Universität Würzburg
Institut für Informatik
Research Report

Implementierung und Test neuartiger Zufallszahlengeneratoren

S. Wahler, O. Rose, A. Schömig

Report No. 180 September 1997

Institut für Informatik, Universität Würzburg
Am Hubland, 97074 Würzburg, Deutschland
Tel.: +49 931 888-5510, Fax: +49 931 888-4601
http://www-info3.informatik.uni-wuerzburg.de

Zusammenfassung

Neben dem klassischen linearen Kongruenzgenerator beginnen sich zunehmend auch andere Techniken der Zufallszahlenerzeugung zu etablieren. Im Rahmen dieser Arbeit werden einige dieser Algorithmen verglichen.
Kapitel 1 rekapituliert einige Grundlagen zur Thematik. Kapitel 2 beschäftigt sich mit der Methode der Zufallszahlenerzeugung aufgrund linearer Kongruenzen, zusätzlich wird der Lagged-Fibonacci-Generator, ein relativ neuer Generatortyp, der ebenfalls lineare Kongruenzen benutzt, vorgestellt. Kapitel 3 widmet sich zwei Vertretern der nichtlinearen Kongruenzgeneratoren, dem „inversen Kongruenz- generator" und dem „explizit inversen Kongruenzgenerator". In Kapitel 4 soll schließlich die Methode der Zufallszahlenerzeugung durch physikalische Messungen aufgegriffen werden. Kapitel 5 stellt eine Übersicht über die empirischen Eigenschaften der Generatoren anhand ausgewählter Tests dar und Kapitel 6 zeigt die Implementierung und Einbindung der Generatoren in ein bestehendes Simulationstool.

Einführung

„It is not easy to invent a foolproof source of random numbers. This fact was convincingly impressed upon the author several years ago, when he attempted to create a fantastically good generator... The moral of this story is that random numbers should not be generated with a method chosen at random. Some theory should be used.
-- Donald E. Knuth

Anforderungen an einen Zufallszahlengenerator

Zufallszahlengeneratoren sind ein wichtiger Bestandteil jeder Simulation oder Analyse. Durch die gestiegene Rechenleistung moderner PCs oder Workstations wird es mögliche selbst komplexe Systeme mit einer Detailtreue zu simulieren, wie es noch vor einigen Jahren undenkbar war. Komplexere Simulationen steigern aber auch die Anforderungen an den jeweiligen Zufallszahlengenerator. Lange Zeit wurde der Einfluß von Zufallszahlengeneratoren auf die Ergebnisse von Simulation und Analyse unterschätzt. Durch die zunehmende Detailtreue machen sich Defizite bei der Zufallszahlenerzeugung im Ergebnis aber immer deutlicher bemerkbar. Auch erfordert die parallele Simulation von immer mehr Prozessen, viele Teilfolgen (streams) der eigentlichen Zufallszahlenfolge erzeugen zu können, ohne dabei statistische oder strukturelle Eigenschaften einzubüßen.
Durch umfangreiche Tests wurde gezeigt, daß es keinen Pseudozufallszahlengenerator geben kann, der alle möglichen Anforderungen der Simulationstechnik abdecken kann. Um so wichtiger ist es, mehrere verschiedene Methoden der Zufallszahlenerzeugung inklusive ihrer Vor-, aber auch ihrer Nachteile, zu kennen und so für die gestellte Aufgabe den richtigen Generator auswählen zu können.
Die Forderungen heutiger Simulationstechnik an einen Zufallszahlengenerator lassen sich wie folgt formulieren:

1. Die erzeugten Zahlen sollen eine im gegebenen Intervall gleichverteilte Zufallsvariable „simulieren", also Zufallszahlen erzeugen, die
   - gleichverteilt sind im Intervall $U([0;1[)$
   - keine Korrelation untereinander aufweisen
   - eine gute Streudichte aufweisen
2. Der Algorithmus soll schnell sein, dabei aber keine hohen Speicheranforderungen stellen
3. Jede Folge von Zufallszahlen soll reproduzierbar sein, um
   - Fehlersuche (debugging) und Verifikation des Simulationsmodelles zu ermöglichen.
   - den Vergleich verschiedener Simulationsmodelle durch Verwendung der gleichen Zufallszahlenfolge zu ermöglichen.
4. Es soll möglich sein, durch Aufteilung der Zufallszahlenfolge in Streams, parallel ablaufende Prozesse zu simulieren.

