14 Pseudo-Zufallszahlen

14.1 Die folgenden Worte werden John von Neumann zugeschrieben:

> *Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.*

Trotzdem werden heute sehr oft Zahlen mit dem Rechner erzeugt, die „so aussehen, als ob“ sie zufällig wären. Die Erfahrungen und Erkenntnisse der letzten Jahre und Jahrzehnte zeigen allerdings, dass es sehr viel schwerer ist, „gute“ Pseudozufallszahlen[1] zu erzeugen als man vielleicht meint. Die folgenden Abschnitte sollen einen kleinen Einblick geben, damit man in der Praxis zumindest die gröbsten Anfängerfehler vermeiden kann.

14.2 Man findet das Zitat aus Punkt 14.1 zum Beispiel am Anfang von Kapitel 3 von TAOCP (Knuth 1998), das sich mit Zufallszahlen beschäftigt.

Einen guten Überblick gibt auch die Arbeit von L'Ecuyer (1998), deren Text auch unter `http://www.iro.umontreal.ca/~lecuyer/myftp/papers/handsim.ps` erhältlich ist. Vom gleichen Autor stammt auch `http://www.iro.umontreal.ca/~lecuyer/myftp/papers/tutaor.ps` (L'Ecuyer 1994).

Unter `https://web.archive.org/web/20181013022731/http://random.mat.sbg.ac.at/results/karl/server/` findet man unter anderem Informationen über eine ganze Reihe von in der Praxis verwendeten Zufallszahlengeneratoren.

14.3 Wegen der Probleme, die man sich mit (schlechten) Pseudozufallszahlen einhandeln kann, könnte man daran denken, statt dessen *echte* Zufallszahlen zu verwenden, die von einem physikalischen Prozess erzeugt werden.

Dieses Verfahren hat aber Nachteile. Wir kommen im Anschluss an die Aufzählung wünschenswerter Eigenschaften von Zufallszahlen noch einmal kurz darauf zu sprechen.

14.1 Allgemeines

14.4 An Pseudozufallszahlengeneratoren stellt man üblicherweise gewisse Anforderungen. Gewünschte Eigenschaften sind unter anderem:

- schnelle Erzeugbarkeit
- lange Periode
- Reproduzierbarkeit
- portable Implementierbarkeit
- schnelles Vorwärtsspringen
- „zufällige“ Zahlen

[1] Was auch immer das sein mag.

Vom letzten Punkt abgesehen werden die Forderungen auch alle von der zyklischen Folge $0, 1, \ldots, m-1, 0, 1, 2, \ldots, m-1, \ldots$ erfüllt. Die große Aufgabe besteht daher darin festzulegen, inwiefern die erzeugten Zahlen „zufällig“ sind; oder vielmehr „zufällig“ aussehen, denn erzeugen möchte man sie ja deterministisch.

Etwas konkreter möchte man von einfachen Generatoren verlangen, dass die Zufallszahlen zwei Eigenschaften haben:

- Die Zufallszahlen sollen (in einem gewissen Intervall, etwa $[0;1[$) gleichverteilt sein.
- Die Zufallszahlen sollen unabhängig voneinander sein.

Die erste Eigenschaft ist unter Umständen noch zu garantieren. Die zweite ist dagegen im allgemeinen verletzt (siehe Definition 14.6). Man ist zufrieden, wenn die erzeugten Zahlen „unabhängig aussehen“; genauer gesagt will man, dass man die statistischen Abhängigkeiten nicht mit einfachen (i. e. schnellen) Tests feststellen kann.

Wir werden in Abschnitt 14.3 auf diese schwierigen Aspekte zurückkommen. Insbesondere ist es so, dass diese oder weitere gewünschte Eigenschaften davon abhängen, in welchem Zusammenhang man die Pseudozufallszahlen verwenden will. Es gibt zum Beispiel Generatoren, die als sehr gut geeignet z. B. für Monte-Carlo-Simulationen gelten, aber für kryptographische Zwecke ungeeignet sind.

14.5 Man kann nun auch einige Nachteile nennen, die physikalische Prozesse zur Erzeugung von Zufallszahlen häufig haben:

- Sie sind nicht gleichverteilt.
- Sie sind nicht unabhängig.
- Sie sind nicht reproduzierbar.

14.6 Definition Ein (Pseudo-)Zufallszahlengenerator (RNG) ist eine Struktur $G = (S, s_0, T, U, G)$ mit folgenden Komponenten:

- einer endliche Menge S von *Zuständen*,
- einem *Anfangszustand* $s_0 \in S$,
- einer *Überführungsfunktion* $T : S \to S$,
- einer Menge U von *Ausgabewerten* und
- einer *Ausgabefunktion* $G : S \to U$.

Der Generator beginnt in Zustand s_0 und durchläuft nacheinander die Zustände $s_1, s_2, \ldots$, mit $s_{i+1} = T(s_i)$ für alle $i \geqslant 0$. In Zustand s_i wird die Ausgabe $u_i = G(s_i)$ produziert. ◇

14.7 Da die Zustandsmenge S eines Zufallszahlengenerators endlich ist, muss die Folge $s_0, s_1, s_2, \ldots$ und damit auch die Folge der Ausgaben zyklisch werden. Die Länge ρ des Zyklus heißt auch *Periodenlänge* des Generators.

