FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Seminararbeit

Unübliche Methoden zum Generieren von Zufallszahlen

von

Philip Kaluđerčić

Matrikel-Nr.: 22423250

Betreuung:
Professor Oliver Keszöcze

13. September 2019

Dieses Dokument wurde mit dem Textsatzsystem LaTeX2e erstellt.

DAS PROBLEM mit der sich diese Arbeit auseinandersetzen will, ist das der Generierung von Zufallszahlen. Während die Problemstellung einfach ist,

> *„Bestimme eine natürliche Zahl aus einem Intervall, wobei die Wahrscheinlichkeit jeder Zahl gleich hoch ist.“*

sind die Anwendung vielfältig und selbst für die vollkommen Uninteressierten relevant. Sei es für nur Spiele, oder bis hin zu wissenschaftlichen Simulationen, besteht ein Interesse an einem sinnvollem, effizientem und schnellem Lösungsansatz für das beschriebene Problem.

DIE LÖSUNGEN[1] mit der sich diese Arbeit auseinandersetzen will, werden nicht immer diese Kriterien ganz erfüllen, aber dafür verschiedene Abwägungen und Ansätze demonstrieren.

Zunächst wird es jedoch notwendig sein zunächst eine „übliche“ Methode zu betrachten, um ein Maßstab kennenzulernen, mit dem wir die Alternativen vergleichen können.

1. Herkömmliche Methoden der Zufallszahlen Generierung

Ein einfacher, aber durchaus populärer Ansatz zum generieren von Zufallszahlen beginnt mit einer initialen Zahl Z_0 (*seed*, Zufallssamen). Mit dieser bestimmt es rekursiv alle Folgenden via der Formel[1, S. 9][2, S. 10]

$$Z_{n+1} = (aZ_n + c) \bmod m. \tag{1.1}$$

Dieser Ansatz wird *Linear Congruent Method* genannt, mit den Parameteren a, c und m.

Mit den Beispielwerten $Z_0 = a = c = 7$ und $m = 10$ ergibt sich die Folge Z von Zufallszahlen:

$$7, 6, 9, 0, 7, 6, 9, 0, 7, 6, 9, 0, \ldots, 7, 6, 9, 0, \ldots \tag{1.2}$$

Anhand dieser Folge können wir ein paar Eigenschaften betrachten, die im später interessant sein werden:

[1] Alle Lösungen die hier Präsentiert werden basieren direkt oder indirekt auf der Auswahl aus D.E. Knuth's *The Art Of Computer Programming*, Band 2

Periodenlänge Intuitiv versteht man Zufälligkeit als Abwesenheit von Mustern. Ein Muster ist schlussendlich eine *a priori* Bestimmung des Ablaufs.

Im obigen Beispiel ist das Muster $7, 6, 9, 0$ zu erkennen, welches sich durch die gesamte Folge wiederholt[1]. Diese Teilfolge verstehen wir als **Periode**[1, S. 9][2, S. 10], d.h. die kürzeste Sequenz dessen unendliche Aneinanderreihung die tatsächlich generierte Folge beschreibt.

Die Länge der Periode, ist die **Periodenlänge**.

Füllrate Für ein festen Modulus können wir uns vorstellen, dass wir mit geschickter Parameterwahl von a und c wir alle Zahlen von 0 bis $m-1$ erreichen könnten. Dementsprechend würde eine ungeschickte Wahl viel weniger Zahlen „treffen“.

In Anlehnung an das initiale Beispiel würde $c = m = 10$, $a = 1$ und $Z = 7$ die Folge

$$7, 7, 7, 7, 7, 7, 7, 7, 7, \ldots,$$

ad infinitum produzieren, welches offensichtlich *nicht* zufällig ist.

Das Kriterium eine möglichst hohe Variation an Zahlen zu Produzieren, messen wir mit der **Füllrate**. Diese ist definiert als die Anzahl der verschiedenen Zahlen innerhalb einer Periode, geteilt durch die Anzahl der Möglichen. Im initialem Beispiel wäre das daher

$$\frac{|\{7, 6, 9, 0\}|}{|\{0, 1, \ldots, 8, 9\}|} = \frac{4}{10} = 40\%$$

Obwohl noch weitere Eigenschaften betrachtet werden könnten, soll es ausreichend sein, sich auf diese zu beschränken. Besonders weil alle Eigenschaften notwendig, aber nicht ausreichend sind.

Bspw. wird der Generator mit $a = c = 1$, und beliebigem m und Z_0 eine Periode von m haben und eine Füllrate von 100%, ohne das die Folge sonderlich gut erscheint. Hier mit $Z_0 = 0$, $m = 5$:

$$0, 1, 2, 3, 4, 0, 1, 2, 3, 4, \ldots$$

Es wird daher notwendig sein über einzelne Kriterien hinaus Experimente und Testverfahren zu greifen. Diese werden prüfen welche Stärken oder Schwächen verschiedene Generatoren haben.

[1] Zur Vereinfachung wird unter Muster nur die *Wiederholung von Zahlenfolgen* verstanden, und nicht Komplexere Ideen wie „Summe von Nachfolgenden Elementen sind Konstant“, wann eine Zahl wieder aufkommt, oder anderen mathematischen Relationen.

2. Testverfahren für Zufallsgeneratoren

2.1. Rauschtest

Wir haben bereits im erstem Kapitel eine Art Testverfahren gesehen, als die Folge (1.2) betrachtet wurde. Da die Periode kurz war, war das Muster „7, 6, 9, 0“ leicht zu erkennen. Stellt man sich aber vor dass die Periodenlänge 20, 100, 10000 oder noch länger[1] wäre, würde es durch reine Betrachtung der Zahlen schwieriger werden ein muster zu erkennen.

Die menschlichen Sinne können dabei helfen dieses Problem zu überwinden. Dazu kann die generierte Folge in visueller oder akustischer Form aufbereitet. Ein Beispiel hierfür kann in Abbildung 2.1 gesehen werden.

