierten Ergebnisse nicht dargestellt werden sollen.

#### **subtractMeanFromResult**

[Default: True] Auf False setzen, wenn das arithmetische Mittel nicht von den Ergebnissen abgezogen werden soll.

#### **drawResults**

[Default: False] Wird dieser Parameter auf True gesetzt, werden die Ergebnisse auf dem Bildschirm ausgegeben. Dies kann sinnvoll sein, wenn keine PDF-Dateien erstellt werden sollen, die Ergebnisse aber dennoch sichtbar sein sollten. Der Standardwert ist auf False gesetzt, da die Skripte automatisiert durch weitere Skripte aufgerufen werden. Die Verarbeitung der Skripte blockiert dabei solange das Fenster zur Anzeige geöffnet ist. In einem automatisierten Kontext würde niemand das Fenster schließen, wodurch der Prozess blockiert werden würde.

#### **exportToPdf**

[Default: False] Auf True setzen, wenn eine PDF-Datei mit den Ergebnissen generiert werden soll.



**exportFilePath**

[Default: „“] Der Pfad zur PDF-Datei, in dem die Ergebnisse gespeichert werden sollen. Wenn „exportToPdf“ auf True gesetzt wird, muss mit dieser Einstellung der Pfad zur PDF-Datei angegeben werden. Ansonsten wird die Einstellung ignoriert.

## 8.2 Beispiel Ergebnisse

Es gilt für alle Beispiele (solange nicht im Beispiel genauer spezifiziert):

Vorbedingung: Zwei Sequenzen mit je 10000 Einträgen.

SeqA:

$$a_n = \begin{cases} 1 & \text{wenn } n \text{ enthalten in } [1005, 2005, 1505, 3505, 5505, 6005, 6505, 8005, 8505, 9005] \\ 0 & \text{sonst} \end{cases}$$

SeqB:

$$b_n = \begin{cases} 1 & \text{wenn } n \text{ enthalten in } [1000, 2000, 1500, 3500, 5500, 6000, 6500, 8000, 8500, 9000] \\ 0 & \text{sonst} \end{cases}$$

Es ist ersichtlich, dass die Sequenzen zueinander um einen Index von 5 verschoben sind.

### 8.2.1 Anzeige der beiden Sequenzen mit normalisiertem Ergebnis

Der Aufruf der Funktion mit den Standard-Parametern der Einstellungen, liefert für die Beispielsequenzen folgendes Ergebnis (dargestellt in Abbildung 33):

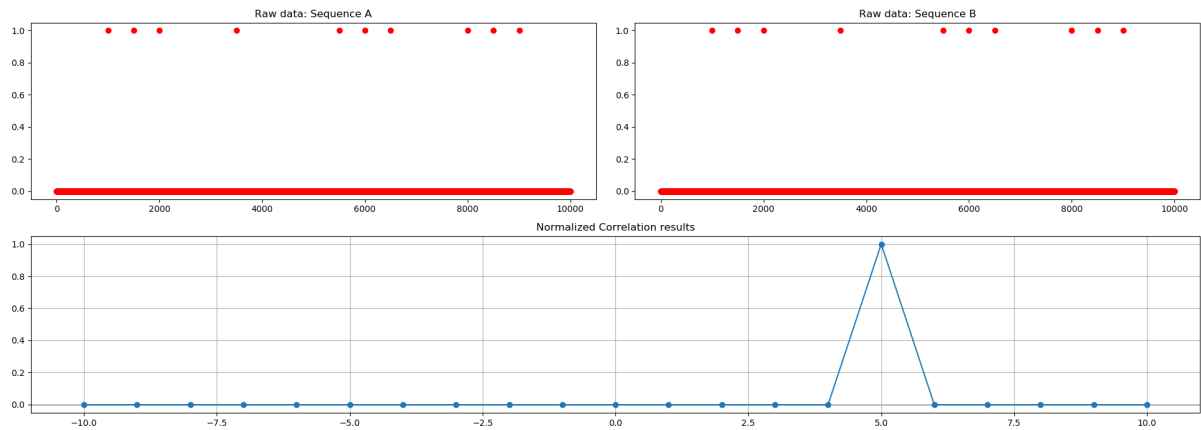


Abbildung 33: Ergebnis der Funktion bei Default-Parametern<sup>31</sup>.

### 8.2.2 Auswirkungen des Parameter: plotNormalizedData

Durch Setzen des optionalen Parameters „plotNormalizedData“ auf True, werden die beiden Sequenzen, zusätzlich in normalisierter Form dargestellt. Siehe dazu Abbildung 34.

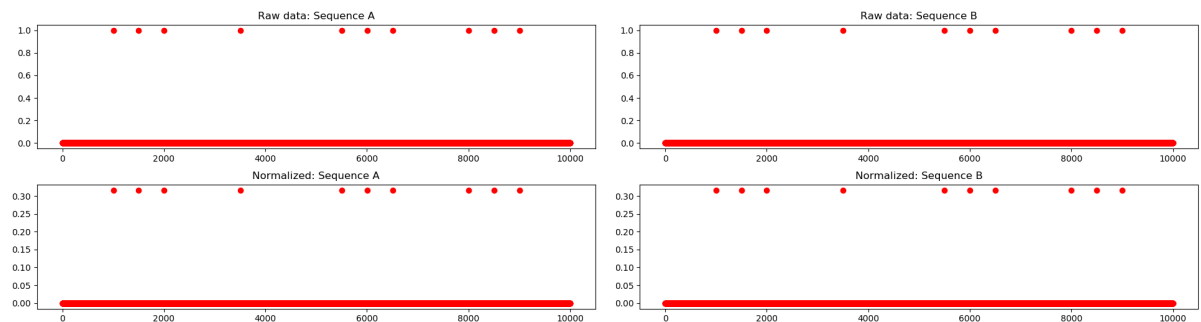


Abbildung 34: Ergebnis der Funktion bei „plotNormalizedData = True“ (Darstellung der Ergebnisse weggelassen)<sup>32</sup>.

### 8.2.3 Auswirkungen des Parameter: plotCorrelations

Durch Setzen des optionalen Parameters „plotCorrelations“ auf True, wird die Kreuz-Korrelation der beiden Sequenzen berechnet. Dabei werden alle drei Berechnungsmethoden „Valid“, „Same“ und „Full“ dargestellt. Siehe dazu Abbildung 35.

<sup>31</sup> Quelle: Eigene Darstellung

<sup>32</sup> Quelle: Eigene Darstellung

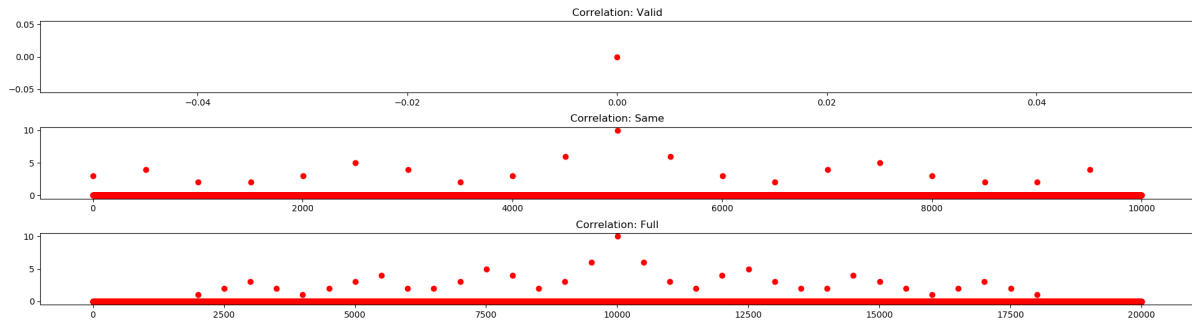


Abbildung 35: Zusätzliche Ausgabe der Funktion bei „plotCorrelations = True“ (Darstellung der Ergebnisse weggelassen)<sup>33</sup>.