14.8 Viele Generatoren sind von der folgenden Bauart: Zustandsmenge sind die nichtnegativen ganzen Zahlen kleiner einem $m \in \mathbb{N}$. Die Ausgabewerte sind die reellen Zahlen in dem Intervall $[0;1[$ und die Ausgabefunktion ist $u_i = s_i/m$.

Bei etwas aufwendigeren Generatoren besteht jeder Zustand aus $k > 1$ Zahlen $(x_i, x_{i-1}, \ldots, x_{i-k+1})$ kleiner m und die Ausgabefunktion ist z. B. $u_i = x_i/m$.

14.2 Generatoren für Pseudo-Zufallszahlen

14.2.1 Middle-square- und Muddle-square-Generator

14.9 Von Neumanns *Middle-square-Generator* sieht auf den ersten Blick zwar ganz nett aus, ist aber praktisch unbrauchbar.

Der Generator funktioniert so: Man arbeitet mit b-Bit Zahlen. Der Startwert ist x_0. Aus x_{n-1} ergibt sich x_n wie folgt: Man schreibt das Quadrat von x_{n-1} als $2b$-Bit Zahl. Daraus werden die mittleren b Bits extrahiert; sie bilden x_n.

Von der Verwendung dieses Generators wird dringend abgeraten, weil seine Periodenlänge zu klein ist.

14.10 Interessanterweise gibt es Modifikationen hiervon, die Knuth (1998) als *Muddle-square-Generator* bezeichnet. Er stammt von Blum, Blum und Shub bzw. eine Verallgemeinerung von Leonid Levin. Letztere ist wie folgt definiert:

Alle Zahlen sind r Bits lang. Es sei m Produkt zweier großer Primzahlen der Form $4\ell+3$. x_0 sei relativ prim zu m. Außerdem ist eine Zahl z als Maske gegeben. Sind die Dualzahldarstellungen $x = (a_{r-1}\cdots a_0)_2$ und $z = (b_{r-1}\cdots b_0)_2$ gegeben, so sei $x \cdot z = a_{r-1}b_{r-1} + \cdots + a_0 b_0$. Dann ist Levins Generator gegeben durch

$$\begin{aligned} x_{n+1} &= x_n^2 \pmod m \\ u_{n+1} &= x_{n+1} \cdot z \pmod 2 \end{aligned}$$

Man kann zeigen, dass der Generator bei zufälliger Wahl von x_0, m und z alle statistischen Test übersteht, die nicht mehr Zeit benötigen als das Faktorisieren natürlicher Zahlen.

Der Generator von Blum/Blum/Shub ist der Spezialfall von Levins Generator für $z = 1$.

In der Praxis scheinen diese Generatoren bislang keine Bedeutung zu spielen.

14.2.2 Lineare Kongruenzgeneratoren

14.11 Definition Ein *linearer Kongruenzgenerator* (engl. *linear congruence generator*, LCG) ist gegeben durch einen Anfangswert x_0, Modulus m, Multiplikator a und Inkrement c:

$$x_n = ax_{n-1} + c \pmod m \tag{14.1}$$

Ein LCG heißt *multiplikativ* oder ein MLCG, wenn $c = 0$ ist. ◇

14.12 Satz. *Der LCG aus Gleichung 14.1 hat genau dann volle Periodenlänge, wenn gilt:*

- $\mathrm{ggt}(m, c) = 1$,
- q *prim und* $q|m \implies q|a-1$ *und*
- $4|m \implies 4|a-1$.

14.13 Ein schlechter Generator war RANDU, der lange im IBM/360 Betriebssystem benutzt wurde. Bei ihm ist $a = 65539$, $c = 0$ und $m = 2^{31}$. Der Generator hat Periodenlänge 2^{29} und eine schlechte Gitterstruktur (siehe Abschnitt 14.3).

14.14 Der ANSI-C Generator `rand` ist der LCG mit $m = 2^{31}$, $a = 1\,103\,515\,245$, $c = 12345$ und $x_0 = 12345$. Dieser Generator ist schlecht. Die niedrigwertigen Bits der erzeugten Zahlen sind „nicht sehr zufällig“.

Der Generator `drand48` von Solaris ist ebenfalls ein LCG, und zwar der mit Parametern $a = 25\,214\,903\,917$, $c = 11$ und $m = 2^{48}$. Er hat volle Periodenlänge. Er ist besser als `rand`, aber auch er besteht manche statistische Tests nicht.