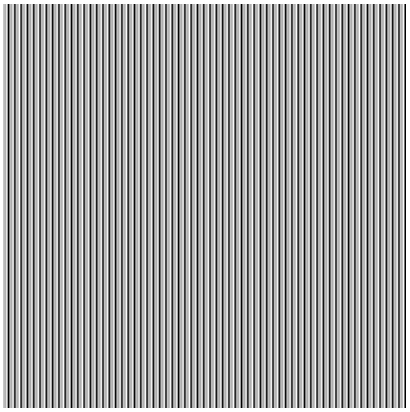


Abbildung 2.1.: Folge (1.2) als Bild dargestellt

Hier wurden die Werte auf Grautöne abgebildet, wo *größere* Zahlenwerte einem *hellerem* Pixelton zugeordnet werden, und *kleinere* einem *dunklerem*. Die Pixel sind dann Zeilenweise von links nach rechts angeordnet, mit Zeilenumbruch nach jeweils 256 Werten. Hier wurden insgesamt 256 Zeilen dargestellt. Wir nennen diesen Ansatz den **Rausch Test**.

Da das Beispiel nur eine Periode der Länge vier hatte, kann man das Muster auch weiterhin klar erkennen, anhand der langen, vertikalen Linien, welche das Bild überdecken.

Eine akustische Aufbereitung könnte die Folge in Frequenzen abbilden die dann nacheinander abgespielt werden würden. Im Beispiel (1.2) würde man einen konstanten Ton hören. Aus persönlichen Experimenten ist diese Methode jedoch nicht so effektiv.

Für ein zweites Beispiel, kann noch ein LCM Generator mit den Parametern

$$\begin{aligned} Z_0 &= 0 \\ a &= 1103515245 \\ c &= 12345 \\ m &= 2^{31} \end{aligned} \tag{2.1}$$

[1] Wie es in der Praxis meist gefordert wird. Selbst Perioden der Länge 2^{32} werden heute als „unzureichend“ angesehen.

betrachtet werden. Die Werte wurden aus der Glibc Implementierung der C Funktion `rand(3)` aus der Standardbibliothek genommen[3, Z. 364 ff.]. Hier wurden die Werte schlau Ausgewählt, um eine möglichst große Periodendauer zu erreichen.

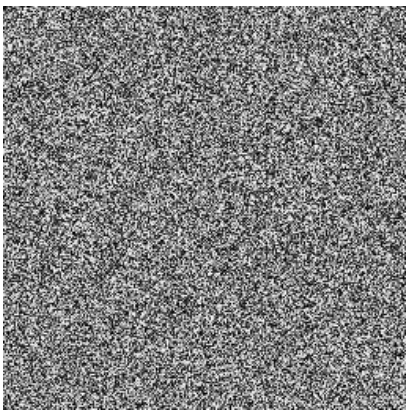


Abbildung 2.2.: Produkt des Generatoren (2.1) als Bild dargestellt

In Abbildung 2.2, ist auch gleich klar zu erkennen, dass sich die Qualität verbessert hat, und das nicht nur weil der Zahlenraum vergrößert wurde. Die Abwesenheit von offensichtlichen Mustern weist auch eine gewisse Ähnlichkeit zu dem physikalischem Effekt des *Weißen Rauschens*, wie es gelegentlich auf älteren Fernsehgeräten sehen kann.

Das akustische Analogon würde sich so anhören wie ein falsch eingestelltes Mittelwellen Radio, welches neben der Interferenz verschiedener Stationen auch sonstige (effektiv Zufälligen) elektromagnetischen Strahlungen hörbar macht.

2.2. Monte-Carlo-Test

Das nächste Testverfahren ist näher an praktischen Anwendungen[2], und ein bekanntest Beispiel die Kreiszahl π zu approximieren.

Hierbei werden immer Zahlenpaare generiert, uns als zweidimensionale Koordinaten interpretiert. Es wird dann jeweils geprüft ob dieses innerhalb des Einheitskreises liegt. Zur Vereinfachung wird nur der erste Quadrant betrachtet, und die Koordinatenwerte sind dann in $[0;1] \times [0;1]$. Dieses Experiment nennt man den **Monte-Carlo** Test, um π zu Approximieren.

Nach mehrfacher Wiederholung, kann aus der Proportion der Punkte innerhalb und insgesamt, eine Annäherung der Fläche des Quadranten bestimmt werden. Daraus wiederrum lässt sich eine Näherung für π herleiten.

Tabelle 2.1 fasst die Resultate der Ausführung eines Monte-Carlo Tests zusammen, mit zunehmender Versuchszahl. Hier ist wieder klar der Unterschied zwischen dem einfachem Generator (1.2)[3] und dem Glibc Generator zu sehen.

Bei dem Monte-Carlo Test kommt es auf *Gleichmäßigkeit* an, vor allem von gruppierten Werten. Hat ein Generator eine Neigung, einen *Bias*, beispielsweise für

[2] Neben der Approximation von Zahlen, hat es zahlreiche Anwendungen in der Physik, Chemie, Biologie, Künstlichen Intelligenz, usw.

[3] Der Fehler ist steets π, weil durch die Gruppierung der Koordinaten in $(7/9, 6/9)$ und $(9/9, 0)$, sind beide Koordinaten ≥ 1 ($\sqrt{7/9^2 + 6/9^2} \approx 1.024393$ und $\sqrt{1^2 + 0^2} = 1$)

	LCG (1.2)		LCG (2.1)	
Versuche	Treffer	$\approx$ Fehler	Treffer	$\approx$ Fehler
10	0	3.14159265359	7	0.3415926535
100	0	3.14159265359	79	0.0184073464
1000	0	3.14159265359	797	0.0464073464
10000	0	3.14159265359	7995	0.0124073461
100000	0	3.14159265359	78627	0.0034873461

Tabelle 2.1.: Vergleich von zwei Monte-Carlo Tests

kleine Werte, würden mehr Punkte *im* Kreis liegen. Dieses hätte die Folge, dass die Näherung von π zu groß werden würde.

Eine Falle bei diesem Experiment wäre, dass es möglich ist ein Generator zu bauen, dessen Zahlenfolge auf keinem Fall zufällig erscheinen, aber der trotzdem gut beim Monte Carlo Test abschneidet. Die Folge

$$\begin{array}{cccccc} 0,0, & 0,1, & 0,2, & 0,3, & \dots & 0,n, \\ 1,0, & 1,1, & 1,2, & 1,3, & \dots & 1,n, \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ n,0, & n,1, & n,2, & n,3, & \dots & n,n \end{array}$$

würde nach n^2 versuchen der Raum gleichmäßig „abgetasten“, und eine Approximation von π liefern, dessen Genauigkeit proportional ist zu n.

