Ein Reinfall mit Computer-Zufallszahlen

von Bernd Gärtner

Computer und Zufall—passt das zusammen? Computer gelten doch meist als schnelle, aber phantasielose Rechenknechte, die stur und vorhersehbar den gegebenen Anweisungen folgen. Ein Gedankenexperiment zeigt die psychologischen Schwierigkeiten mit der Kombination Computer und Zufall: Stellen Sie sich die Ziehung der Lottozahlen so vor, dass ein kleiner schwarzer Kasten auf der Bühne steht, auf dessen Anzeige in Nullkommanichts sieben Zahlen auftauchen—die vom Computer ausgewählten Lottozahlen der Woche! Hätte der Programmierer sechs Richtige, wären Sie nicht überrascht. Aber auch, wenn Manipulationen ausgeschlossen sind (notarielle Aufsicht!), würden Sie sich fragen, wo der Computer die Zahlen „her" hat. Während wir dem Computer fast blind vertrauen, wenn es um Kontoauszüge und Steuererklärungen geht, sehen wir die Lottozahlen doch lieber aus einem altmodischen mechanischen Apparat tropfen. Warum diese Skepsis nicht nur in psychologischer, sondern auch in mathematischer Hinsicht angebracht ist, möchte ich anhand eines Reinfalls erklären, den ich kürzlich mit Zufallszahlen aus dem Computer erlebt habe.

Vorsicht, Interferenzen!

Jede moderne Programmiersprache liefert eine Funktion zur Erzeugung von Zufallszahlen mit. Diese wird zum Beispiel dazu verwendet, Testeingaben für Programme zu erstellen, um deren „typisches" Verhalten zu untersuchen. Auch wenn dieser Ansatz grundsätzlich fragwürdig ist, enthält doch fast jede experimentelle Arbeit in meinem Spezialgebiet *Algorithmische Geometrie* einen Abschnitt über Ergebnisse auf zufälligen Eingaben. Immerhin sind diese in großen Mengen verfügbar und verleihen den Resultaten den Anschein der Objektivität: nach Definition demonstrieren zufällige Eingaben ja das Verhalten des Programms auf Daten, die außerhalb der Kontrolle der Programmierers liegen, von ihm also insbesondere nicht im Hinblick auf ein erwünschtes Resultat hin manipuliert werden können.

Genau hier aber liegt das Problem, wenn keine echten Zufallszahlen verwendet werden, sondern Folgen von *Pseudozufallszahlen*. Diese sehen nur zufällig aus und werden meist durch eine sehr einfache arithmetische Regel erzeugt—man spricht von einem *Pseudozufallszahlengenerator* (im Folgenden einfach Generator).

Programm und Generator können nun unbemerkt so ineinandergreifen, dass die Regelmäßigkeit der „Zufallsfolge" vom Programm ausgenutzt und das Resultat dadurch verfälscht wird. Die Ergebnisse sind dann nämlich keine Ergebnisse für zufällige Daten mehr, sondern für strukturierte Eingaben, mit denen das Programm vielleicht besonders gut (oder besonders schlecht) umgehen kann. Wir können dieses Phänomen auch physikalisch ausdrücken: Programm und Generator interferieren derart, dass der Zufall ausgelöscht wird.

Nun könnte man denken, eine solche Interferenz sei sehr unwahrscheinlich, sind doch Programm und Generator unabhängig voneinander entstanden. Auch ich hatte zwar immer wieder von Problemen mit Generatoren gehört; deren Ursache wurde in den teils verbürgten, teils an „urban legends" gemahnenden Berichten aber stets auf den „schlechten Generator" zurückgeführt. Das Entscheidende aber war mir zu diesem Zeitpunkt nicht klar: auch für jeden noch so *guten* Generator (was immer das heißt) gibt es potentiell Anwendungen, in denen er versagt. Um das zu erkennen, musste ich allerdings erst auf eine solche stoßen [1].