### 8.2.4 Auswirkungen des Parameter: plotNonNormalizedResults

Durch Setzen des optionalen Parameters „plotNonNormalizedResults“ auf True, können die Ergebnisse zusätzlich in unnormalisierter Form ausgegeben werden. Siehe dazu Abbildung 36.

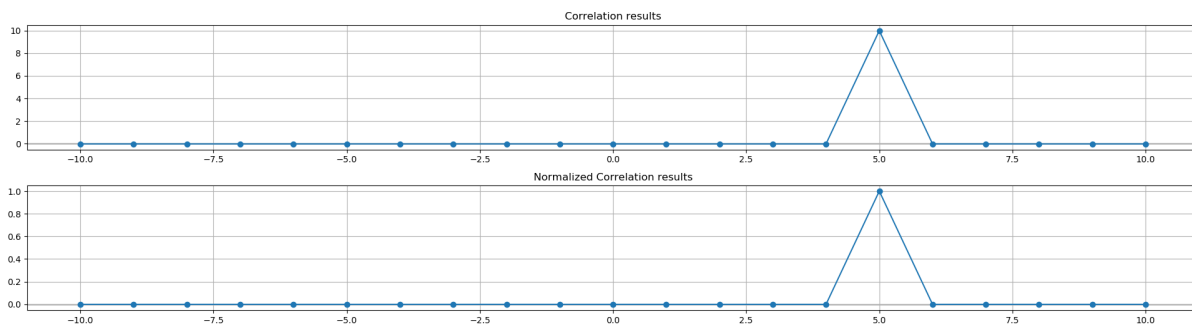


Abbildung 36: Zusätzliche Ausgabe der Funktion bei „plotNonNormalizedResults = True“<sup>34</sup>.

### 8.2.5 Auswirkungen des Parameter: plotNormalizedResults

Durch Setzen des optionalen Parameters „plotNormalizedResults“ auf False, kann die Ausgabe der normalisierten Ergebnisse verhindert werden.

<sup>33</sup> Quelle: Eigene Darstellung

<sup>34</sup> Quelle: Eigene Darstellung

### 8.2.6 Auswirkungen des Parameter: subtractMeanFromResult

Um die Auswirkung des Parameters darzustellen, werden folgende Sequenzen vorausgesetzt (je 10000 Einträge):

SeqA:

$$a_n = \begin{cases} 2 & \text{wenn } n \text{ enthalten in } [1005, 2005, 1505, 3505, 5505, 6005, 6505, 8005, 8505, 9005] \\ 1 & \text{sonst} \end{cases}$$

SeqB:

$$b_n = \begin{cases} 2 & \text{wenn } n \text{ enthalten in } [1000, 2000, 1500, 3500, 5500, 6000, 6500, 8000, 8500, 9000] \\ 1 & \text{sonst} \end{cases}$$

Die Sequenzen sind wieder um einen Wert von 5 verschoben, allerdings wechseln diese nicht zwischen 0 und 1, sondern zwischen 1 und 2. Die Ergebnisse können dabei unübersichtlich werden, deshalb wird ohne Angabe des Parameters, True als Standardwert verwendet. Dadurch wird das arithmetische Mittel des Ergebnis vom Ergebnis selbst abgezogen. Abbildung 38 und 37 zeigen die Auswirkung des Parameters.

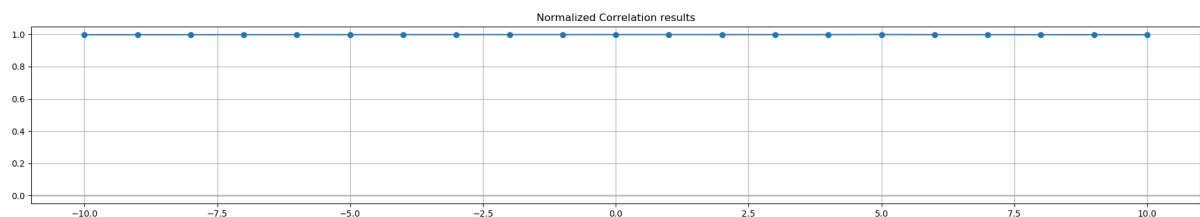


Abbildung 37: Darstellung des Ergebnis bei „subtractMeanFromResult = False“. Ausschläge sind sehr schwer erkennbar<sup>35</sup>.

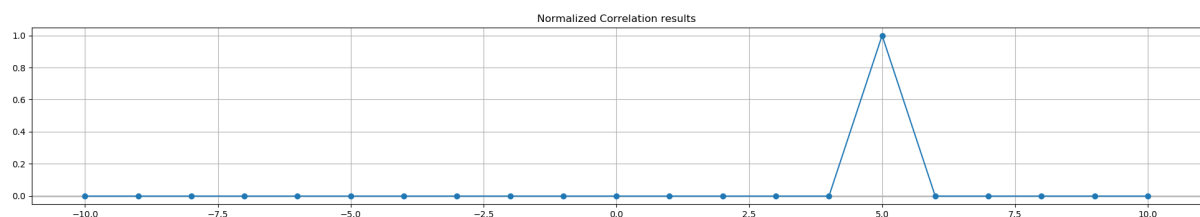


Abbildung 38: Darstellung des Ergebnis bei „subtractMeanFromResult = True“. Ausschläge sind besser erkennbar<sup>36</sup>.

<sup>35</sup> Quelle: Eigene Darstellung

## 8.3 Beispiel-Skripte

Ein Skript welches die Funktion aus diesem Kapitel nutzt ist als Beispiel im Projekt unter „example\_script\_crosscorrelation.py“ abgelegt.

Das Skript das im Zuge der Automatisierung verwendet wird, ist im Unterverzeichnis „cross-correlation“ unter „automated\_crosscorrelation.py“ zu finden.

---

<sup>36</sup> Quelle: Eigene Darstellung

## 9 Suche nach Pattern mit Hilfe der Kreuz-Korrelation

Das Python Skript „functions\_crosscorrelation\_patternsearch.py“ bietet zwei Funktionen an, die verwendet werden können, um nach einem gegebenen Pattern in einer Sequenz zu suchen. Ein Beispiel für die Verwendung ist in „example\_script\_crosscorrelation.py“ abgebildet.

### 9.1 Angebotene Funktionen

Folgende Funktionen werden angeboten:

#### 9.1.1 Funktion: `getCorrelationDataForPatternSearch`

Mit Hilfe der „`getCorrelationDataForPatternSearch`“ Funktion kann nach einem Pattern in einer gegebenen Sequenz gesucht werden. Dafür sind folgende Parameter nutzbar:

**seqA** Die Sequenz in der nach dem Pattern gesucht werden soll. Übergeben in Form eines eindimensionalen Arrays. Die enthaltenen Zahlenwerte müssen in Float konvertierbar sein.

**pattern** Das Pattern nach dem gesucht werden soll. Übergeben in Form eines eindimensionalen Arrays. Die enthaltenen Zahlenwerte müssen in Float konvertierbar sein.

**plotCorrelation** [Default: False] Auf True setzen, wenn die berechnete Kreuz-Korrelation auf dem Monitor dargestellt werden soll.

Die Funktion gibt die berechnete Kreuz-Korrelation zurück, die in der nächsten Funktion „extractIndicesFromCorrelationData“ verwendet werden kann.

Die Funktion verwendet intern die von NumPy bereitgestellte „correlate“ Funktion. Als Modus wird der in Kapitel 7.3.1 beschriebene Modus „Valid“ genutzt.

### 9.1.2 Funktion: extractIndicesFromCorrelationData

Die Funktion kann verwendet werden, um die Indizes zu berechnen, an denen das Pattern in der Sequenz auftritt. Dafür muss der Funktion das Ergebnis der „getCorrelationDataForPatternSearch“ Funktion übergeben werden. Als Rückgabe erhält man die Indizes und der Wert der Korrelation an dieser Stelle.