Die Beschränkung auf Gleichverteilung im Intervall $U([0;1[)$ ist dabei in der Realität kein Nachteil. Zufallszahlenerzeugung ist immer ein zweistufiger Prozeß. Erst werden gleichverteilte Zufallszahlen erzeugt, im zweiten Schritt werden diese dann in die gewünschte Verteilung transformiert. Die hierzu verwendeten Methoden sind nicht Thema dieser Arbeit.

Anmerkungen zum Thema „Zufälligkeit"

Besondere Beachtung verdient dabei Punkt 1 der obigen Forderungen Der wesentliche Punkt bei der Simulation von Zufall mit einem Computer ist, daß letzterer ist, was vorheriger absolut nicht ist: deterministisch. Auch das o.g. Zitat von J. von Neumann verdeutlicht, weshalb die Formulierung von „simulierten Zufallsvariablen" gebraucht wurde.
Pseudozufallszahlengeneratoren sind nicht mehr und nicht weniger, als Algorithmen, die nach genau definierten mathematischen Regeln eine periodische Folge von Zahlen produzieren, die man als Realisationen einer gleichverteilten Zufallsvariable im Intervall $U([0;1[)$ auffassen kann. Anders ausgedrückt:

> Pseudozufallszahlen sind für Anwendungen gemacht. Sie haben Eigenschaften, die stark von der gewählten Art der Generierung abhängen. Was jedoch „zufällig" ist, das kommt ganz auf die Erfordernisse der jeweiligen Situation an. Allein die aktuelle Anwendung bestimmt die statistisch, strukturell oder empirisch relevanten Eigenschaften, die die generierten Zufallszahlen erfüllen müssen.

Auch wenn diese einleitenden Worte nur Zusammenhänge und Tatsachen widerspiegeln, die im Prinzip seit mehreren Jahrzehnten in der Simulationstechnik bekannt sind, so erscheint es, vor allem vor dem Hintergrund vieler Veröffentlichungen der letzten Zeit, notwendig, sie immer wieder in das Gedächtnis zurückzurufen. Das obige Zitat von Donald E. Knuth hat also, obwohl bereits aus dem Jahre 1968, noch nichts von seiner Aktualität eingebüßt.

Vorstellung der Generatoren

„Pseudorandom number generatores are like antibiotics.
No generator will be appropriate for all tasks"
-- P. Hellekalek

Alle hier aufgeführten Generatoren produzieren idealerweise eine Folge $(y_n)_{n=0}^{N-1}$ von Werten aus der Grundmenge $\{0,1,...,M-1\}$. $U([0;1[)$ gleichverteilte Zufallszahlen $(x_n)_{n=0}^{N-1}$ lassen sich durch die Normalisierung $x_n = \frac{y_n}{M}$ erzeugen..
Diese Methode bietet zwei Vorteile:

- Die gesamte Berechnung der Folge (u_n) wird ausschließlich durch Ganzzahl-Arithmetik durchgeführt. Rundungsfehler durch Fließkomma-Arithmetik treten also höchstens bei der anschließenden Normierung auf und haben keinen Einfluß auf andere Folgenglieder. Als Folge davon läßt sich der Generator leichter auf andere Rechnersysteme portieren, da die u.U. differierenden Repräsentationen der Gleitkommazahlen keine Rolle spielen.
- Die abgeschlossenen Menge $\{0,1,...,M-1\}$ bietet durch Ausnutzung ihrer algebraischen und numerischen Struktur die Möglichkeit schnellstmöglicher Berechnungen. Eine genaue Behandlung dieses Gebietes findet sich in [LiNi83] und [Le95].

Die bekannteste und wichtigste Klasse der Generatoren ist nach wie vor die der linearen Kongruenzgeneratoren, weshalb die Vorstellung der Generatoren auch mit diesem Typ beginnt.