14.15 Ein besserer LCG ist der mit $m = 2^{32}$, $a = 69069$ und $c = 1$ von Marsaglia.

14.2.3 Mehrfach rekursive Generatoren

14.16 Definition Ein *mehrfach rekursiver Generator* (MRG) eine Verallgemeinerung eines MLCG. Hier ist

$$x_n = a_1 x_{n-1} + \cdots + a_k x_{n-k+1} \pmod{m} \tag{14.2}$$

wobei die Arithmetik modulo m durchgeführt werde und die Konstanten $a_1, \ldots, a_k$ aus dem Bereich $\{-(m-1), \ldots, m-1\}$ stammen. ◇

14.17 Definition Spezialfälle der MRG sind die sogenannten *lagged Fibonacci generators* (LFG). Der Name rührt von der einfachen Version

$$x_n = x_{n-1} + x_{n-2} \pmod{m}$$

her, die aber zu schlecht ist. Statt dessen empfiehlt sich

$$x_n = x_{n-r} + x_{n-s} \pmod{m}$$

für geeignete Zahlen r und s. Außerdem kann man statt der Addition auch andere Operationen in Betracht ziehen. ◇

14.18 Exklusives Oder in LFG hat sich als schlecht herausgestellt. Lange Zeit galt $(r, s) = (24, 55)$ und Addition als gute Wahl. Für $m = 2^e$ hat der Generator dann Periodenlänge $2^{e-1}(2^{55} - 1)$. Allerdings sind z. B. die niedrigstwertigen Bits der x_i nicht „gut“ verteilt. Marsaglia (1985) schlägt einige Varianten vor.

14.19 L'Ecuyer (1998) schlägt den folgenden MRG vor:

$$\begin{aligned} x_n \;=\; & 2\,620\,007\,610\,006\,878\,699\, x_{n-1} + \\ & 4\,374\,377\,652\,968\,432\,818\, x_{n-2} + \\ & 667\,476\,516\,358\,487\,852\, x_{n-3} \qquad \bmod (2^{31}-1)(2^{31}-2000169) \end{aligned}$$

14.20 Satz. *Ein MRG hat maximale Periodenlänge* $\rho = m^k - 1$, *falls das sogenannte* charakteristische Polynom $P(z) = z^k - a_1 z^{k-1} - \cdots - a_k$ *primitives Polynom des Körpers* $\mathbb{Z}_m$ *ist.*

14.2.4 Inverse Kongruenzgeneratoren

14.21 Definition Ein *inverser Kongruenzgenerator* (ICG, Eichenauer und Lehn 1986) ist durch die Gleichung

$$x_{n+1} = (ax_n^{-1} + c) \pmod p$$

festgelegt, wobei p eine Primzahl sei. Da ein $x_i = 0$ sein kann, werde festgelegt: $0^{-1} = 0$. ◇

14.22 Diese Generatoren haben zum einen bemerkenswert gute theoretische Eigenschaften. Zum anderen bestehen aber auch empirische Tests mit sehr guten Ergebnissen. Ein kleiner Nachteil von ICG ist die nicht ganz so einfache Implementierung (Geschwindigkeit des Generators?).

Konkrete Parameter für gute ICG findet man im WWW unter https://web.archive.org/web/20180818144643/http://random.mat.sbg.ac.at/generators/wsc95/inversive/.

14.2.5 Mersenne-Twister

14.23 Definition Der *Mersenne-Twister* stammt von Matsumoto und Nishimura (1998))(Artikel verfügbar unter http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/mt.pdf) und ist wie folgt definiert. Alle Zahlen sind Bitvektoren der Länge w und r ist eine Zahl mit $0 \leqslant r \leqslant w-1$. Es sind zwei Zahlen k und m mit $1 \leqslant m < k$ festgelegt und eine $w \times w$-Matrix A ist geeignet gewählt.

Es bezeichne x_{n+1}^l die „unteren“ (lower) r Bits von x_{n+1} und x_n^u die „oberen“ (upper) $w-r$ Bits von x_n. Außerdem bezeichne $x_n^u | x_{n+1}^l$ die Konkatenation der beiden Bitfolgen. Dann ist

$$x_{n+k} = x_{n+m} \oplus (x_n^u | x_{n+1}^l)A \,. \qquad \diamond$$

14.24 Eine konkrete Implementierung MT19937 eines Mersenne-Twisters hat Periodenlänge $2^{19937}-1$ und auch ausgezeichnete Eigenschaften hinsichtlich statistischer Tests.

Weitere Informationen einschließlich der Implementierungen in verschiedenen Programmiersprachen findet man über die Seite http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html im WWW.

14.25 Aber auch beim Mersenne-Twister gibt es noch Raum für Verbesserungen. Man lese etwa den Aufsatz von Panneton, L'Ecyuer und Matsumoto (2006). Dort werden Generatoren (WELL) vorgeschlagen, die zum Teil noch längere Periode, und dabei aber noch bessere Eigenschaften als der Mersenne-Twister haben. Implementierungen findet man z. B. unter http://www.iro.umontreal.ca/~panneton/WELLRNG.html.

14.2.6 Zellularautomaten als Generatoren

Manche Zellularautomaten zeigen, ausgehend selbst von „einfachen“ Anfangskonfigurationen, „zufälliges“ Verhalten. Zum Beispiel wird in Mathematica der folgende besonders einfache Generator benutzt:

14.26 DEFINITION Gegeben ist ein Feld von k Bits $x_0, x_1, \ldots, x_{k-1}$. Die sogenannte *Regel 30* besagt, dass in einem Schritt gleichzeitig neue Werte $y_0, \ldots, y_{k-1}$ nach der folgenden Vorschrift berechnet werden:

x_{i-1}	1	0	1	0	1	0	1	0
x_i	1	1	0	0	1	1	0	0
x_{i+1}	1	1	1	1	0	0	0	0
y_i	0	0	0	1	1	1	1	0

Dabei sind die Indexrechnungen $i-1$ und $i+1$ modulo k zu verstehen. Die so berechneten y_i bilden die Werte x_i für den nächsten Schritt.