Optional kann ein Schwellwert übergeben werden. Je nachdem wie dieser Schwellwert gewählt ist, werden mehr oder weniger Indizes zurückgegeben. Bei einem hohen Schwellwert wird jeweils nur der erste Index zurückgegeben, bei dem das Pattern auftritt. Bei einem niedriger gewählten Schwellwert werden alle Indizes zurückgegeben, an denen das Pattern auftritt. Bei einem zu niedrigem Schwellwert werden auch Indizes zurückgegeben, die außerhalb des eigentlichen Patterns liegen. Diese Bereiche befinden sich direkt vor und nach dem Ende des Patterns. Ein zu niedriger Schwellwert kann erkannt werden, wenn der gefundene Bereich größer als das Pattern selbst ist. Durch einen höheren Schwellwert kann dieses „Auswaschen“ oder „Aufweichen“ verhindert werden.

### 9.1.3 Beispiel für die Pattern-Suche

Vorbedingung: Eine Sequenz mit 10000 Einträgen.

SeqA:

$$a_n = \begin{cases} 1 & \text{wenn } n \text{ enthalten in } [1005, 1007, 1010, 6005, 6007, 6010] \\ 0 & \text{sonst} \end{cases}$$

Gesuchtes Pattern:

$$pattern = [1, 0, 1, 0, 0, 1, 0]$$

Das Pattern tritt in der gegebenen Sequenz zwei Mal auf (an Index 1005 und 6005). Der Aufruf der „getCorrelationDataForPatternSearch“ Funktion mit dem „plotCorellation = True“ zeigt die in Abbildung 39 dargestellte Kreuz-Korrelation.

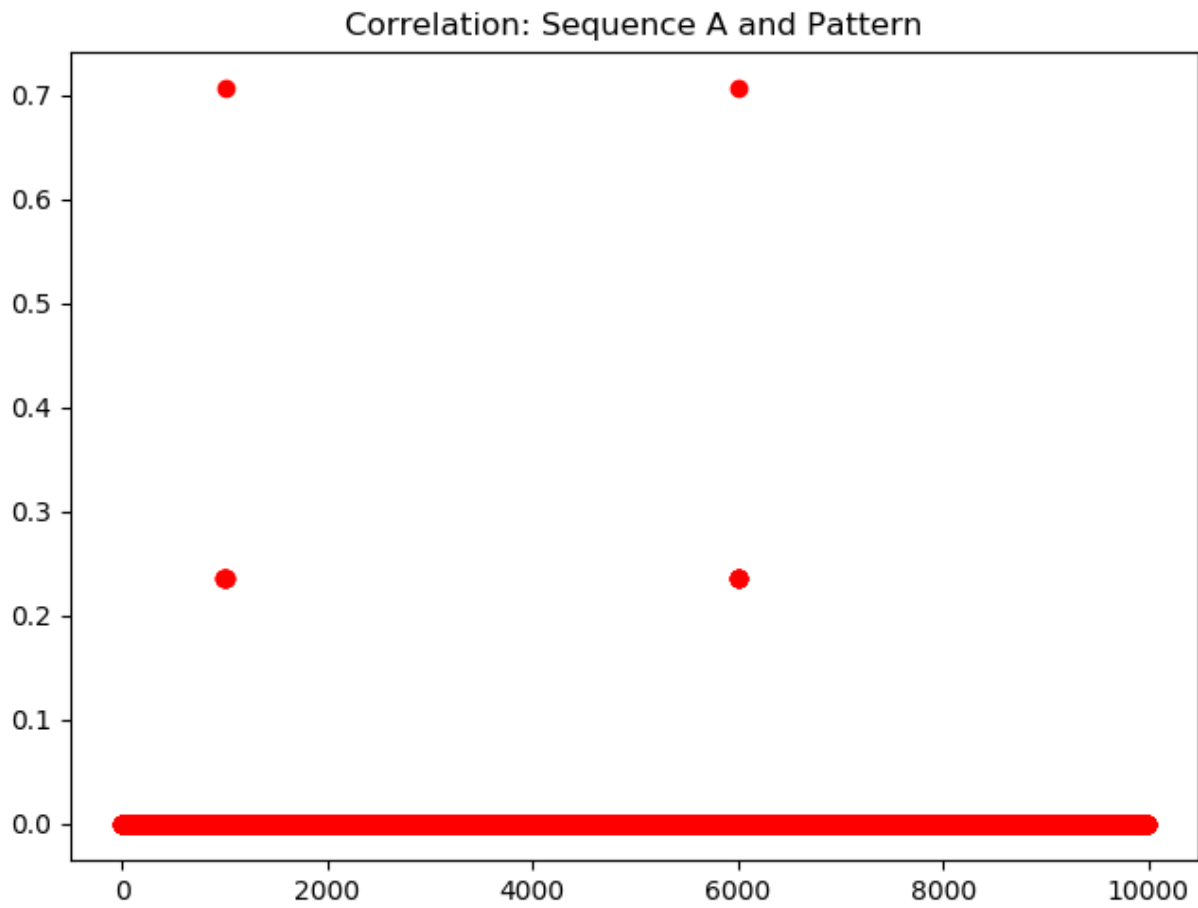


Abbildung 39: Kreuz-Korrelation der Patternsuche<sup>37</sup>.

---

<sup>37</sup> Quelle: Eigene Darstellung



Mit Hilfe der dargestellten Kreuz-Korrelation kann der Schwellwert für die „extractIndices-FromCorrelationData“ festgelegt werden. Ein Aufruf mit einem Schwellwert von „0.3“ liefert folgendes Ergebnis:

```
[6005,0.70710678],  
[1005,0.70710678]
```

Dabei handelt es sich um die Indizes, an denen das Pattern in der Sequenz beginnt (inklusive den Wert an dieser Stelle).

Ein Aufruf mit einem Schwellwert von „0.2“ liefert dagegen folgendes Ergebnis:

```
[6005,0.70710678],  
[1005,0.70710678],  
[6003,0.23570226],  
[1000,0.23570226],  
[6007,0.23570226],  
[6010,0.23570226],  
[6002,0.23570226],  
[6000,0.23570226],  
[1010,0.23570226],  
[1008,0.23570226],  
[1007,0.23570226],  
[1003,0.23570226],  
[1002,0.23570226],  
[6008,0.23570226]
```

Darin enthalten sind Indizes im Bereich von 1000 bis 1010 sowie 6000 bis 6010. Die Bereiche sind größer als das Pattern selbst, allerdings ist das Pattern in diesen Bereichen enthalten.

# 10 Ausführung der automatisierten Skripte

Die einzelnen Module sind durch Skripte automatisierbar. Ein Skript welches alle Module ausführt und als Einstiegspunkt dient, ist in der „automated\_script.py“-Datei abgelegt und kann direkt verwendet werden. Voraussetzung für die Ausführung sind die Pakete aus Kapitel 5, deren Installation im entsprechenden Kapitel beschrieben sind.

Automatisierung bedeutet in diesem Kontext, dass Quelldateien automatisch eingelesen und verarbeitet werden. Die Ergebnisse werden in Form von PDF-Dateien im Quellverzeichnis mit abgelegt. Beispiele für die Ergebnisse einer Ausführung sind im Anhang unter Kapitel 12.1 dargestellt.

## 10.1 Anleitung zur Ausführung der Skripte

Das automatisierte Skript kann über die Kommandozeile ausgeführt werden. Optional ist es möglich einen Pfad zu einem Ordner anzugeben, in dem die Quelldateien abgelegt sind. Wenn kein Pfad angegeben wird, wird ein Ordner mit dem Namen „sourceFiles“ verwendet, der sich im gleichen Verzeichnis wie das Skript befinden muss. Sollte dieser Ordner nicht existieren, legt das Skript diesen automatisch an. Die Quelldateien können dann in diesem Ordner abgelegt werden und werden bei der nächsten Ausführung des Skripts verarbeitet. Alternativ dazu kann Docker verwendet werden, siehe dazu auch im Anhang unter Kapitel 12.3.