Erste Fußspuren

Der Programmteil, der alles ins Rollen brachte, hatte die Aufgabe, die *exakte* Inverse A^{-1} einer $(n \times n)$-Matrix A über *Fließkommazahlen* zu verwalten und Produkte $A^{-1}b$ mit verschiedenen Vektoren b zu berechnen. Fließkommazahlen sind die „Billig-Versionen" der reellen Zahlen für arme Zeitgenossen wie Computer, die sich nur beschränkte Darstellungsgenauigkeit leisten können. Eine Fließkommazahl ist von der Form $s \cdot 2^e$, wobei die Mantisse s und der Exponent e ganze Zahlen sind. Die wesentliche Einschränkung ist, dass s eine maximale Anzahl p von Binärstellen hat, die vom gewählten System abhängt ($p = 53$ ist heutzutage ein typischer Wert).

Für die Komponenten A^{-1}_{ij} der Inversen von A gilt nach der Cramerschen Regel bekanntlich

$$A^{-1}_{ij} = (-1)^{i+j} \frac{\det(A^{ji})}{\det(A)}, \qquad (1)$$

wobei die A^{ji} (Streichungs-)Submatrizen von A sind. Wollen wir die Inverse in diesem Format *exakt*—das heißt ohne Rundungsfehler—repräsentieren, so genügt es nicht, die Zahlen $\det(A^{ji})$ und $\det(A)$ wieder durch Fließkommazahlen darzustellen, denn die Determinanten erfordern im allgemeinen eine viel höhere Genauigkeit als die Einträge von A selbst. Ist man aber bereit, auf die Bequemlichkeit der eingebauten Fließkommaarithmetik zu verzichten, können zur Darstellung dieser Determinanten Zahlen s und e

verwendet werden, deren Stellenzahl sich dynamisch den Erfordernissen anpassen kann. Zum Glück existieren Programmbibliotheken, die solche Zahlen und die benötigten arithmetischen Grundoperationen zur Verfügung stellen.[1]

Allerdings muss man sich bewusst sein, dass Addition und Multiplikation von solchen „langen" Zahlen kostspielige Operationen sind. Ihre Komplexität wächst mindestens linear (im Fall der Multiplikation sogar superlinear) mit der Mantissenlänge. An die Effizienz der Fließkommaarithmetik, die oft sogar fest verdrahtet und hochoptimiert im Computer realisiert ist, kann keine Langzahlenarithmetik heranreichen.

Deshalb staunte ich beim Testen mit „zufälligen" Matrizen A auch nicht schlecht über die unerwartet schnelle Berechnung der Produkte $A^{-1}b$ unter der rationalen Langzahlendarstellung (1) von A^{-1}, auch bei größeren Werten der Matrixdimension n. Ich beschloss, der Sache nachzugehen und ließ mir zunächst einmal die berechneten Werte von $\det(A)$ für verschiedene n anzeigen. Und siehe da: deren Beträge wuchsen zwar exponentiell in n, wie ich es erwartet hatte, allerdings manifestierte sich dieses Verhalten hauptsächlich im Exponenten e, kaum aber in der Mantisse s. Konkret hieß das für $n = 20$, dass ich Mantissen s mit nur etwa 90 Binärstellen erhielt, wo nach einer „Pi-mal-Daumen-Rechnung" ungefähr 960 Stellen zu erwarten gewesen wären (wie man auf diesen Wert kommt, werden wir noch sehen). In meinen „zufälligen" Determinanten waren also unerwartet große Zweierpotenzen versteckt, die die fehlenden Stellen geschluckt hatten. Mit solchen Zahlen kann man im Format $s \cdot 2^e$ natürlich sehr schnell rechnen!

Die Ursache musste eine verborgene Interferenz zwischen dem Programm und dem verwendeten Generator sein. Wie Reinhold Messner bei der Suche nach dem Yeti war ich einem Phänomen auf der Spur, an das ich erst glauben konnte, nachdem ich seine „Fußspuren" in meinem Programm gesehen hatte.

Blick hinter die Fassade

In der Programmiersprache `C++` wird üblicherweise ein Generator namens `drand48` benutzt; der heißt so, weil er eine Folge von Fließkommazahlen der Form $X_i \cdot 2^{-48}, i \geq 1$ erzeugt, wobei die Mantissen X_i natürliche Zahlen mit maximal 48 Binärstellen sind. Die erzeugten Zufallszahlen liegen also im Intervall $[0, 1)$. Die Folge hängt von einem Startwert X_0 ab, der vom Programmierer vorgegeben wird.