Der Name „Regel 30“ rührt daher, dass die Folge 00011110 der Bits für y_i die Dualzahldarstellung der Zahl 30 ist. ◇

14.27 Wenn man k groß genug wählt (in Mathematica einige Hundert), in der Anfangskonfiguration mindestens eine 1 hat, zunächst eine hinreichende große Zahl Schritten zur Initialisierung durchläuft, und dann in einer willkürlich aber fest gewählten Zelle die Folge der durchlaufenen Werte benutzt, erhält man einen Generator, der z. B. alle von Marsaglia's `diehard`-Tests besteht (Schloissnig 2003).

14.2.7 Kombination mehrerer Generatoren

Es ist eine recht naheliegende Idee, durch geeignete „Kombination“ zweier oder mehrerer Generatoren neue zu erhalten, die ein „besseres“ Verhalten aufweisen, als ihre Komponenten.

Zum Beispiel ist in vielen Fällen leicht einzusehen, dass die Periodenlänge größer wird. Manchmal zeigen theoretische Überlegungen oder empirische Tests, dass sich auch das statistische Verhalten verbessert.

Zwei Kombinationsverfahren werden besonders gerne verwendet.

14.28 Beim sogenannten *Shuffling* werden zwei Generatoren für Zahlen x_i und y_i benutzt. Man unterhält eine Tabelle mit einer gewissen Anzahl zuletzt erzeugter x_i. Wann immer ein weiterer Wert benötigt wird, benutzt man das nächste y_j, um zufällig einen Wert aus der Tabelle auszuwählen. Der entsprechende Platz wird außerdem mit dem nächsten neuen x_i gefüllt.

Diese Methode hat zwei Nachteile. Zum einen ist sie theoretisch nicht gut verstanden, zum anderen ist es unbekannt, wie man schnell viele Zufallswerte überspringen kann.

14.29 Die andere Klasse von Methoden besteht darin, aus zwei Zufallsfolgen $x_0, x_1, \ldots$ und $y_0, y_1, \ldots$ mittels einer Operation $\bullet$ eine neue Folge $x_0 \bullet y_0, x_1 \bullet y_1, \ldots$ zu erzeugen. Die Operation kann Addition, bitweises Exklusives Oder, usw. sein.

Auch bei dieser Vorgehensweise vergrößert sich (bei „vernünftiger“ Vorgehensweise) die Periodenlänge. Außerdem wird die Uniformität der Verteilung besser (oder jedenfalls nicht schlechter) und die Unabhängigkeit aufeinanderfolgender Werte größer (siehe z. B. Marsaglia 1985).

14.3 Tests für Pseudo-Zufallszahlen

Wie stellt man fest, ob ein Pseudozufallszahlengenerator „gut“ ist? Was heißt überhaupt „gut“?

Um das zu präzisieren, hat es sich eingebürgert, Generatoren verschiedenen Tests zu unterziehen.

14.30 Ergebnis solcher Tests sind üblicherweise Zahlen (etwa zwischen 0 und 1), die in naheliegender Weise als Qualitätsangabe interpretiert werden können. Manchmal ist es möglich, solche Zahlen auf Grund der Definition des Generators mathematisch herzuleiten. Man spricht dann von einem *theoretischen Test*. In anderen Fällen muss man *empirische Test* durchführen, das heißt den Generator zur Erzeugung vieler Pseudozufallszahlen verwenden und die konkreten Ergebnisse untersuchen.

14.3.1 Grundsätzliches: χ^2- und KS-Test

14.31 Definition Es seien $s_1, \dots, s_k$ die möglichen Ergebnisse eines Zufallsexperiments, die mit Wahrscheinlichkeiten $p_1, \dots, p_k$ auftreten. Für eine Reihe von n *unabhängigen* Zufallsexperimenten bezeichne Y_i die absolute Häufigkeit, mit der Ergebnis s_i auftrat. Es ist also $\sum_i Y_i = n$ und $\mathbf{E}[Y_i] = np_i$.

Dann bezeichnet man

$$V = \sum_{i=1}^{k} \frac{(Y_i - \mathbf{E}[Y_i])^2}{\mathbf{E}[Y_i]}$$

als χ^2-Statistik der beobachteten Größen $Y_1, \dots, Y_k$. ◇

14.32 Setzt man in die obige Formel $\mathbf{E}[Y_i] = np_i$ ein, so erhält man

$$V = \sum_{i=1}^{k} \frac{(Y_i - np_i)^2}{np_i} = \sum_{i=1}^{k} \frac{Y_i^2 - 2Y_i np_i + n^2 p_i^2}{np_i} = \frac{1}{n} \sum_{i=1}^{k} \frac{Y_i^2}{p_i} - 2n + n = \frac{1}{n} \sum_{i=1}^{k} \frac{Y_i^2}{p_i} - n \,.$$

14.33 Es stellt sich nun die Frage, wie die Größe V verteilt ist. Hierfür gibt es Tabellen mit Näherungen. Sie enthalten für jeden Wert $\nu = k - 1$, die sogenannte *Anzahl der Freiheitsgrade*, eine Zeile und für einige Wahrscheinlichkeitswerte p, z. B. $p = 1\%$, $p = 5\%$, …, $p = 99\%$ eine Spalte. Man beachte, dass die Anzahl n nicht eingeht; sie muss groß genug gewählt sein. Eine Daumenregel besagt, dass n so groß sein muss, dass jedes $s_i \geqslant 5$ ist.