### 10.1.1 Kommando zur Ausführung

Nachdem man mit einer Konsole in das Verzeichnis gewechselt ist in dem der Code angelegt ist, lässt sich das Skript über folgendes Kommando ausführen:

```
1 Windows Kommandozeile:  
2     python automated_script.py  
3 Windows Powershell:  
4     python .\automated\_script.py  
5 Mit optionalem Parameter:  
6     python automated_script.py C:\Temp\sourcefolder
```

Listing 10: Kommando zur Ausführung des Skripts unter Windows

Unter Linux/macOS analog dazu:

```
1 Je nach Konfiguration des Systems:  
2     python automated_script.py  
3 Oder:  
4     python3 automated_script.py
```

Listing 11: Kommando zur Ausführung des Skripts unter Linux

Anschließend wird das Skript ausgeführt und die Log-Meldungen werden auf der Konsole ausgegeben. Ein Beispiel für die ausgegebenen Log-Meldungen ist im Anhang unter Listing 14 zu sehen.

## 10.2 Hinweis zur zusätzlichen Anzeige der generierten Daten

Bei der Ausführung werden automatisiert PDF-Dateien erzeugt. Die Auflösung der PDF-Dateien kann bei der Analyse zu einem Problem werden, wenn diese nicht mehr ausreicht. Deshalb ist es möglich, eine Anzeige der generierten Daten zu aktivieren, bei der mehr Möglichkeiten zur Analyse bestehen. Dabei öffnet Matplotlib ein Fenster, in dem die Graphen genauer betrachtet werden können. Die Fenster sind in Abbildung 40 dargestellt.

Die Anzeige ist einzeln für die Kategorisierung und die Kreuzkorrelation aktivierbar. Entsprechende Kommentare sind in den Skripten gesetzt.

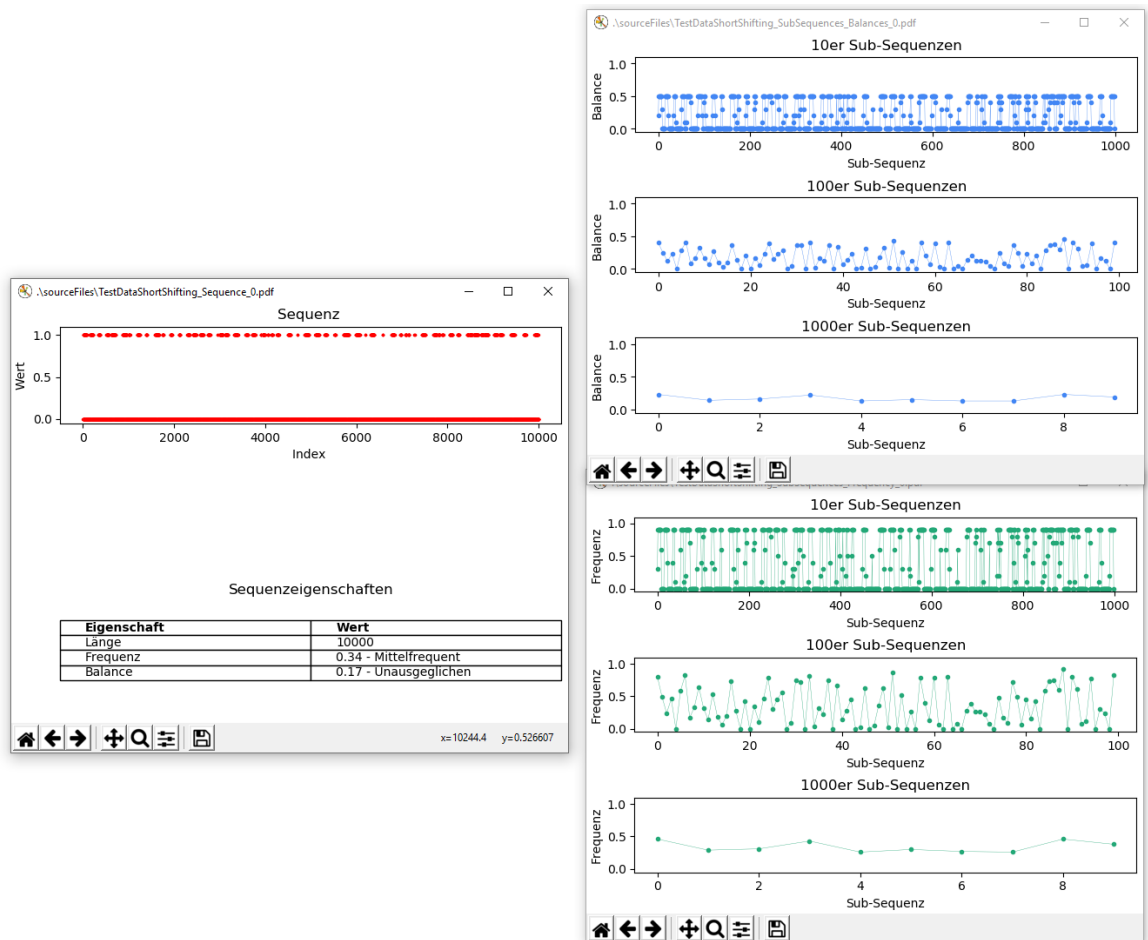
**Warnung:** Das Skript wird so lange blockiert, bis alle geöffneten Fenster wieder geschlossen wurden. Dies ist eine Beschränkung durch Matplotlib.

### 10.2.1 Anzeige der Kategorisierungsdaten

Die Anzeige in einem extra Fenster kann in dem Skript „automated\_script.py“ beim Aufruf der Funktion „executeCategorization“ aktiviert werden. Dafür muss der zweite Parameter auf True gesetzt werden. Dann wird sowohl die Übersichtseite einer Sequenz, sowie die Darstellung der Sub-Sequenzen angezeigt (Balance und Frequenz je in einem eigenen Fenster).

### 10.2.2 Anzeige der Kreuzkorrelation

Da die Funktionen zur Berechnung der Kreuzkorrelation deutlich mehr Einstellungsmöglichkeiten besitzen, sind diese in einer eigenen Klasse abgelegt. Diese Einstellungen werden beim Aufruf zur Berechnung der Kreuzkorrelation mitgegeben. Wenn man die zusätzliche Anzeige aktivieren möchte, muss man in der „automated\_crosscorrelation.py“ Datei den Parameter „drawResults“ auf True setzen.

Abbildung 40: Stellt die Ergebnisse der Kategorisierung in zusätzlichen Fenstern dar<sup>38</sup>.<sup>38</sup> Quelle: Eigene Darstellung

## 10.3 Ablauf der automatisierten Ausführung

Die folgende Aktionen werden durch das bereitgestellte Skript („automated\_script.py“) ausgeführt. Der Übersicht wegen, werden einige Aktionen zusammengefasst und nicht im Detail dargestellt. Hinweis: Die Dateien des Quellverzeichnisses werden in Batches der Größe 10 abgearbeitet. Die Verarbeitung erfolgt Batch-weise, um den Ressourcenverbrauch gering zu halten. Ansonsten müssten zu Beginn alle Dateien eingelesen und im Arbeitsspeicher gehalten werden. Da das automatisierte Skript eine potenziell große Anzahl an Quelldateien lesen muss, könnte dies zu einem großen Ressourcenbedarf führen.