Ein Blick in die elektronische Handbuchseite enthüllt folgende Informationen über die Arbeitsweise von `drand48` (und anderen mitgelieferten Generatoren, die noch verschiedene Einstellungen erlauben).

```
All the routines work  by generating a sequence of
48-bit  integer  values,  X[i],  according  to  the
linear congruential formula

  X sub (n+1) = (aX sub n + c) sub (mod m)   n>=0.

The parameter m=2**48; hence 48-bit integer arith-
metic is performed.  Unless lcong48() has been in-
voked, the multiplier value a and the addend value
c are given by

  a = 5DEECE66D base 16 = 273673163155 base 8
  c = B base 16 = 13 base 8.
```

Computerfreaks der ersten Stunde wird immer noch warm ums Herz, wenn Hexadezimal- oder Oktalzahlen auftauchen; mich erinnern diese an meinen ersten Schachcomputer, den ich Anfang der Achtziger Jahre bekam. Die Stellungsbewertung wurde hexadezimal angegeben, und obwohl diese Notation in der Bedienungsanleitung kurz erklärt wurde, hieß es gleichzeitig entschuldigend, die Anzeige sei doch mehr für den internen Gebrauch der Entwickler gedacht. Ob das bei den elektronischen Handbuchseiten wohl auch der Fall ist?

Jedenfalls ist die Funktionsweise von `drand48` jetzt klar: die Mantissen X_i werden nach der Formel

$$X_{i+1} = (aX_i + c) \bmod m, \quad i \geq 0 \tag{2}$$

erzeugt, wobei $a = 25214903917$, $c = 11$ und $m = 2^{48}$ (alle Angaben im Dezimalsystem). a und c sehen auf den ersten Blick zwar recht beliebig aus, sind es aber keinesfalls! Dazu später mehr.

Regel (2) definiert die *lineare Kongruenzmethode* zur Erzeugung von Pseudozufallszahlen. Dies ist die weitestverbreitete Methode, weil sie so einfach zu programmieren ist und für viele Anwendungen gute Zufallszahlen liefert. Es existieren unzählige Generatoren, die auf der linearen Kongruenzmethode basieren und sich jeweils nur in den Parametern a, c und m unterscheiden. Ich wusste vom Hörensagen, dass `drand48` „besser" sein soll als sein Vorgänger `rand`, der früher zum Standard der Programmiersprache `C` gehörte. War er vielleicht trotzdem nicht gut genug für meine Anwendung? Und was sind überhaupt Kriterien, nach denen die Güte der Parameter beurteilt werden kann?

[1] Genannt sei hier die *Library of Efficient Data Types and Algorithms* (LEDA), siehe `http://www.mpi-sb.mpg.de/LEDA/`.

Der Spektraltest und ein Hochstapler

Aus der Formel (2) ergibt sich sofort, dass die Folge (X_i) irgendwann periodisch wird, wobei die Periodenlänge höchstens m beträgt. Soll der Generator (wie in meiner Anwendung) dazu eingesetzt werden, zufällige Zahlen X_i/m im Intervall $[0, 1)$ zu erzeugen, so muss zunächst einmal m sehr groß sein, um eine möglichst feine Unterteilung des Intervalls zu bekommen. Das setzt aber auch voraus, dass die Periode der Folge möglichst lang ist. Wenn der Modulus m eine Zweierpotenz ist (und auf diesen Fall wollen wir uns im folgenden beschränken), wird genau dann Periode m erreicht, wenn $a-1$ durch 4 teilbar und c ungerade ist [3].

Mögliche Generatoren mit voller Periode für $m = 256$ sind dann etwa

$$X_{i+1} = (137X_i + 187) \bmod 256$$

und

$$X_{i+1} = (193X_i + 73) \bmod 256.$$

Da es üblich ist, solchen Generatoren Namen zu geben (was dokumentieren soll, dass die Wahl der Parameter mit Bedacht erfolgt ist), möchte ich den ersten mit `knuth16` bezeichnen, den zweiten mit `adhoc16`.[2] 256 ist kein geeigneter Modulus für praktische Anwendungen, liefert hier aber anschauliche Beispiele.