Ein werteres Problem ist, dass asymmetrische Verhalten entlang der Flächendiagonale nicht erkannt werden muss. Würden bspw. alle Punkte gleichmäßig oberhalb der Diagonale, wäre die Approximation nicht schlechter, obwohl das Ziel eines guten Generators verfehlt werden würde.

2.3. Spektral-Test

Die letzte hier betrachtete Methode untersucht das Verhältnis zwischen nacheinander Folgenden Zahlen. Bezeichnen wir die Folge der generierten Zahlen als Z, und Z_n als das n'te Folgenglied, werden beim **Spektral-Test**[1, S. 89][2, S. 93] Tupel der Form

$$(Z_n, Z_{n+1}, \dots, Z_{n+t})$$

für t Dimensionen betrachtet. Üblicherweise ist t gleich 2 oder 3. Also sind die Tupel

$$(Z_n, Z_{n+1}) \qquad \text{bzw.} \qquad (Z_n, Z_{n+1}, Z_{n+2}).$$

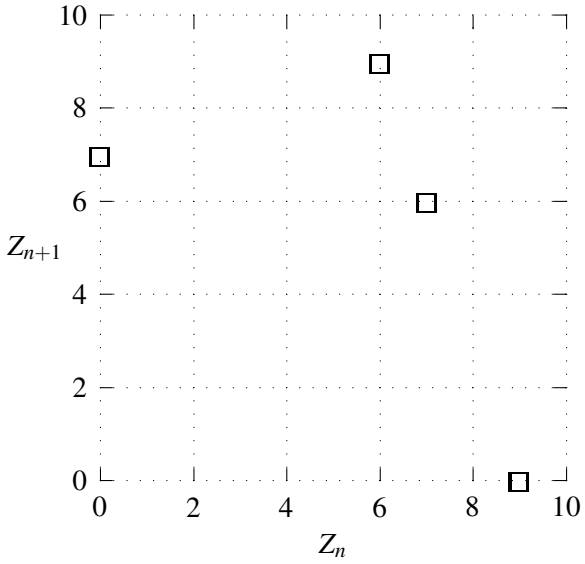


Abbildung 2.3.: 2d Spektral Test in Graphform für Gleichung (1.2)

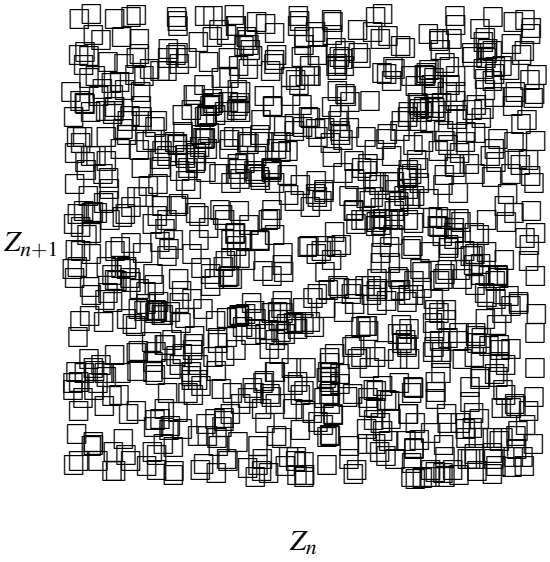


Abbildung 2.4.: 2d Spektral Test in Graphform für Gleichung (2.1), ohne Achsenbeschriftung

Diese Tupel können mit einem Diagramm visualisiert werden, in dem Tupel als Koordinaten aufgefasst werden, siehe Abbildung 2.3

3. Variationen auf der Linerar Kongruenten Methode

Die Beispiele aus Kapitel 2 haben gezeigt, dass ein *Linear Kongruenter Generator* durchaus in der Lage ist „ausreichende“ Zufälligkeit zu liefern. Trotzdem wird auf die Frage eingegangen, welche Verbesserungen machbar sind können, und welche Änderungen positive oder negative Folgen haben.

3.1. Fibonacci Methode

Erinnert man sich an die Definition der Linear Kongruenten Methode

$$Z_{n+1} = (aZ_n + c) \bmod m, \tag{1.1}$$

könnte man beobachten das sich dieses als eine Funktion definierten lässt. Sei

$$s(z) = (az + c) \bmod m \tag{3.1}$$

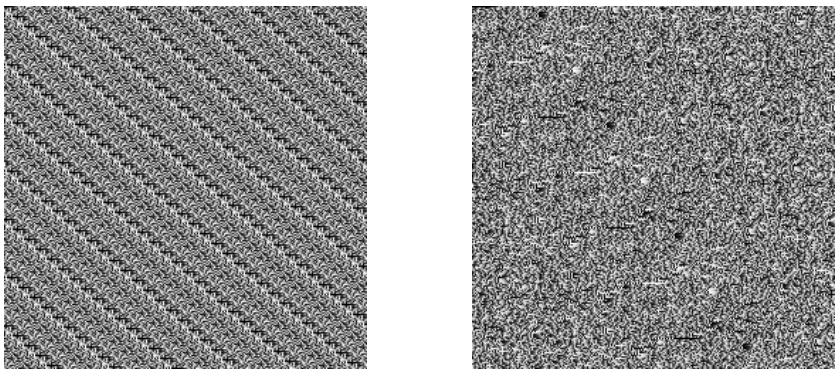
die „successor“ Funktion, bemerkt man, dass bei konstanten Werten für a, c und m die Ausgabe von s lediglich von Eingabe z abhängt. Weiter ist bekannt das wegen der Modulo Operation, höchstens m verschiedene Werte von 0 bis $m-1$ resultieren können. Dieses bedeutet das die Periode auch *höchstens m* sein kann, da sobald eine Zahl generiert wird, die bereits einmal aufgekommen ist wird sich die Sequenz ab diesem Punkt nur noch wiederholen.