1. Auswahl des Quellverzeichnis (über optionalen Parameter oder Standardverzeichnis).
2. Aufteilung der Quelldatei-Pfade in Batches. Je Batch:
3. Einlesen der CSV-Dateien aus dem ausgewählten Quellverzeichnis.
4. Verarbeitung der einzelnen Quelldateien
  - a) Anlegen eines Analysis-Requests
  - b) Analyse und Verarbeitung der einzelnen Zeilen zur Unterscheidung zwischen Rohdaten und Metadaten
  - c) Expansion der Rohdaten (Dekomprimierung)
5. Kategorisierung der einzelnen Sequenzen
  - a) Analyse der Frequenz
    - i. Berechnung der Frequenz für die Quellsequenz
    - ii. Bildung der Subsequenzen inklusive Berechnung der jeweiligen Frequenzen
  - b) Analyse der Balance
    - i. Berechnung der Balance für die Quellsequenz
    - ii. Bildung der Subsequenzen inklusive Berechnung der jeweiligen Balance
  - c) Generierung der PDF-Dateien zur Darstellung der Ergebnisse der Kategorisierung
6. Berechnung der Kreuzkorrelation zwischen Sequenzen einer Quelldatei
  - a) Vergleich der Quellsequenzen hinsichtlich der Möglichkeit die Kreuz-Korrelation zu berechnen
  - b) Berechnung der Kreuzkorrelation bei geeigneten Sequenz-Paaren
  - c) Generierung der PDF-Dateien zur Darstellung der Ergebnisse

## 10.4 Benennung der erzeugten Dateien

Die Namen der generierten Dateien beginnen immer mit dem Namen der jeweiligen Quelldatei, damit eine Zuordnung gewährleistet ist. Wenn eine Quelldatei mehr als eine Sequenz enthält, werden die einzelnen Sequenzen von 0 an durchnummeriert. Die erste Sequenz in der Quelldatei bekommt dabei den Index 0, die zweite den Index 1 usw. Wenn zwei Sequenzen mit gleicher Länge in einer Quelldatei abgelegt werden, wird automatisch die Kreuzkorrelation zwischen den beiden Sequenzen berechnet. Um die Ergebnisdatei zuordnen zu können, bekommen diese eine Benennung entsprechend der Sequenzindizes. Die Namen der Ergebnisdateien zu den Sub-Sequenzen beinhalten ebenfalls den Index der zugehörigen Sequenz. Ein Beispiel der Benennung ist im Anhang unter Abbildung 42 dargestellt.

## 10.5 Übernommene Metadaten

Wenn die Quelldateien Metadateninformationen beinhalten, werden diese in die PDFs übernommen. Dabei ist es unerheblich, ob jede Sequenz in einer Quelldatei Metadateninformationen beinhaltet. Wenn die Metadaten vorhanden sind, werden diese für die entsprechende Sequenz übernommen.

### 10.5.1 Beispiel

In Listing 12 sind vier Sequenzen abgebildet, von denen nur zwei über Metadateninformationen verfügen. Die generierten Dateien werden mit den Metadaten erweitert, allerdings nur bei den beiden Sequenzen, die über die zusätzlichen Information verfügen.

```

1 $Machine,Maschine2,Start,2015-07-22-00-00-00,End,2015-07-22-24-00-00,Interval,1000ms$
2 1004|0,1|1,999|0,1|1,2000|0,1|1
3 $Machine,Maschine1,Start,2015-07-22-00-00-00,End,2015-07-22-24-00-00,Interval,1000ms$
4 1000|0,1|1,999|0,1|1,2000|0,1|1,4|0
5 1000|0,1|1,999|0,1|1,2000|0,1|1,4|0
6 1000|0,1|1,999|0,1|1,2000|0,1|1,4|0

```

Listing 12: Quelldatei mit Sequenzen (mit und ohne Metadaten)

## 10.6 Ergebnis der Ausführung

Die Ergebnisse der Analyse wird im Quellverzeichnis in Form von PDF-Dateien abgelegt. Beispiele für die generierten PDF Dateien sind im Anhang unter Kapitel 12.1 zu finden.

## 10.7 Verbesserungsmöglichkeiten und Erweiterungen

Da Matplotlib nicht threadsafe ist<sup>39</sup>, kann die Generierung von PDF-Dateien nicht ohne Weiteres parallelisiert werden. Bei einer parallelen Generierung würde Matplotlib gleichzeitig aus verschiedenen Programmbereichen aus aufgerufen werden. Bei Programmen die nicht threadsafe sind, kann dies zu Fehlern führen. Wenn ein Programm threadsafe ist, sind solche parallelen Aufrufe ohne Behinderungen oder Fehler möglich.

Da ein Großteil der Ausführungszeit für die Generierung der PDFs benötigt wird, wäre allerdings eine Parallelisierung in diesem Bereich von Vorteil. Eine mögliche Erweiterung um die Leistung durch Parallelisierung zu steigern, wäre die Einführung eines Multi-Prozess-Ansatzes zur Generierung der PDF-Dateien. Dabei würde für jede zu generierende Datei ein eigener Prozess gestartet werden, dessen Speicherbereich von den anderen Generierungsprozessen getrennt ist. Da ein Prozess schwergewichtiger als ein einfacher Thread ist, hat diese Lösung natürlich auch seine eigenen Nachteile. Ein weiterer Nachteil wäre die notwendige Verwaltung der Prozesse sowie die Kommunikation zwischen den Prozessen. Eine solche Lösung ist nur mit zusätzlichem Entwicklungsaufwand umsetzbar.

---

<sup>39</sup> Siehe: StackOverflow 2016.



# 11 Fazit

In Kapitel 2 wurde die binäre Sequenz, welche die Zustände 0 für *kein Fehlerzustand* und 1 für *Fehlerzustand* definiert, eingeführt. Sie stellt die geringste Komplexität einer Sequenz für ein Mehrzustandssystem dar und besitzt die Eigenschaften der Balance und Frequenz mit der sie beschrieben werden kann. Der Vorteil der Sequenzen liegt darin, dass sie sich auch für komplexerer Produktionsdaten verwenden lassen. Mit weiteren Symbolen, lassen sich weitere Zustände abbilden.

Maschinen in Fertigungslinien haben meistens Abhängigkeiten zueinander. Diese lassen sich durch die Visualisierung der Daten oder deren Eigenschaften, aber auch durch die Kreuzkorrelationsfunktion nachweisen. Für die Klassifizierung von Sequenzen wurden zudem grundlegende Muster definiert. Diese lassen sich ebenfalls über die Kreuzkorrelation in Sequenzen nachweisen.

Die auf Grundlage der definierten Eigenschaften vorgestellten Vorgehensweisen zur Analyse von Produktionsdaten, lassen sich automatisiert auf Daten anwenden. Für die Verarbeitung wurde ein proprietäres Datenformat beschrieben. Daneben wurde ein Tool zur Generierung von Simulationsdaten sowie ein Konverter für ein Maschinendatenformat in C# mit dem .NET Core Framework entwickelt.

Die automatisierte Auswertung der Simulations- bzw. Produktionsdaten findet mit Hilfe einer in Python geschriebenen Anwendung statt. Diese bietet Module zur Berechnung der Balance und Frequenz sowie der Kreuzkorrelation. Automatisiert können so beliebig viele Sequenzen nach den beschriebenen Verfahren verarbeitet werden. Die erstellten PDFs können zur weiteren Analyse herangezogen werden.

Auf Grundlage dieser Arbeit kann der binäre Ansatz weiter verfolgt und vertieft werden.