In einem solchen Fall besagt ein Wert x in Zeile $\nu = k - 1$ und Spalte p: Der Wert V ist kleiner oder gleich x mit Wahrscheinlichkeit p (falls n groß genug ist).

14.34 Beispiel. Zum Beispiel findet man in einer Tabelle für χ^2:

	$p = 0.01$	$p = 0.05$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 0.95$	$p = 0.99$
$\nu = 11$	3.053	4.575	7.584	10.34	13.70	19.68	24.72

Ist etwa $k = 12$ und hat man etwa für eine Reihe von Versuchen einen Wert $V = 29.44$ errechnet, dann liest man aus der Tabelle ab: In 99% der Fälle ist $V \leqslant 24.72$, also ist $V > 24.72$ und erst recht $V > 29.44$ in höchsten einem Prozent der Fälle. Man wird es also als sehr unwahrscheinlich ansehen, dass die Versuchsreihe der angenommenen Verteilung genügt.

Andererseits betrachte man den Fall, dass die Y_i alle sehr sehr gut den Erwartungswerten entsprechen. Dann ist also V klein, etwa $V = 1.17$. Der Tabelle entnimmt man, dass auch das in weniger als einem Prozent der Fälle passiert.

Hat man es nicht mit diskreten, sondern mit kontinuierlich verteilten Werten zu tun, dann hilft der KS-Test.

14.35 Definition Der *Kolmogorov-Smirnov-Test* (kurz KS-Test) wird wie folgt durchgeführt. Gegeben sind n *unabhängige* Zufallsexperimente mit Ergebnissen $X_1, \ldots, X_n$. Diese induzieren eine empirische Verteilungsfunktion

$$F_n(x) = \frac{|\{i \mid X_i \leqslant x\}|}{n} .$$

Zum Vergleich mit einer vorgegebenen kontinuierlichen Verteilungsfunktion $F(x)$ berechnet man die Größen

$$\begin{aligned} K_n^+ &= \sqrt{n} \max_{-\infty<x<+\infty} (F_n(x) - F(x)) \qquad \text{und} \\ K_n^- &= \sqrt{n} \max_{-\infty<x<+\infty} (F(x) - F_n(x)) \end{aligned}$$

Für diese Größen kann man wieder in einer Tabelle mit Zeilen für verschiedene n und Spalten für verschiedene p nachschlagen, mit welcher Wahrscheinlichkeit ein bestimmter K-Wert kleiner oder gleich (bzw. größer) einem bestimmten Wert ist. ◇

14.36 Eine mögliche Anwendung des KS-Test ist die folgende. Sinnvollerweise macht einen χ^2-Test nicht nur einmal, sondern mehrmals. Da die Verteilung für χ^2 (jedenfalls näherungsweise) bekannt sind, kann man auf die sich ergebenden Wahrscheinlichkeiten einen KS-Test anwenden.

Analog kennt man (näherungsweise für große n) die Verteilung der K_n und kann auf diese Werte einen weiteren KS-Test anwenden.

14.3.2 Einfache empirische Tests

Im Folgenden gehen wir auf einige empirische Test für Zufallszahlengeneratoren ein. Zu den Zielen solcher Tests gehört es, Informationen über die vermutliche Art der Verteilung zu bekommen, die Un-/Abhängigkeit der erzeugten Zahlen, oder etwa eventuelle Korrelationen.

Es gibt frei verfügbare Programmpakete, die Zufallszahlengeneratoren einer Reihe von Tests unterziehen. Zum Beispiel ist unter https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/ die Sammlung *Diehard* von George Marsaglia zu finden. Dabei handelt es sich „schwer“ zu bestehende Tests (siehe Marsaglia 1985). Weitere Sammlungen finden sich bei Richard Simard unter http://simul.iro.umontreal.ca/testu01/tu01.html und beim NIST unter https://web.archive.org/web/20130304154702/http://csrc.nist.gov/groups/ST/toolkit/rng/index.html. TestU01 enthält auch die Implementierungen von fast zweihundert Pseudozufallszahlengeneratoren, von denen L'Ecuyer und Simard (2007) sagen: „some are good and many are bad“.

14.37 Im Folgenden gehen wir davon aus, dass eine Folge $U_0, U_1, \ldots$ von reellen Zahlen vorliegt, die angeblich unabhängig und gleichverteilt sind zwischen 0 und 1.

Werden für einen Test nichtnegative ganze Zahlen benötigt, so werde statt dessen die Folge der $Y_i = \lfloor dU_i \rfloor$ für eine geeignete Konstante $d \in \mathbb{N}$ betrachtet.