Wenn aber die „successor“ Funktion nicht nur von einem Wert, sondern von zwei abhängen soll, bspw. den letzten zwei Eingaben, existiert die Möglichkeit die Periode auf bis zu m^2 zu erhöhen[1, S. 26][2, S. 27].

Wir betrachten dazu zunächst eine einfache Methode[1, S. 26][2, S. 27]

$$Z_{n+1} = (Z_n + Z_{n-1}) \bmod \tag{3.2}$$

mit $n \geq 2$ und $Z_0 = Z_1 = 1$, genannt **Fiboacchi Methode**.



(a) $m = 100$ (b) $m = 10007$

Abbildung 3.1.: Rauschtest für zwei Fiboacchi-Generatoren

3.1.1. Rausch Test

Die Ergebnisse des Rausch-Test in Bild-Form können in Abbildungen 3.1a und 3.1b gesehen werden. Dabei ist beachten wie mit einem kleinem Moduluswert von 100, starke Muster zu erkennen sind, wie auch zu erwarten ist für ein so kleinen Modulus.

Und obwohl Abbildung 3.1b wesentlich besser erscheint, teilt es ein paar problematische Eigenschaften mit dem vorherigem Beispiel. Würde man den Anfang betrachten, bemerkt man im oberem linkem Eck eine schwarze Linie die langsam heller wird — dieses deutet auf eine „Aufwärmphase“. Dieses kommt durch die Zahlenfolge

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, \ldots$$

zustande, bis diese m erstmals übersteigt. Zudem tritt eine Aufwärmphase jedes mal wieder ein, wenn zwei konstitutive 1'er produziert werden, was an den wiederholten schwarzen Linien erkannt werden kann, in beiden Bildern.

Ebenfalls interessant sind „Abwärmphasen“, also weiße Linien die langsam dunkler werden. Diese kommen auf, wenn zwei Zahlenwerte nahe m addiert genau unter $2m$ landen, und somit die Zahl, Modulo m, wieder nur etwas unter m liegt. Auf jeden Fall ein Muster, welches man nicht von eine zufälligen Sequenz zu sehen sein sollte.

3.1.2. Monte-Carlo-Test

Der nächste Versuch zeigt die praktischen Konsequenzen der schon betrachteten Muster. Tabelle 3.2 zeigt wie mehr Versuche nicht unbedingt zu besseren Ergebnissen führen müssen, und stattdessen der Fehler sich um ≈ 0.12 einpendelt, was schlechter ist als wo nur 100 Versuche durchgeführt wurden.

Teilweise Schuld daran ist die Tatsache, dass dieser Generator nur eine Periodenlänge von 300 hat. Daher kann sich das Ergebnis nach 300 Versuchen nicht verbessern. Dieses

Versuche	Treffer	≈ Fehler
10	7	0.34159265358
100	78	0.02159265358
1000	750	0.14159265358
10000	7534	0.12799265358
100000	75336	0.12815265358

Tabelle 3.1.: Monte-Carlo Tests für eine Fiboacchi Generator mit $m = 100$

deutet darauf hin, dass obwohl die Periodenlänge größer ist als was ein LCG mit gleichem Modulus bzw. Wortgröße erreichen kann (in diesem Fall 200), reicht dieses nicht aus für praktisch gute Ergebnisse[1].

3.1.3. Spektral Test

Wir schließen die Betrachtung dieser Methode ab, mit dem Spektral-Test, aber dieses mal in drei Dimensionen. Die Resultate sind in Abbildung 3.2 zu sehen. Besonders auffällig hierbei ist die Tatsache das der Graph in zwei „Ebenen“ geteilt wurde, die beide Parallel zu einander sind.

Ein Grund dafür ist, dass die Gleichung

$$Z_n < Z_{n+1} < Z_{n-1} \tag{3.3}$$

nie erfüllt wird[1, S. 33, Aufgabe 2][2, S. 36, ibid]. Dieses stellt wieder eindeutig ein Muster dar. Der Durch Fallunterscheidung nachvollzogen werden was der Grund dahinter ist:

$Z_n + Z_{n-1} < m$**,** Wenn die Summe *kleiner* ist als m, so wird $(Z_n + Z_{n-1}) \bmod m = Z_n + Z_{n-1} = Z_{n+1}$ gelten, was im Widerspruch zu Gleichung (3.3) steht, da dann $Z_n, Z_{n-1} < Z_{n+1}$ gilt.

$Z_n + Z_{n+1} \geq m$**,** Angenommen es gelte $Z_{n-1} = m - k$, für ein $0 < k < m$. Dann folgt

$$Z_{n+1} = (Z_n + Z_{n-1}) \bmod m = (Z_n + m - k) - m = Z_n - k,$$

was bedeutet das Z_{n+1} *echt* kleiner ist als Z_n, und somit auch hier Gleichung (3.3) verletzt.

Es stellt sich heraus, dass beide Fälle jeweils einer „Ebene“ in Abbildung 3.2 entsprechen.

[1] Zum Vergleich: Ein LCG mit $m = 100$ kann einen Fehler von bis zu ≈ 0.001264489 erreichen