# Literatur

- Amberg, Brian (7. Juli 2010). *Convolution of box signal with itself.gif*. Wikimedia. URL: [https://commons.wikimedia.org/wiki/File:Convolution\\_of\\_box\\_signal\\_with\\_itself.gif](https://commons.wikimedia.org/wiki/File:Convolution_of_box_signal_with_itself.gif) (besucht am 04.11.2018).
- Athanasias, Dale (14. Mai 2014). *Python Wiki: NumPy*. URL: <https://wiki.python.org/moin/NumPy> (besucht am 24.04.2019).
- Cmglee (3. Sep. 2013). *Comparison convolution correlation de.svg*. Wikimedia. URL: [https://commons.wikimedia.org/wiki/File:Comparison\\_convolution\\_correlation\\_de.svg](https://commons.wikimedia.org/wiki/File:Comparison_convolution_correlation_de.svg) (besucht am 04.11.2018).
- Community, SciPy (2018). *Documentation: numpy.correlate*. URL: <https://docs.scipy.org/doc/numpy/reference/generated/numpy.correlate.html#numpy.correlate> (besucht am 04.11.2018).
- Duden (2019). *Sequenz*. URL: <https://www.duden.de/rechtschreibung/Sequenz> (besucht am 12.06.2019).
- Foundation, The Python Software (24. Apr. 2019). *pip - The Python Package Installer*. URL: <https://pip.pypa.io/en/stable/> (besucht am 24.04.2019).
- Fuxjäger, Maximilian (2017). *Python: Numeric and Scientific*. URL: <https://wiki.python.org/moin/NumericAndScientific> (besucht am 24.04.2019).
- Gillhuber, Andrea (2019). *Künstliche Intelligenz in der Smart Factory*. URL: <https://www.scope-online.de/smart-industry/kuenstliche-intelligenz-in-der-smart-factory.html> (besucht am 12.06.2019).
- Inductiveload (5. März 2009). *Convolution Animation (Gaussian).gif*. Wikimedia. URL: [https://commons.wikimedia.org/wiki/File:Convolution\\_Animation\\_\(Gaussian\).gif](https://commons.wikimedia.org/wiki/File:Convolution_Animation_(Gaussian).gif) (besucht am 04.11.2018).
- MathWorld, Wolfram (2018). *Cross-Correlation*. URL: <http://mathworld.wolfram.com/Cross-Correlation.html> (besucht am 04.11.2018).
- Rahmann, Prof. Dr. Sven, Hrsg. (2014). *Algorithmen auf Sequenzen*. Deutsch. TU Dortmund.

- Rossant Cyrille, 1985- [Verfasser/in], Hrsg. (2013). *Learning IPython for interactive computing and data visualization: learn IPython for interactive Python programming, high-performance numerical computing, and data visualization*. Englisch. Community experience distilled. Fernleihe. Birmingham [u.a.]: Packt Publ., IV, 123 Seiten. ISBN: 978-1-78216-993-2.
- Runkler, T.A., Hrsg. (2015). *Data Mining*. Deutsch. Springer.
- StackOverflow, o.V. (2016). *Why can't matplotlib plot in a different thread?* URL: <https://stackoverflow.com/questions/34764535/why-cant-matplotlib-plot-in-a-different-thread> (besucht am 24. 04. 2019).
- (2017). *signal.correlate 'same'/'full' meaning?* URL: <https://stackoverflow.com/questions/43736297/signal-correlate-same-full-meaning> (besucht am 04. 11. 2018).
- Stevens, S.S., Hrsg. (1946). *On the theory of scales of measurement*. Englisch. Science. team, The Matplotlib development (28. März 2019). *Matplotlib*. URL: <https://matplotlib.org/> (besucht am 24. 04. 2019).
- Wiktionary (2019). *Sequenz*. URL: <https://de.wiktionary.org/wiki/Sequenz> (besucht am 12. 06. 2019).
- Woyand, Hans-Bernhard (2017). *Python für Ingenieure und Naturwissenschaftler: Einführung in die Programmierung, mathematische Anwendungen und Visualisierungen : mit zahlreichen Bildern und Tabellen sowie 68 Aufgaben*. Deutsch. "Im Internet: Beispiele und Lösungen zu den Aufgaben" - Cover. München: Hanser, IX, 264 Seiten. ISBN: 3446451986.

# 12 Anhang: Implementierung

## 12.1 Beispiel: Ergebnisse im Quellverzeichnis

In Abbildung 41 ist das Quellverzeichnis dargestellt. Es beinhaltet die Quelldateien, die analysiert werden sollen.

Name	Änderungsdatum	Typ	Größe
 cross_corr_test.csv	11.06.2019 16:43	OpenOffice.org 1....	1 KB
 TestDataShortShifting.csv	11.06.2019 15:48	OpenOffice.org 1....	14 KB

Abbildung 41: Abbildung des Quellverzeichnisses, bevor das Skript ausgeführt wurde<sup>40</sup>.

Bei der Ausführung des Skripts werden die generierten PDF-Dateien im Quellverzeichnis mit abgelegt. Dadurch ergibt sich danach das in Abbildung 42 dargestellte Ergebnis.













Name	Änderungsdatum	Typ	Größe
 cross_corr_test.csv	11.06.2019 16:43	OpenOffice.org 1....	1 KB
 cross_corr_test_CrossCorrelation_Sequence_0_Sequence_1.pdf	11.06.2019 17:22	Adobe Acrobat D...	16 KB
 cross_corr_test_Sequence_0.pdf	11.06.2019 17:22	Adobe Acrobat D...	22 KB
 cross_corr_test_Sequence_1.pdf	11.06.2019 17:22	Adobe Acrobat D...	22 KB
 cross_corr_test_SubSequences_Balances_0.pdf	11.06.2019 17:22	Adobe Acrobat D...	12 KB
 cross_corr_test_SubSequences_Balances_1.pdf	11.06.2019 17:22	Adobe Acrobat D...	12 KB
 cross_corr_test_SubSequences_Frequency_0.pdf	11.06.2019 17:22	Adobe Acrobat D...	10 KB
 cross_corr_test_SubSequences_Frequency_1.pdf	11.06.2019 17:22	Adobe Acrobat D...	10 KB
 TestDataShortShifting.csv	11.06.2019 15:48	OpenOffice.org 1....	14 KB
 TestDataShortShifting_Sequence_0.pdf	11.06.2019 17:22	Adobe Acrobat D...	22 KB
 TestDataShortShifting_SubSequences_Balances_0.pdf	11.06.2019 17:22	Adobe Acrobat D...	16 KB
 TestDataShortShifting_SubSequences_Frequency_0.pdf	11.06.2019 17:22	Adobe Acrobat D...	15 KB

Abbildung 42: Abbildung des Quellverzeichnisses, nachdem das Skript ausgeführt wurde<sup>41</sup>.

Die generierten Dateien haben dabei folgenden Inhalt (exemplarisch):

---

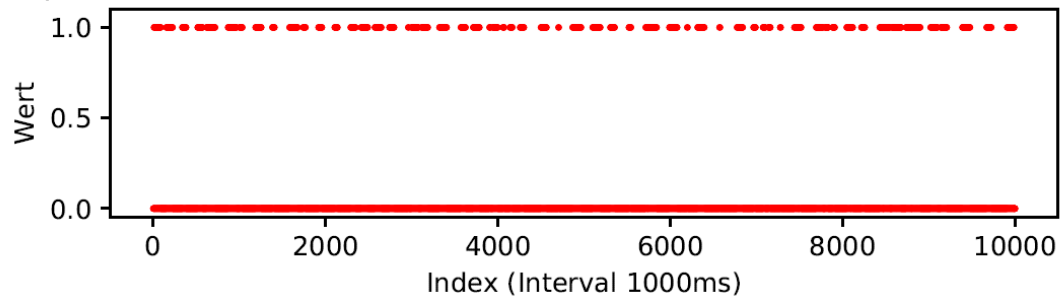
<sup>40</sup> Quelle: Eigene Darstellung

<sup>41</sup> Quelle: Eigene Darstellung

### 12.1.1 Kategorisierung

Abbildung 43 zeigt exemplarisch den Inhalt einer generierten PDF-Datei, die die berechneten Eigenschaftswerte beinhaltet.

Sequenz Maschine2 Start: 2015-07-22-00-00-00 End: 2015-07-22-02-46-40



#### Sequenzeigenschaften

Eigenschaft	Wert
Länge	10000
Frequenz	0.34 - Mittelfrequent
Balance	0.17 - Unausgeglichen

Abbildung 43: Stellt den Inhalt einer generierten PDF-Datei mit dem Kategorisierungsergebnis dar<sup>42</sup>.

<sup>42</sup> Quelle: Eigene Darstellung

### 12.1.2 Sub-Sequenzen: Frequenz

Abbildung 44 zeigt exemplarisch den Inhalt einer generierten PDF-Datei, die die Frequenzen der Sub-Sequenzen beinhaltet.