14.38 Der einfachste Test besteht darin, zu versuchen festzustellen, ob unter den Y_i jeder der möglichen Werte $0, \ldots, d-1$ gleich oft vorkommt.

14.39 Die analoge Eigenschaft kann man für Paare, Tripel, etc. prüfen. Man beachte, dass man dann die s-dimensionalen *nicht überlappenden* Vektoren $\mathbf{Y}_i = (Y_{si}, Y_{si+1}, \dots Y_{si+s-1})$ benutzen muss und *nicht* die überlappenden Vektoren $(Y_i, Y_{i+1}, \dots Y_{i+s-1})$.

Knuth 1998 nennt dies den *Serial Test*.

14.40 Eine Variante dieses Tests für reelle Pseudozufallszahlen (L'Ecuyer 1998) besteht darin, den s-dimensionalen Einheitswürfel in $k = 2^{\ell s}$ gleich große Teilwürfel aufzuteilen und bei n zufällig gewählten Punkten für jeden Würfel j die Anzahl X_j der in ihm liegenden Punkte zu bestimmen. Für die Größe

$$\sum_{j=1}^{k} \frac{(X_j - n/k)^2}{n/k}$$

weiß man, dass ihr Erwartungswert $k-1$ und ihre Varianz $2(k-1)(n-1)/n$ sind, wenn die Punkte wirklich zufällig gleichverteilt liegen. Gegen diese Hypothese kann man die empirischen Daten testen.

14.41 Für den *Lückentest* (engl. *gap test*) sind zwei relle Zahlen $0 \leqslant \alpha < \beta \leqslant 1$ gegeben und man betrachtet die Längen der maximalen Teilfolgen $U_j, \dots, U_{j+r}$, so dass $\alpha \leqslant U_{j+r} \leqslant \beta$ aber die vorherigen Werte nicht. Man spricht dann von einer Lücke der Länge r.

Für $p = \beta - \alpha$ ist $p_r = p(1-p)^r$ die Wahrscheinlichkeit für eine Lücke der Länge r. Man kann daher leicht den χ^2-Test anwenden.

14.42 Beim *Permutationstest* betrachtet man nicht überlappende Vektoren $\mathbf{U}_i = (U_{si}, U_{si+1}, \dots U_{si+s-1})$ der Länge s. Für jeden Vektor wird die Permutation π der Indizes bestimmt, die die Komponenten sortiert. Es wird eine Statistik erstellt, welche Permutation wie häufig auftritt, und getestet, inwieweit das mit der theoretisch gegebenen Gleichverteilung verträglich ist.

14.43 Beim *Maximum-von-t Test* werden die Größen $V_i = \max\{U_{si}, U_{si+1}, \dots U_{si+s-1}\}$ berechnet. Falls die Pseudozufallszahlen wirklich gleichverteilt sind, ist

$$\mathbf{P}(V_i \leqslant x) = \mathbf{P}(\max\{U_{si}, U_{si+1}, \dots U_{si+s-1}\} \leqslant x) = \prod \mathbf{P}\big(U_{si+j} \leqslant x\big) = x^s \ .$$

Diese Hypothese kann mit einem KS-Test überprüft werden.

14.44 Der *Geburtstagsabstandstest* (engl. *birthday spacings test*) von Marsaglia arbeitet wie folgt: Gegebene ganze Pseudozufallszahlen $Y_1, Y_2, \dots, Y_m$ aus dem Bereich $\{1, \dots, n\}$ werden zunächst sortiert. Es bezeichne $Z_1, Z_2, \dots, Z_m$ die aufsteigend sortierte Reihenfolge der gleichen Zahlen. Hieraus werden die Abstände $S_i = Z_{i+1} - Z_i$ berechnet. Es bezeichne $R = \big|\{S_i \mid \exists j < i : S_j = S_i\}\big|$ die Anzahl der mehrfach vorkommenden Abstände. Man kann zeigen, dass R näherungsweise Poisson-verteilt ist mit Parameter $\lambda = m^3/(4n)$.

Man kann nun (bei geeignetem d) für die diskreten Ereignisse $R = 0$, $R = 1$, …, $R = d-1$ und $R \geqslant d$ die Wahrscheinlichkeiten ausrechnen, hinreichend oft in einem Experiment einen konkreten Wert R bestimmen und für die Ergebnisse einen χ^2-Test machen.

Lagged-Fibonacci-Generatoren haben üblicherweise Probleme mit dem Geburtstagsabstandstest, auch wenn sie viele andere Test bestehen (das gilt z. B. auch für den schon in Punkt 14.18 erwähnten Generator $x_n = x_{n-24} + x_{n-55} \pmod{2^e}$.