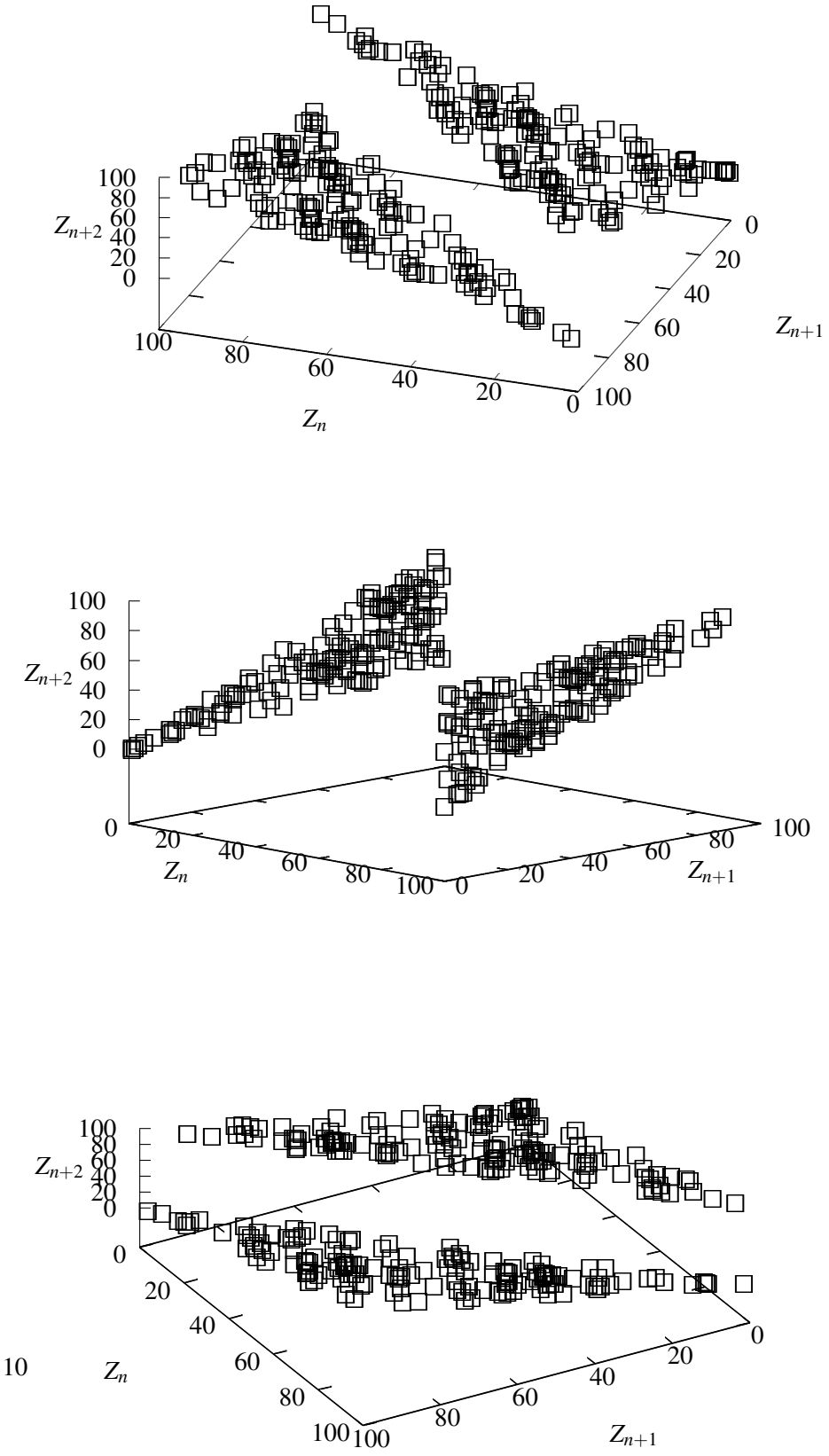


Abbildung 3.2.: Spektral Test in drei Dimensionen; Verschiedene Perspektiven

3.2. Mitchell-Moore Methode

Die schlechten Ergebnisse der Fiboacchi Methode sprechen jedoch bisher nicht gegen das Argument, welches am Anfang von Abschnitt 3.1 aufgestellt wurde, welches war, dass die Periode durch die Betrachtung von mehr als nur der letzten generierten Zahl verbessert werden kann.

Ein besseres Beispiel wäre die **Mitchell-Moore Methode**[1, S. 26][2, S. 27]

$$Z_n = (Z_{n-24} + Z_{n-55}) \bmod m, \qquad \text{für } n > 55 \tag{3.4}$$

wobei $Z_0, Z_1, \ldots, Z_{55}$ den Zufallssamen beschreiben, und beliebige Werte annehmen können. Bemerkenswert ist, dass nur ein Parameter m benötigt wird, was die Konstruktion eines solchen Generatoren einfacher macht.

Die Wahl der Zahlen 24 und 55, die *Lags* genannt werden, sind nicht willkürlich. Obwohl man die Theorie dahinter noch nicht vollkommen versteht, ist bekannt, dass für Lagwerte p und q, mit $p < q$, der Generator

$$Z_n = (Z_{n-p} + Z_{n-q}) \bmod m,$$

dann „gute“ Ergebnisse liefert wenn

$$x^p + x^q + 1$$

ein *Primitiver Polynom* ist, dh. die Nullstellen einen endlichen Körper bilden[1, S. 30][2, S. 32].

Zudem ist für ein Modulus der Form $m = 2^e$, für ein $e \in \mathbb{N}$, bekannt das die Periode die Länge $2^{e-1}(2^q - 1)$[1, S. 28][2, S. 28].

3.2.1. Rausch Test

In diesem und den folgenden Kapitelen wir ein Mitchell-Moore Generator benutzt, mit dem Zufallssamen

$$\begin{aligned} Z_0 &= 0 \\ Z_1 &= 3 \\ Z_2 &= 6 \\ &\vdots \\ Z_{55} &= 165, \end{aligned}$$

und $m = 503$ (prim).

Die Wahl des Zufallssamens ist für mit Absicht *nicht* zufällig, um zu demonstrieren das die Wahl unerheblich auf die Leistungsfäighkeit ist.

Und tatsächlich, wie Abbildung 3.3 zeigt, sind nach der initialen „Aufwärmphase“, keine deutlichen Muster zu erkennen. Dazu kommt noch, dass es keine weiteren Auf- oder Abwärmphasen gibt. Die Existenz einer „Aufwärmphase“ in diesem Fall, ist nur durch die absichtliche Regelmäßigkeit der Eingabe, welches beweisen soll, dass selbst vollkommen nicht-zufällige Zufallssamen anständige Ergebnisse liefern können.