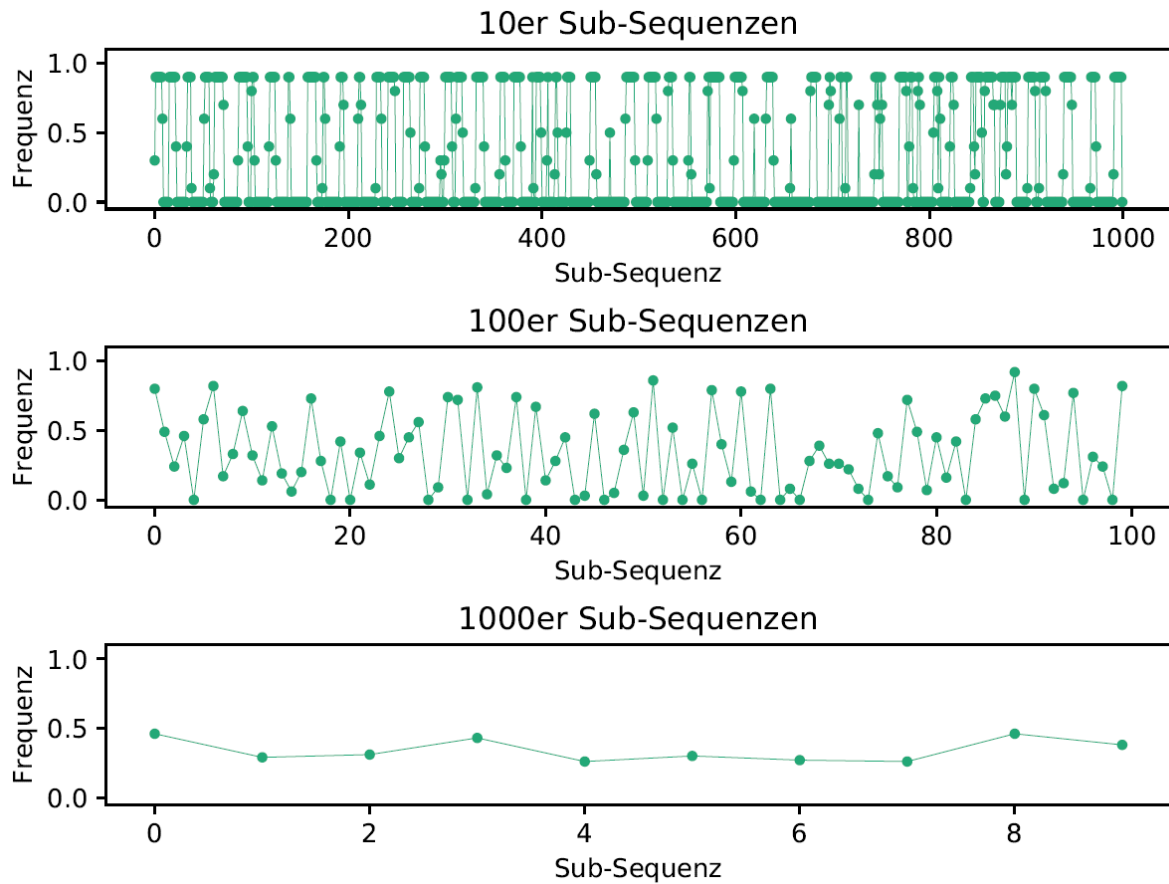


Abbildung 44: Stellt den Inhalt einer generierten PDF-Datei mit den Frequenzen der Sub-Sequenzen dar<sup>43</sup>.

<sup>43</sup> Quelle: Eigene Darstellung

### 12.1.3 Sub-Sequenzen: Balance

Abbildung 45 zeigt exemplarisch den Inhalt einer generierten PDF-Datei, die die Balancen der Sub-Sequenzen beinhaltet.

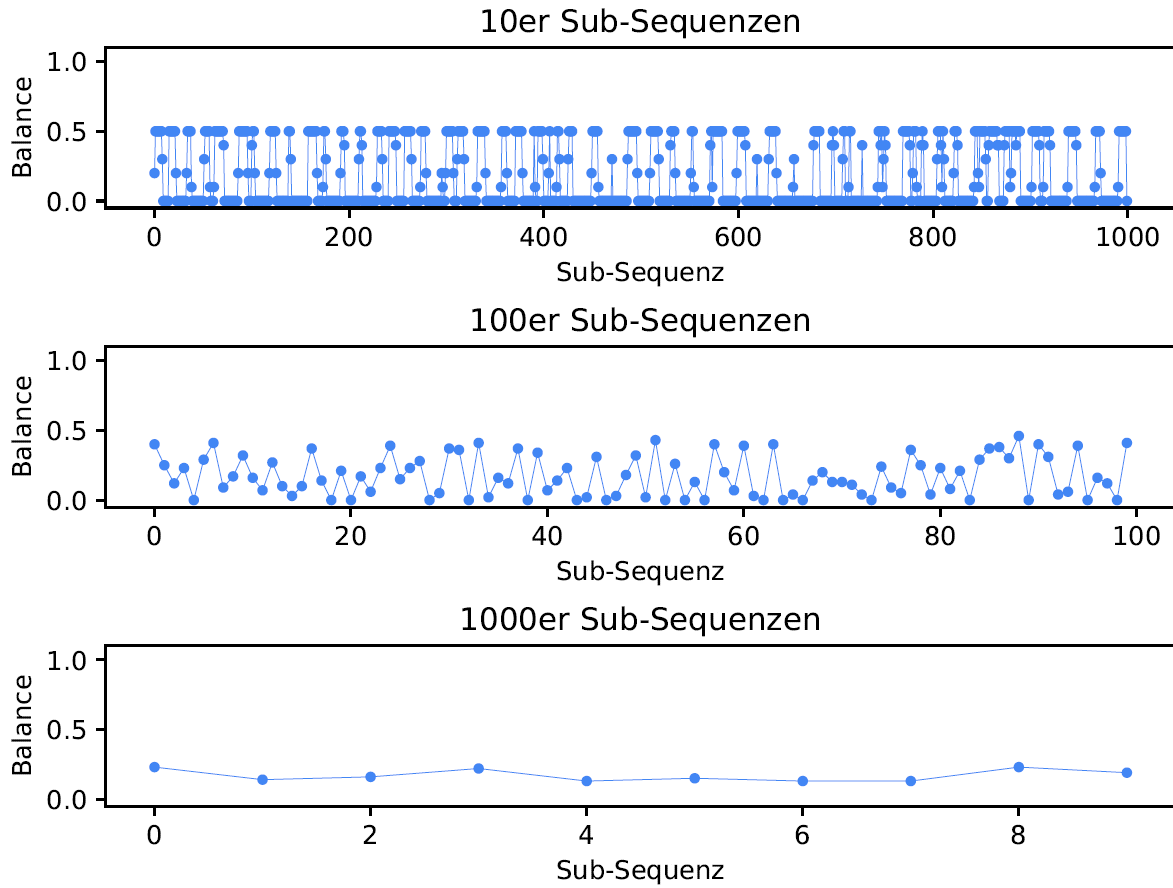


Abbildung 45: Stellt den Inhalt einer generierten PDF-Datei mit den Balancen der Sub-Sequenzen dar<sup>44</sup>.

<sup>44</sup> Quelle: Eigene Darstellung

### 12.1.4 Kreuzkorrelation

Abbildung 46 zeigt exemplarisch den Inhalt einer generierten PDF-Datei, die die berechnete Kreuzkorrelation beinhaltet.