14.3.3 Weitere Tests

14.45 Für die Erklärung des Spektraltests betrachten wir zunächst den (zugegebenermaßen besonders) schlechten Pseudozufallszahlengenerator, der durch $x_{n+1} = 85x_n + 2 \pmod{256}$ und die Ausgabefunktion $u_n = x_n/256$ gegeben ist. Für mehr als 256 überlappende Paare (u_i, u_{i+1}) ihn sind in Abbildung 14.1 die entsprechenden Punkte im Einheitsquadrat eingezeichnet. Wie man sieht,

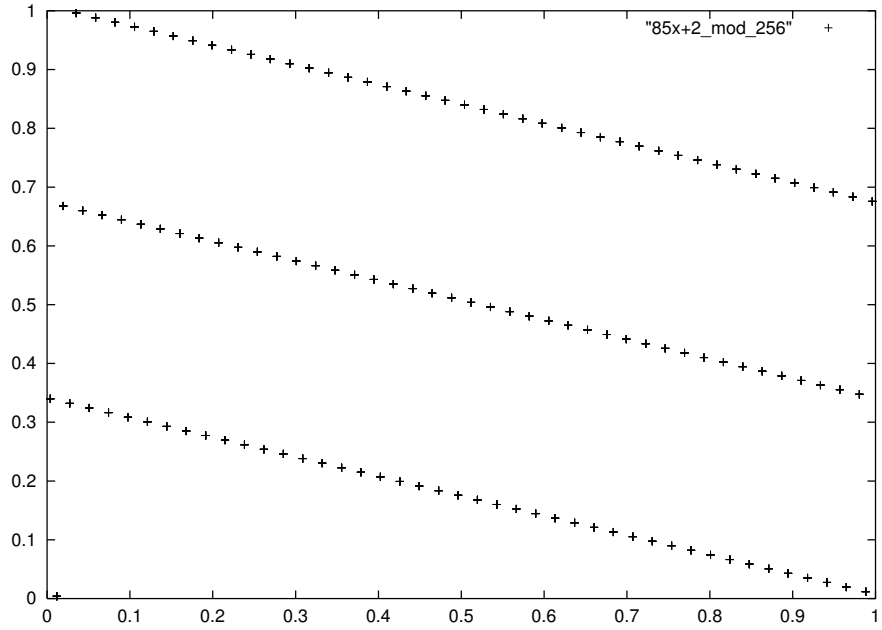

Abbildung 14.1: Gitterstruktur bei einem sehr schlechten Pseudozufallszahlengenerator.

sind die Punkte überhaupt nicht gleichmäßig und schon gar nicht „zufällig“ verteilt.

Die zu beobachtende *Gitterstruktur* tritt bei linearen Kongruenzgeneratoren aber auch bei einigen anderen Generatorarten auf.

14.46 Der *Spektraltest* für Dimension s besteht darin, für s-dimensionale überlappende Vektoren $\mathbf{u}_i = (u_i, u_{i+1}, \ldots, u_{i+s-1})$ im Einheitshyperwürfel die Zahl d_s zu ermitteln. Dabei ist d_s der maximale Abstand zwischen zwei Hyperebenen, genommen über alle Familien paralleler Hyperebenen im Einheitswürfel, die alle Punkte $\mathbf{u}_i$ beinhalten.

Je kleiner ein d_s ist, als desto „besser“ wird man den verwendeten Generator (hinsichtlich Dimension s) ansehen.

14.47 Ein anderes „Qualitätsmaß“ für Pseudozufallszahlengeneratoren ist die *Diskrepanz* (und als Sonderfall die *Sterndiskrepanz* (engl. *star discrepancy*)) (siehe auch Niederreiter 1992).

Hierfür betrachtet man wieder die Vektoren $\mathbf{u}_i = (u_i, u_{i+1}, \ldots, u_{i+s-1})$ im Einheitshyperwürfel. Es seien N solche Punkte gegeben. Für jede Menge $R = \prod_{j=1}^{s} [\alpha_j, \beta_j[$ mit $0 \leqslant \alpha_j < \beta_j \leqslant 1$ sei $I(R)$ die Anzahl der $\mathbf{u}_i$, die in R liegen, und $V(R) = \prod_{j=1}^{s} (\beta_j - \alpha_j)$ das Volumen von R.

Die s-dimensionale Diskrepanz $D_N^{(s)}$ der Punkte $\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}$ ist

$$D_N^{(s)} = \max_R |V(R) - I(R)/N| \,.$$

Falls man sich auf Mengen R einschränkt, deren eine Ecke der Nullpunkt ist, falls also $\alpha_j = 0$ für alle j ist, dann spricht man von der Sterndiskrepanz $D_N^{*(s)}$.

Für Punktemengen, deren Verteilung „weit“ von Gleichverteilung entfernt ist, ergeben sich zu große Werte bei der Diskrepanz, und wenn die Verteilung „zu gleichmäßig“ ist, zu kleine Werte.

14.48 Beim *Näheste Paare Test* (engl. *nearest pair test*) erzeugt man zufällig n Punkte im s-dimensionalen Einheitshyperwürfel und bestimmt das Minimum D der (euklidischen) Abstände zwischen je zwei Punkten. Man kann zeigen, dass für große n die Zufallsvariable $T = n^2 D^s/2$ exponentiell verteilt ist mit Erwartungswert $1/V_s$, wobei V_s das Volumen der s-dimensionalen Einheitskugel ist.

Für einen Test erzeugt man N Werte für T und vergleicht sie mit der Exponentialverteilung.