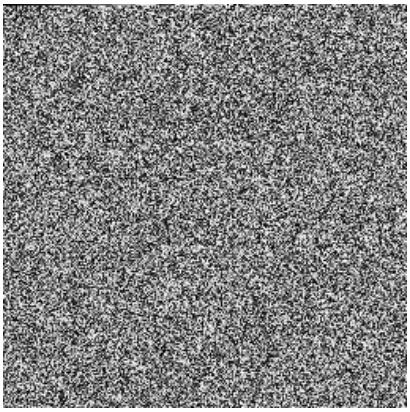


Abbildung 3.3.: Folge des beschriebenen Generators als Bild dargestellt

3.2.2. Monte-Carlo Test

Wie aus den Ergebnisse des *Rausch Test* zu erwarten, schneidet der *Mitchell-Moore* Generator auch gut beim Monte-Carlo Test ab:

Versuche	Treffer	≈ Fehler
10	8	0.05840734641020706
100	80	0.05840734641020706
1000	794	0.03440734641020704
10000	7851	0.0011926535897930357
100000	78465	0.0029926535897932816

Tabelle 3.2.: Monte-Carlo Tests für *Mitchell-Moore*

Es ist dabei zu erwähnen, dass der Generator „warm gelaufen“ wurde, d.h. die ersten 150 generierten Werte übersprungen wurden. Dieses wurde deswegen gemacht, da ansonsten die Experimente mit 10 oder 100 Runden ohne guten Grund schlechter sein würden als sie es wirklich sind. Stattdessen sehen wir schon mit wenigen versuchen gute und sich (meist) verbessernde Ergebnisse.

3.2.3. Spektral-Test

Im Spektral Test ist zum Schluss noch einmal die provozierte Aufwärmphase klar zu erkennen. Abbildung 3.4 zeigt die ersten 150 Werte, und die Abhänigkeiten zwischen Z_n und Z_{n+1} sind hier noch klar zu erkennen, obwohl schon „Tendenzen“ der Zufälligkeit schon zu erkennen sind.

Vergleicht man dieses jedoch mit Abbildung 3.5, dh. die nächsten 150 Werte erkennt man klar das die zuvor gesehenen Abhängigkeiten verschwunden sind.

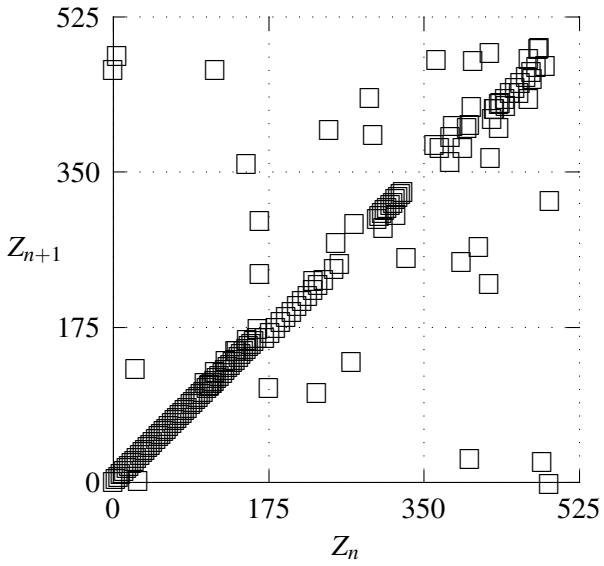


Abbildung 3.4.: Noch nicht Warmgelaufener *Mitchell-Moore* Generator aus Abschnitt 3.1.1 (150 Elemente)

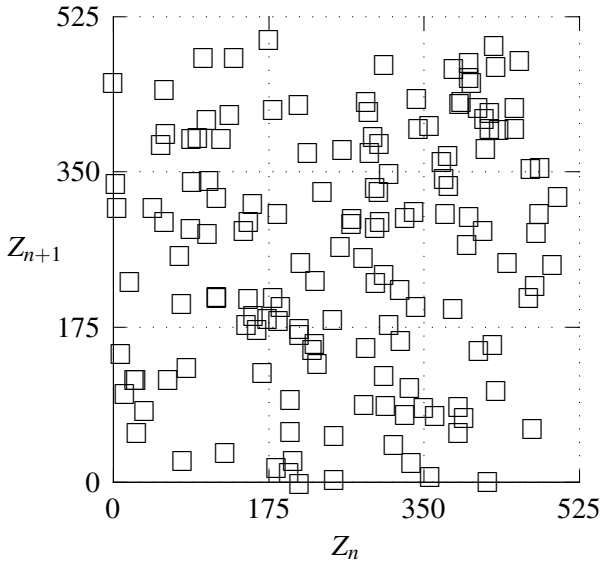


Abbildung 3.5.: Warmgelaufener *Mitchell-Moore* Generator aus Abschnitt 3.1.1 (150 Elemente)

Trotz allen Vorteilen dieser Methode, solle man zumindest insofern Vorsichtig sein, als dass es Anzeichen gab, dass in großen Monte-Carlo Experimenten Biase auftreten[2, S. 29], dh. nicht alle Zahlen praktisch gleich Wahrscheinlich auftreten, was die Problemstellung fordert.

3.3. Weiterführende Variationen

Das bisherige Kapitel hat sich betrankt auf die Analyse von *Lag*-Basierten Variationen auf der LCM. Im Sinne der Vollständigkeit soll hier noch eine kurze Aufzählung weiterer Ansätzen folgen:

Komplexere Terme Obwohl die Periode nicht über m verlängert wird (aber mit Glück über das von einem LCG), kann es sich lohnen den Term des Generatoren zu zu komplizieren, wie im Falle des **Quadradisch Kongruenten Methode**[1, S. 9][2, S. 10]

$$Z_{n+1} = (dZ_n^2 + aZ_n + c) \bmod m, \tag{3.5}$$

Es ist jedoch wichtig darauf hinzuweisen, dass nicht jede beliebige Änderung vom Vorteil sein muss, siehe [2, S. 26].

Alternative Operationen Anstatt sich nur auf die Addition von Termen zu beschränken, können auf bitweise Kontravalenz (`xor`)[1, S. 30f][2, S. 32], Multiplikation[2, S. 29], Subtraktion, usw. herangekommen werden.

In den meisten Fällen würde sich die Betrachtung hiervon aber nur aus geschwindigkeits-technischen Gründen lohnen, nicht zur Verbesserung der Methode *per se*.

Lineare Kombinationen Als Verallgemeinerung des *Lag* Konzepts, versteht sich die **Methode der Linearen Kombinationen**[1, S. 28][2, S. 29]

$$Z_n = (a_1 Z_{n-1} + a_2 Z_{n-2} + \ldots + a_k Z_{n-k} + c) \bmod m \tag{3.6}$$

für beliebiges k mit Parametern $a_1, \ldots, a_{n-k}, c$. Interessant ist, dass diese Methode alle bis zu diesem Abschnitt (ausgeschlossen) besprochenen Methoden auch beschreiben kann (Mitchell-Moore ist bspw. $a_{24} = a_{55} = 1, c = 0$ und alle anderen $a_i = 0$).