Ein Kriterium zur Unterscheidung der Generatoren liefert die Betrachtung der Folge aller m Paare $(X_i/m, X_{i+1}/m)$, $0 \le i < m$. Wenn es halbwegs zufällig zugehen soll, muss deren Verteilung in irgendeiner Weise die Gleichverteilung auf dem Einheitsquadrat approximieren. Abbildung 1 zeigt die Verteilungen der jeweils 256 Punkte $(X_i/256, X_{i+1}/256), 0 \le i < 256$ für beide Generatoren. Intuitiv ist klar, dass `knuth16` eine bessere Annäherung an die Gleichverteilung darstellt.

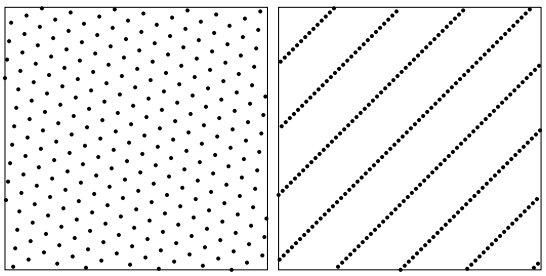

Abbildung 1: Verteilung der Paare $(X_i/256, X_{i+1}/256)$ bei `knuth16` (links) und `adhoc16` (rechts).

Der sogenannte *Spektraltest* formalisiert diese Intuition (auf eine mögliche Weise) und münzt sie in eine Vorschrift zur Bestimmung der Qualität eines Generators um. Er betrachtet Streifen (Regionen zwischen parallelen Geraden) im Einheitsquadrat; dabei bestimmt der breiteste Streifen, der kein Paar $(X_i/m, X_{i+1}/m)$ enthält, die Güte des Generators. Der Kehrwert dieser maximalen Breite ist dann das (zweidimensionale) Gütemaß.

Die Qualität von `adhoc16` ist damit $\sqrt{32}$, während man bei `knuth16` durch etwas schärferes Hinsehen auf $\sqrt{274}$ kommt. Man kann zeigen, dass die bestmögliche Qualität, die mit der linearen Kongruenzmethode erreichbar ist, ungefähr $\sqrt{m}$ beträgt, wobei diese Schranke nicht genau ist, wie man im Fall von `knuth16` sieht. Der Wert von $\sqrt{m}$ ist jedenfalls ein guter Anhaltspunkt—sehr viel schlechter darf die Qualität eines guten Generators nicht sein.

Der Spektraltest kann auch in höheren Dimensionen durchgeführt werden; im dreidimensionalen etwa betrachtet man Tripel $(X_i/m, X_{i+1}/m, X_{i+2}/m)$ im Einheitswürfel und Streifen zwischen parallelen Ebenen. Furore machte der Spektraltest durch die Entlarvung des größten Hochstaplers in der Geschichte der linearen Kongruenzmethode. Der Generator `randu`, von IBM Anfang der Sechziger Jahre eingeführt, war fast ein Jahrzent lang weitverbreitet. Er hat die Formel

$$X_{i+1} = 65539X_i \bmod 2^{31}$$

und ist damit ein sogenannter *multiplikativer* Generator. Werte von $c = 0$ sind durchaus üblich und sinnvoll, obwohl multiplikative Generatoren keine volle Periode erreichen können. `randu` hat immerhin Periode 2^{29}, was in diesem Fall bestmöglich ist. Nicht bestmöglich (sondern fast schlechtestmöglich, wie Knuth findet), ist allerdings die Wahl des Multiplikators $a = 2^{16} + 3$, der dazu führt, dass alle 2^{29} Tripel $(X_i/m, X_{i+1}/m, X_{i+2}/m)$ sich in nur 15 parallelen Ebenen aufhalten (siehe Abbildung 2)! Die dreidimensionale Qualität dieses Generators ist $\sqrt{118}$, während der anzustrebende maximale Wert etwa $\sqrt[3]{m} \approx 1000$ beträgt. `randu` schneidet also spektakulär schlecht ab, wenn es darum geht, zufällige Punkte im dreidimensionalen Einheitswürfel zu erzeugen. Da der Generator laut Park und Miller darüber hinaus auch keine signifikanten wiedergutmachenden Eigenschaften aufweist [5], gehört er auf den Müllhaufen der Informatik-Geschichte.