14.49 Die *Ränge zufälliger Boolescher Matrizen* können ebenfalls zum Test herangezogen werden. Eine echt zufällig mit Nullen und Einsen gefüllte $m \times n$-Matrix hat Rang r mit $1 \leqslant r \leqslant \min(m, n)$ mit Wahrscheinlichkeit

$$2^{-(n-r)(m-r)} \prod_{i=0}^{r-1} \frac{(1-2^{i-n})(1-2^{i-m})}{1-2^{i-r}} \,.$$

Mit diesen Werten können empirisch ermittelte Zahlen verglichen werden. Insbesondere ist es interessant, für die Bits jeweils einer Zeile der Matrix die Bits *einer* Pseudozufallszahl zu benutzen.

14.50 Beim *OPSO-Test* (engl. *overlapping pairs sparse occupancy test*) werden jeweils 10 Bits (z. B., aus einer Pseudozufallszahl) als ein Symbol aus einem 1024-elementigen Alphabet angesehen. Es werden $2^{21} + 1$ solche Symbole erzeugt und für alle 2^{21} überlappenden Paare darauf hin überprüft, welche konkreten Paare von Symbolen fehlen. Die Theorie besagt, dass deren Zahl normalverteilt ist mit Erwartungswert 141 909 und Standardabweichung 290. Damit können die empirischen Zahlen verglichen werden.

Marsaglia (1985) schlägt auch noch andere Tests vor, bei denen allerdings nicht klar ist, welches die korrekten theoretischen Werte sind, mit denen die empirischen Ergebnisse verglichen werden müssen. Die Vergleichswerte bestimmt er daher statt auf theoretischem Wege ebenfalls mit der Hilfe von Pseudozufallszahlengeneratoren. An dieser Stelle ist natürlich eine gewisse Skepsis angebracht.

Allerdings vertritt z. B. L'Ecuyer (1992) die Ansicht, dass dies zumindest akzeptabel sei, solange viele vermutlich gute Pseudozufallszahlengeneratoren zu übereinstimmenden Zahlen kämen, die als Ersatz für die fehlenden theoretischen Werte genommen würden.

14.51 Als Beispiel für einen solchen Test sei hier der *parking lot test* genannt. Dabei ist z. B. ein Quadrat mit Seitenlänge 100 vorgegeben. Auf dieser Fläche sollen nun Einheitskreise („Hubschrauber von oben gesehen“) an zufällig gewählten Stellen $\mathbf{u}_i$ positioniert („geparkt“) werden. Damit ist gemeint, dass $\mathbf{u}_i$ als Mittelpunkt des Kreises gewählt wird, sofern er sich nicht mit anderen, bereits positionierten Kreisen schneidet. Ansonsten liegt eine Kollision vor und es wird nicht geparkt.

Man macht eine gewisse Anzahl n von Versuchen und zählt, wie oft ein Kreis ohne Kollision positioniert werden konnte.

14.52 Das gleiche Problem hat der *OQSO Test* (engl. *overlapping quadruples sparse occupancy test*). Er ist analog zum OPSO-Test benutzt aber Quadrupel von Symbolen über einem 32-elementigem Alphabet.

Literatur

Eichenauer, J. und J. Lehn (1986). „A non-linear congruential pseudo random number generator“. In: *Statistische Hefte* 27, S. 315–326 (siehe S. 125).

Knuth, Donald E. (1998). *The Art of Computer Programming*. 3rd. Bd. 2. Addison-Wesley (siehe S. 121, 123, 129).

L'Ecuyer, Pierre (1992). „Testing Random Number Generators“. In: *Winter Simulation Conference*. ACM, S. 305–313. ISBN: 0-7803-0798-4 (siehe S. 131).

— (1994). „Uniform Random Number Generation“. In: *Annals of Operations Research* 53, S. 77–120 (siehe S. 121).

— (1998). „Random Number Generation“. In: *Handbook of Simulation*. Wiley. Kap. 4, S. 93–137 (siehe S. 121, 124, 129).

L'Ecuyer, Pierre und Richard Simard (2007). „TestU01: A C Library for Empirical Testing of Random Number Generators“. In: *ACM Transactions on Mathematical Software* 33.4 (siehe S. 128).

Marsaglia, George (1985). „A Current View of Random Number Generation“. In: *Computer Science and Statistics, 16th Symposium on the Interface*. Elsevier Science Publishers, S. 3–10 (siehe S. 124, 126, 128, 131).

Matsumoto, Makoto und Takuji Nishimura (1998). „Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator“. In: *ACM Transactions on Modeling and Computer Simulation* 8.1, S. 3–30 (siehe S. 125).

Niederreiter, Harald (1992). *Random Number Generation and Quasi-Monte Carlo Methods*. Bd. 63. CBMS-NSF regional conference series in Appl. Math. SIAM (siehe S. 130).

Panneton, Francois, Pierre L'Ecyuer und Makoto Matsumoto (2006). „Improved Long-Period Generators Based on Linear Recurrences Modulo 2“. In: *ACM Transactions on Mathematical Software* 32.1, S. 1–16. URL: http://www.iro.umontreal.ca/~lecuyer/myftp/papers/wellrng.pdf (siehe S. 125).

Schloissnig, Siegfried (2003). „Zellularautomaten und Zufallszahlen“. Studienarbeit. Univ. Karlsruhe, Fak. f. Informatik (siehe S. 126).