Dieses könnte auch fortgeführt werden für Methoden die Zufallsvektoren generieren, und dafür eine Parametermatrix A und ein Zuwachsvektor C brauchen würden. Hierbei würde de Zufallswert durch Matix-Vektor Multiplikation von A und M, und anschließende komponentenweise Addition mit C ermittelt werden.

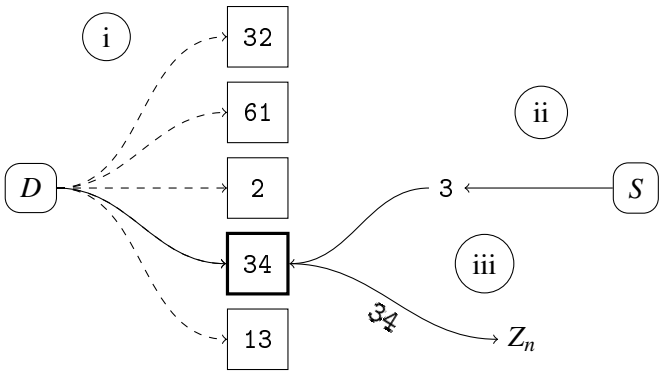


Abbildung 4.1.: Visualisierung von *Algorithm M*

4. Alternativen zu der Linerar Kongruenten Methode

Alle bisherigen Betrachtungen können als *Term-Basierte Methoden* klassifiziert werden. D.h. jede Methode war definiert im Sinne eine Funktion f mit k eingaben, und

$$Z_n = f(Z_{n-1}, Z_{n-2}, \ldots, Z_{n-k}) \tag{4.1}$$

Nun soll aber der Fokus auf Generatoren liegen, die bestehende Generatoren in sich Verbaut haben, und *benutzen*, um neue Werte zu generieren.

4.1. „Randomising by Shuffling“ (Algorithm M)

Für diesen Algorithmus[1, S. 32][2, S. 33] werden wir zwei Generatoren benutzen, wobei der erste, D (**d**ata), zum generieren der ausgegebenen Werte benutzt wird, und der zweite, S (**s**elect) zum „aussuchen“.

Betrachte man Abbildung 4.1 so sieht man drei wesentliche Schritte:

i. Fülle einen Vektor der Länge k mit k Werten aus D.

ii. Generiere eine Ganzzahl j zwischen 0 und k (ausgeschlossen) mittels S.

iii. Gebe aus dem Vektor an der stelle j zurück, und ersetze es danach mit einem neuem Wert aus D.

Für jeden neuen Wert, fange von ii. an.

Unter der Voraussetzung, dass die beiden Generatoren nicht zu ähnlich sind[1, S. 33, Aufgabe 3][2, S. 36, ibid], sollte der Algorithmus ermöglichen, dass die „Muster“ beim herkömmlichen Produzieren der Zahlenwerte von D durcheinander gebracht werden. Die Periode dieses Generatoren wird dann gleich der KGV der Periodenlänge von S und D sein[1, S. 33, Aufgabe 2][2, S. 36, ibid]. Aus diesem Grund kann dieser Ansatz als Verstärker angesehen werden.

Eine Einschränkung muss hierbei jedoch beachtet werden. Ruft man die Sequenz (1.2) wider in Gedächtnis, erinnert man sich das diese nur 4 verschiedene Zahlenwerte produziert, unabhängig davon wie viele Werte man versucht zu generieren. Dieses bedeutet das selbst mit einer *echten* Zufallsquelle als S, wird *Algorithm M* mit dem Generator aus Kapitel 2 nicht in der Lage sein die Füllrate zu verbessern.

Zur praktischen Performance reicht es auszusagen, dass sowohl der *Rausch Test* wie auch der *Spektral Test* keine erkennbaren Defizite aufweist. Daher beschränke ich mich nun auf eine Demonstration der „Verstärkungsfähigkeit“. Dazu wird als Generator D den Fibbonacci Generator benutzt wird, aus Abschnitt 3.1.2. Für S wird der Beispielgenerator für die Mitchell-Moore Methode aus Abschnitt 3.2.2 genommen.

Die Resultate sind in Tabelle 4.1 zu finden. Obwohl diese *an sich* nicht all zu erstaunlich sind, ist der Verstärkungseffekt klar zu erkennen, wenn dieses verglichen wird mit dem nicht „verstärkten“ Ergebnissen aus Tabelle 3.2.

Versuche	Treffer	$\approx$ Fehler
10	9	0.458407346410207
100	80	0.05840734641020706
1000	783	0.009592653589792999
10000	7810	0.017592653589793006
100000	78244	0.011832653589793019

Tabelle 4.1.: Monte-Carlo Tests für *Algorithm M*

Die beste Näherung würde beim herkömmlichen Fibbonacci Generator erreicht mit einem Fehler von 0.02159, für $n = 100$. Der Verstärkte hingegen schaff es bei $n = 1000$ bis zu 0.00959 zu erreichen.

Beim Vergleich von Abbildung 4.2 und Abbildung 4.3 kann man auch diese Zunahme der Leistung nachvollziehen. Durch *Algorithm M* werden wesentlich mehr Punkte erreicht, als es der Fibbonacci Generator selbst schaffen kann.

Bei initial genauerer Betrachtung mag man auch sagen das hier Tatsächlich die Füllrate gestiegen ist, was im Widerspruch stehen würde zu dem was oben gesagt wurde. Doch ist dieses eine „Illision“, basierend auf der Tatsache das zwei Werte immer gruppiert werden, zu einem neuem Wert, was bei Permutation der Reihenfolge, eine vielzahl an neuen Punkten ermöglicht (siehe „Akkumulierender Generator“, Abschnitt 4.2).