Der Generator `drand48` hingegen „besteht“ den Spektraltest in allen relevanten Dimensionen. Was hat der Test also mit den Problemen zu tun, die ich mit diesem Generator hatte?

[2] `knuth16` wird schon von Knuth als Beispiel verwendet [3], und `adhoc16` ist das Resultat sorgfältiger Entwicklungsarbeit meinerseits.

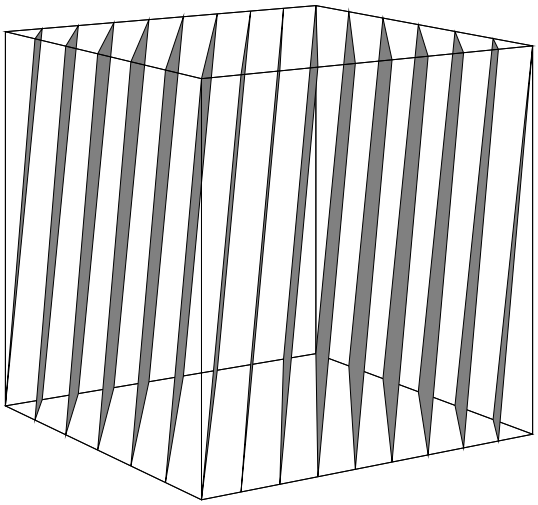

Abbildung 2: Die 15 Ebenen von `randu`

Die Suche endet hinter Gittern

Der Spektraltest in Dimension d befasst sich (bis auf Normierung) mit der Struktur der d-dimensionalen Vektoren $\mathbf{X}_i := (X_i, X_{i+1}, \ldots, X_{i+d-1})^T$, $0 \le i < m$, und mit ein wenig linearer Algebra können wir diese Struktur entschlüsseln. Zunächst kann man aus (2) per Induktion über s leicht die Formel

$$X_{i+s} - X_s \equiv a^s (X_i - X_0) \pmod m, \quad i, s \ge 0$$

herleiten, woraus für den Vektor $\mathbf{X}_i - \mathbf{X}_0$ die Beziehung

$$\mathbf{X}_i - \mathbf{X}_0 \equiv (X_i - X_0) \begin{pmatrix} 1 \\ a \\ \vdots \\ a^{d-1} \end{pmatrix} \pmod m$$

folgt (Kongruenz ist dabei komponentenweise erklärt). Verwenden wir die Definition der Kongruenzrelation, erhalten wir

$$\mathbf{X}_i - \mathbf{X}_0 = (X_i - X_0) \begin{pmatrix} 1 \\ a \\ \vdots \\ a^{d-1} \end{pmatrix} + m \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix},$$

wobei die u_j ganze Zahlen sind. Das heißt aber, $\mathbf{X}_i - \mathbf{X}_0$ ist eine *ganzzahlige* Linearkombination der $d+1$ Vektoren

$$\begin{pmatrix} 1 \\ a \\ \vdots \\ a^{d-1} \end{pmatrix}, \begin{pmatrix} m \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ m \\ \vdots \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ 0 \\ \vdots \\ m \end{pmatrix}. \quad (3)$$

Der Vektor $(m, 0, \ldots, 0)^T$ ist dabei sogar redundant, weil er sich ganzzahlig aus den anderen kombinieren lässt. Die $\mathbf{X}_i - \mathbf{X}_0$ liegen also für alle i auf einem *Gitter*, welches durch die ganzzahligen Linearkombinationen einer *Gitterbasis*

$$B = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ a & m & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a^{d-1} & 0 & \cdots & m \end{pmatrix}$$

erzeugt wird. Einen Ausschnitt dieses Gitters (jeweils um X_0 verschoben und mit dem Faktor $1/m$ skaliert) sehen wir in Abbildung 1 für die beiden Generatoren `knuth16` und `adhoc16`.

Was heißt das für `drand48` und mein Problem? Erinnern wir uns: ich hatte beobachtet, dass die Determinante einer „zufälligen“ Matrix von Fließkommazahlen

$$A = \frac{1}{m} \begin{pmatrix} X_0 & X_1 & \cdots & X_{n-1} \\ X_n & X_{n+1} & \cdots & X_{2n-1} \\ \vdots & \vdots & & \vdots \\ X_{(n-1)n} & X_{(n-1)n+1} & \cdots & X_{n^2-1} \end{pmatrix}$$

(mit $m = 2^{48}$) eine sehr kurze Mantisse s in der Darstellung $\det(A) = s \cdot 2^e$ hat, dafür aber einen übermäßig großen Exponenten e.