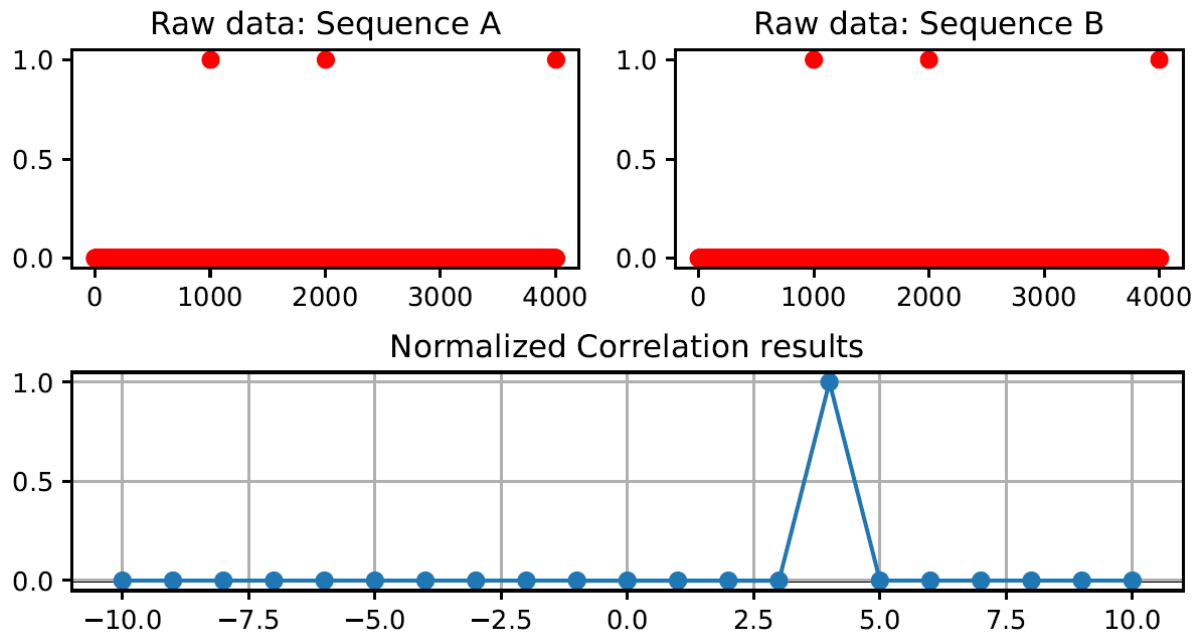


Abbildung 46: Stellt den Inhalt einer generierten PDF-Datei mit dem Ergebnis der Kreuzkorrelation dar<sup>45</sup>.

<sup>45</sup> Quelle: Eigene Darstellung



## 12.2 Log-Meldungen des automatisierten Skripts

Das automatisierte Skript schreibt diverse Log-Meldungen in die Konsole. Listing 14 ist ein Beispiel dafür, was durch das Skript geloggt wird.

```

1 Reading files from source path:
2 .\sourceFiles
3 Found 2 source files
4
5 Reading data from .\sourceFiles\cross_corr_test.csv ...
6 Found sequences: 2
7 Expanding data for .\sourceFiles\cross_corr_test.csv Sequence 0
8 Expanding data for .\sourceFiles\cross_corr_test.csv Sequence 1
9
10 Reading data from .\sourceFiles\TestDataShortShifting.csv ...
11 Found sequences: 1
12 Expanding data for .\sourceFiles\TestDataShortShifting.csv Sequence 0
13
14 Calculating frequency for .\sourceFiles\cross_corr_test.csv Sequence 0
15 Calculating frequency for .\sourceFiles\cross_corr_test.csv Sequence 1
16 Calculating balance for .\sourceFiles\cross_corr_test.csv Sequence 0
17 Calculating balance for .\sourceFiles\cross_corr_test.csv Sequence 1
18 Calculating block size information for .\sourceFiles\cross_corr_test.csv
19 Calculating block infos for .\sourceFiles\cross_corr_test.csv Sequence 0
20 Calculating block infos for .\sourceFiles\cross_corr_test.csv Sequence 1
21
22 Calculating frequency for .\sourceFiles\TestDataShortShifting.csv Sequence 0
23 Calculating balance for .\sourceFiles\TestDataShortShifting.csv Sequence 0
24 Calculating block size information for .\sourceFiles\TestDataShortShifting.csv
25 Calculating block infos for .\sourceFiles\TestDataShortShifting.csv Sequence 0
26
27 Generating pdfs...
28 Generating pdf reports for .\sourceFiles\cross_corr_test.csv Sequence 0
29 Generating pdf reports for .\sourceFiles\cross_corr_test.csv Sequence 1
30 Generating pdf reports for .\sourceFiles\TestDataShortShifting.csv Sequence 0
31
32 Crosscorrelation for file .\sourceFiles\cross_corr_test.csv
33 .\sourceFiles\cross_corr_test.csv exporting Cross-Correlation between sequence 0 and
    1

```

Listing 13: Anhang: Beispiel für generierte Log-Meldungen

## 12.3 Verwendung von Docker

Ein verwendbares Docker-File ist im Quellverzeichnis als „dockerfile“ abgelegt. Aus dieser Datei kann ein Image erstellt werden, das Python und die notwendigen Bibliotheken enthält. Aus diesem Image kann ein Container erstellt werden, in dem ein Quellverzeichnis des Host-Rechners gemountet wird. Dadurch hat der Container Zugriff auf die Dateien in diesem Ordner und kann diese verarbeiten. Im Folgenden sind die notwendigen Befehle dokumentiert, um das Image und den Container zu erstellen und zu verwenden.

```

1 Befehlstemplate zur Erstellung des Images. Der Name für das Image ist frei wählbar,
2 muss aber in den folgenden Befehlen verwendet werden:
3     docker build . -t <image_name>
4 Beispiel:
5     docker build . -t analysisimage
6
7 Die Skripte die verwendet werden, sind ein fester Bestandteil des Images.
8 Wenn Änderungen an den Skripten vorgenommen werden, muss auch das Image neu erstellt
9 werden.
10 Folglich muss dann auch der Container neu erstellt werden.
11
12 Befehlstemplate zur Erstellung eines Containers:
13     docker create -v <Pfad zum Quellordner>:/usr/src/app/sourceFiles --name
14         <container_name> <image_name>
15 Beispiel:
16     docker create -v C:\Temp\sourcefiles:/usr/src/app/sourceFiles --name
17         MyAnalysisContainer analysisimage
18 Der Pfad zum Quellordner ist ein fester Bestandteil des Containers und kann nicht
19 ohne weiteres verändert werden.
20 Soll ein anderer Ordner verwendet werden, muss ein neuer Container erzeugt werden.
21 Dadurch ist auch die Möglichkeit gegeben, unterschiedliche Ordner in
22 unterschiedlichen Containern zu verwenden.
23
24 Befehlstemplate zur Ausführung eines Containers:
25     docker start -a <container_name>
26 Beispiel:
27     docker start -a MyAnalysisContainer
28 Dadurch wird ein Container ausgeführt und die Ausgabe wird auf der Konsole
29 ausgegeben. Der Container beendet sich nach Ausführung von selbst.
30

```

```
31 Weitere, hilfreiche Befehle:
32
33 Löschen eines Containers:
34     docker rm <container_name>
35 Beenden eines Containers:
36     docker stop <container_name>
37 Löschen eines Images:
38     docker rmi <image_name>
```

Listing 14: Anhang: Beispiel für die Verwendung von Docker

# 13 Anhang: Literaturrecherche

Folgende Journals und Websites wurden bei der Literaturrecherche gesichtet:

- <https://www.journals.elsevier.com/data-and-knowledge-engineering>
- <https://www.industrie-management.de/view/industriehome>
  - Autorenhinweise auf der Internetseite
- <https://whitepaper.computerwoche.de>
  - hat einen Whitepaper Bereich, auch speziell für Business Analytics
- <https://www.informatik-aktuell.de/>
  - nur online. Nehmen Themenvorschläge an unter: <https://www.informatik-aktuell.de/ueberuns.html>
- <https://whitepaper.cio.de>
  - Identisch zur Computerwoche => gleicher Inhalt bei den Whitepapers, nur unterschiedliche Startseite
- Informatik Spektrum, Springer Verlag
  - in der Bibliothek vorhanden