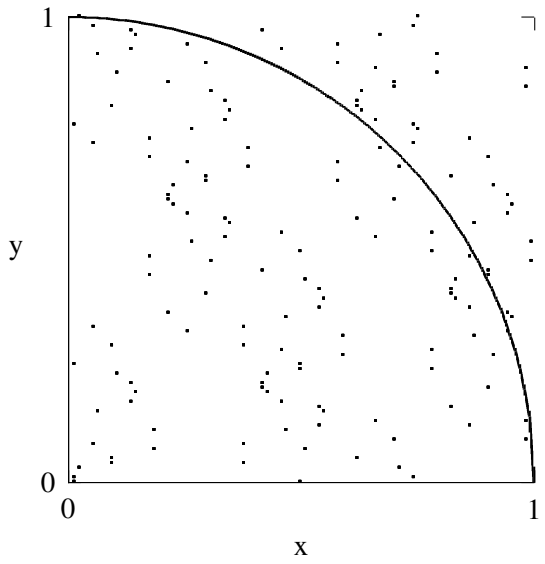


Abbildung 4.2.: Visualisierung von den Ergebnissen aus Tabelle 3.2.

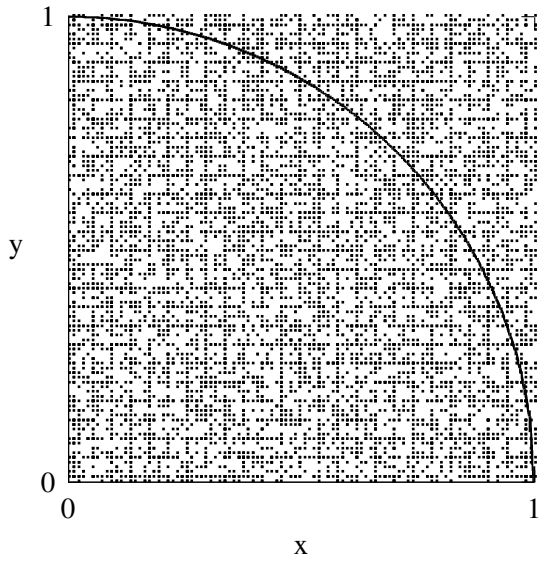


Abbildung 4.3.: *Algorithm M*, mit Generator aus Tabelle 3.2 als *D* Generator, und Beispiel aus Abschnitt 3.2 *S*.

4.2. Weiterführende Alternativen

Auch dieses mal werden noch Fortführungen oder ähnliche Ansätze noch besprochen. Eine kurze List dieser könnte enthalten:

***Algorithm B* Generator** In [1, S. 32][2, S. 34] wird neben dem oben besprochenem *Algorithmus M* auch die Möglichkeit erwähnt einen Generator zu verwenden, sowohl für Vektorfüllung wie auch –Auswahl. Dieses hat den Vorteil die Implementierung zu vereinfachen, und bei guten Ur-Generator auch besser zu verstärken.

„Überspringende" Generatoren Anstatt jeden Wert auszugeben, kann ein überspringender Generator „zwischenwerte" Verwerfen, nach entweder einem festem Konzept, wie beim sog. **Lüscher Generator**[2, S. 35], wo wiederholt 55 konsekutive Werte ausgegeben werden, und die nächsten 455 ignoriert, oder mittels eines zweiten Generatoren mit Prädikat, welcher bestimmt ob ein Wert der „nächste" ist oder nicht. Dieses hat Ähnlichkeiten zum Ansatz von *Algorithm M*, aber kostet dafür mehr Laufzeit.

Bit-Auswähler Beschränkt man seine Absicht darauf einzelne Bits zu generieren, kann eine Anpassung von *Algorithmus M/B* getätigt werden, wobei nicht Vektoren gefüllt werden sondern eine Zahl generiert, und ein Bit aus dieser Zahl als Ausgabe zufällig ausgewählt wird, dh. ein Index zwischen 0 und der Wortbreite der benutzen Zahl, ausgeschlossen.

Die generierte Zahl wird dann insgesamt verworfen, und in der nächsten Runde dementsprechend neu generiert[1]

Term Generatoren Wird ein Generator als „Variable" interpretiert„ beschreibt

$$Z_n = (X_n - Y_n) \bmod m$$

einen Generator, der sowohl „höherer Ordnung" ist, aber auch leichter zu implementieren/weniger Ressourcen braucht als *Algorithm M*[2, S.35].

Aggregierende Generatoren Wir können auch mehrere Werte generieren Lassen, und diese dann kombinieren. Ein einfacher Ansatz wie

$$Z_n = (X_n + X_{n-1} + \ldots + X_{n-k}) \bmod m$$

hätte Ähnlichkeiten zur *Methode der Linearen Kombinieren* aus Abschnitt 3.3.

[1] Es ergibt sich das dieser Ansatz viel bessere Ergebnisse liefert als, bspw. ein LCM mit $m = 2$. Aus diesem Grund kann es benutzt werden um eine Art *Aggregierenden Generator* zu bauen, wo ein Wert aus der „Bit-Konkatenation" mehere Zufälliger Bits aufgebaut wird. Dieses ist mit dem LCM/$m = 2$ Ansatz nicht so gut möglich[1, S. 530].

Eine schon gesehene Variation eines aggregierenden Generators war implizit enthalten im Monte-Carlo Test, wo zwei nachfolgenden Werte zu „einer Koordinate“ zusammenfasst wurden.

A. Literatur

[1] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Reading, Massachusetts: Addison-Wesley, 1981. ISBN: 0-201-03822-6.

[2] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Reading, Massachusetts: Addison-Wesley, 1998. ISBN: 0201896842.

[3] Free Software Foundation Inc. *Source code for `random_r.c`*. URL: `https://sourceware.org/git/?p=glibc.git;a=blob;f=stdlib/random%5C_r.c;h=3292713641b6b19ffaa9785b52764c62678ee0c7;hb=HEAD`.

[4] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Reading, Massachusetts: Addison-Wesley, 1969. ISBN: 0201038021.

[5] Donald E. Knuth. *The Art of Computer Programming: Errata to Volume 2 (3rd Edition)*. Reading, Massachusetts, Jan. 2011. URL: `https://cs.stanford.edu/~knuth/all2-pre.ps.gz`.

[6] Donald E. Knuth. *The Art of Computer Programming: Errata to Volume 2 (3rd Edition)*. after 2011. Reading, Massachusetts, Mai 2019. URL: `https://cs.stanford.edu/~knuth/err2.ps.gz`.