Offensichtlich hat der Skalierungsfaktor 2^{-48} mit der Mantissenlänge nichts zu tun. Die Zeilen der ganzzahligen Matrix

$$M = mA$$

sind dann aber genau n-dimensionale Vektoren der Form $\mathbf{X}_0, \mathbf{X}_n, \ldots, \mathbf{X}_{(n-1)n}$, über deren Struktur wir jetzt schon eine ganze Menge wissen!

Die Determinante von M, der wir nun eine große Zweierpotenz als Teiler nachweisen wollen, berechnet sich als

$$\begin{aligned} \det(M) &= \det(\mathbf{X}_0, \mathbf{X}_n, \ldots, \mathbf{X}_{(n-1)n}) \\ &= \det(\mathbf{X}_0, \mathbf{X}_n - \mathbf{X}_0, \ldots, \mathbf{X}_{(n-1)n} - \mathbf{X}_0) \\ &= \frac{1}{m} \det(L), \end{aligned}$$

mit

$$L := (m\mathbf{X}_0, \mathbf{X}_n - \mathbf{X}_0, \ldots, \mathbf{X}_{(n-1)n} - \mathbf{X}_0).$$

Wir können leicht sehen, dass alle Spalten von L Elemente des von B erzeugten Gitters sind. Für die letzten $n-1$ Spalten hatten wir uns genau das vorher schon überlegt, und für die Spalte $m\mathbf{X}_0$ folgt diese Tatsache daraus, dass die Menge der Vektoren in (3) ein Erzeugendensystem des Gitters ist.

Das heißt aber auch, dass L in der Form

$$L = BQ$$

darstellbar ist, wobei Q eine ganzzahlige Koeffizientenmatrix ist. Insbesondere gilt dann

$$\begin{aligned} \det(M) &= \frac{1}{m}\det(L) = \frac{1}{m}\det(B)\det(Q) \\ &= m^{n-2}\det(Q), \end{aligned} \tag{4}$$

womit im Fall von `drand48` bewiesen ist, dass die wirklich beachtliche Zweierpotenz

$$2^{48(n-2)}$$

ein Teiler von $\det(M)$ ist! Für $n = 20$ verschwinden also $48 \cdot 18 = 864$ Binärstellen der Determinante ungeplant im Exponenten, was ziemlich genau die Differenz zwischen erwarteter und beobachteter Mantissenlänge erklärt.[3]

Gefahr erkannt, Gefahr gebannt?

Kann `drand48` nun guten Gewissens als Bösewicht gebrandmarkt werden, der in eine Kategorie mit dem Hochstapler `randu` fällt? Sicher nicht, denn im Gegensatz zu `randu` kann der Generator `drand48` gar nichts für sein schlechtes Verhalten! Sein wesentliches Defizit, Determinanten mit großen Potenzen des Modulus m als Teiler zu produzieren, teilt er nach Gleichung (4) nämlich mit allen seinen Geschwistern, die auch von der linearen Kongruenzmethode abstammen. Die Methode selbst ist also im Zusammenhang mit Determinanten gefährlich, und Eltern haften für ihre Kinder.

Was ist generell zu tun, um Reinfälle mit Generatoren zu vermeiden? Zunächst einmal sind viele Klassen von Generatoren gut untersucht, und ihre wesentlichen Defizite sind bekannt und veröffentlicht. Im Zweifel sollte man sich also im Voraus über geeignete Generatoren informieren—es gibt sogar Anleitungen dazu[2]. Es mag immer wieder Fälle geben, in denen neue, bisher unentdeckte Defekte zum Vorschein kommen, die meisten Reinfälle dürften aber auf das Konto mangelnder Kenntnis der bekannten Defizite gehen. In meinem Fall war das nicht anders: bereits 1970 hatte George Marsaglia, der Guru der Pseudozufallszahlen, das hier beschriebene Phänomen in ähnlicher Form publiziert [4].

Eine interessante Alternative zu Pseudozufallszahlen scheinen „echte“ Zufallszahlen zu sein, die zum Beispiel im Internet angeboten werden. Eine Quelle sind John Walkers *HotBits*, die durch radioaktiven Zerfall erzeugt werden. Walkers köstliche Beschreibung des theoretischen Hintergrunds ist wirklich lesenswert. Um *HotBits* zu verwenden, schickt man einfach eine elektronische Anfrage an Walker und erhält die gewünschte Menge von Zufallszahlen frei Rechner geliefert.[4]

Problematisch bei solchen Hardwarelösungen sind mögliche Konstruktionsmängel, aufgrund derer die scheinbar verbannten Interferenzphänomene durch die Hintertür wieder hereinkommen können. Auch ist die Rate, mit der Hardware-Generatoren ihre Zahlen liefern, oft wesentlich kleiner als die arithmetischer Methoden. Bei Simulationen, die wirklich viele Zufallszahlen benötigen, sind sie dann zu langsam. Schließlich ist die Reproduzierbarkeit der Ergebnisse ein Problem: bei Hardware-Generatoren bleibt einem dafür nichts anderes übrig, als die gesamte Zufallsfolge abzuspeichern. Das ist oft nicht machbar. Insofern behalten Pseudozufallszahlengeneratoren (darunter auch die lineare Kongruenzmethode) sicher noch lange ihre Bedeutung, denn bei ihnen weiß man wenigstens genau, woran man ist.

Zum Schluss möchte ich den geneigten Lesern noch eine von George Marsaglia herausgegebene CD ans Herz legen, die knapp 5 Milliarden zufällige „Bits“ in Form einer Folge von Nullen und Einsen enthält. Diese wurden nach Angaben von Marsaglia erzeugt, indem die Ausgabe modernster Generatoren mit verschiedenen Quellen weißen Rauschens sowie mit „schwarzem“ Rauschen—digital aufgenommener Rap-Musik—kombiniert wurde.[5] Da eine wirklich zufällige Folge informationstheoretisch überhaupt keinen Gehalt hat, ist diese CD sicher das ultimative Geschenk für alle Opfer von Reizüberflutung.

Literatur

[1] B. Gärtner. Exact arithmetic at low cost – a case study in linear programming. In *Proceedings of the 9th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 157–166, 1998.

[3] Hier ist noch die „Pi-mal-Daumen-Rechnung“ für die erwartete Mantissenlänge nachzutragen: zunächst gilt, dass eine ganzzahlige Matrix M mit wirklich zufälligen Einträgen in $\{0, \ldots, m-1\}$ mit hoher Wahrscheinlichkeit keine große Zweierpotenz als Teiler hat—das folgt aus einer einfachen Formel für die Anzahl der nichtsingulären Matrizen über $GF(2^q)$, siehe [6, Beweis von Proposition 1.3.18]. Ferner gibt es eine hübsche explizite Formel für die erwartete Größe von $\det^2(M)$. Es gilt nämlich $E[\det^2(M)] = n!\,((m-1)(m+1)/12)^n\,(1+3n(m-1)/(m+1))$. Als „erwartete“ Stellenzahl für s habe ich dann $\log_2(E[\det^2(M)])/2$ angesetzt. Das ist keine beweisbar korrekte Schätzung, sie stimmt aber gut mit den Beobachtungen überein.

[4] siehe `http://www.fourmilab.ch/hotbits/`

[5] siehe `http://whyfiles.news.wisc.edu/009poll/random.html`

[2] P. Hellekalek. Good random number generators are (not so) easy to find. *Mathematics and Computers in Simulation*, 46:485–505, 1998.

[3] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 3rd edition, 1998.

[4] G. Marsaglia. Regularities in congruential random number generators. *Numerische Mathematik*, 16:8–10, 1970.

[5] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31:1192–1201, 1988.

[6] R. P. Stanley. Enumerative Combinatorics I. In *Cambridge Studies in Advances Mathematics*, volume 49. Cambridge University Press, 1997.

Adresse des Autors:
Bernd Gärtner
Institut für Theoretische Informatik
ETH Zürich
Haldeneggsteig 4
CH-8092 Zürich, Schweiz
gaertner@inf.ethz.ch