Lineare Kongruenzgeneratoren

Vorgestellt wurde diese Methode erstmals im Jahr 1951 durch Lehmer (vgl. [LaKe91]). Diese Generatoren stellen heute quasi das „workhorse" der Simulationstechnik dar und werden in allen bekannten Simulationstools (z.B. ModSim, Arena) in verschiedenen Versionen eingesetzt.

Linearer Kongruenzgenerator

Einführung und Schreibweise

Bei der Erzeugung von Zufallszahlen durch einen linearen Kongruenzgenerator (linear congruential generator, LCG) sind folgende Parameter zu wählen:

- M ... Modulus, $\{0,1,...,M-1\}$ stellt die Grundmenge des Generators dar
- a ... der Vorfaktor
- b ... die additive Konstante
- y_0 ... der Startwert („seed"), mit dem der Generator initialisiert wird.

Der lineare Kongruenzgenerator wird dann definiert durch die Rekursion

$$y_{n+1} = a * y_n + b(modM), n \geq 0.$$

Dieser Generator wird durch die Schreibweise $LCG(a,b,M,y_0)$ eindeutig charakterisiert.
Die übliche Normalisierung $x_n = \frac{y_n}{M}$ erzeugt daraus dann die gewünschte Sequenz $(x_n)_{n\geq 0}$ in $U([0;1[)$.

Eigenschaften

Nicht umsonst ist der LCG die heute am weitesten verbreitete Methode der Zufallszahlenerzeugung. Die wesentlichen Vorteile der LCGs sind anhand der Rekursionsvorschrift erkennbar:

- schnelle Berechnung durch einfache Rekursion und ausschließliche Verwendung von Ganzzahl-Arithmetik.
- geringer Speicherplatzbedarf, da zur Berechnung einer Zufallszahl nur der direkte Vorgänger bekannt sein muß.
- Reproduzierbarkeit der Zufallszahlenfolge ist durch den verwendeten deterministischen Algorithmus völlig problemlos, es ist nur erforderlich sich den Startwert der Folge zu speichern.

Ein weiterer Vorteil des LCG ist, daß seine Zufallszahlenfolge auch iterativ berechnet werden kann. Durch Induktion zeigt man:

$$y_i = \left[a^i y_0 + \frac{c(a^i-1)}{a-1}\right] (\text{mod m})$$

Unzweifelhaft produziert die lineare Kongruenzmethode eine periodische Folge von Zufallszahlen. Nachdem jede berechnete Zahl nur vom Vorgänger abhängt, ist klar, daß der Generator seine Periode vollendet hat, falls ein Wert zweimal auftaucht. Die maximal mögliche Periodenlänge, eines solchen Generators ist dabei sein Modulus M. Generiert ein LCG alle Werte im Intervall $[0, M-1]$, so hat er „volle Periode". Seine Periodenlänge ist damit unabhängig von der gewählten Initialisierung (Vorsicht, falls), was bei Generatoren mit unvollständiger Periode keineswegs gegeben ist.

Die Frage, ob ein LCG volle Periode hat, läßt sich aufgrund des folgenden Theorems entscheiden [LaKe91]:

> **Theorem**: Ein LCG hat volle Periode dann und nur dann, wenn die folgenden drei Bedingungen erfüllt sind:
>
> 1. Die einzige positive ganze Zahl, die M und b teilt, ist 1. Anders ausgedrückt: M und b sind teilerfremd.
> 2. Falls q eine Primzahl ist, die M teilt, dann teilt q auch (a-1).
> 3. Falls 4 Teiler von M ist, dann auch von (a-1).

In der Praxis werden deshalb drei Klassen von LCGs verwendet:

- $LCG(a, b, 2^a, y_0)$ wobei M eine Zweierpotenz darstellt, $a \equiv 5$ (mod 8), b ein ungerader Integer-Wert, y_0 beliebig ist.
- $LCG(a, 0, M, y_0)$ wobei M Primzahl, a eine primitive Wurzel (mod M) und $y_0 \neq 0$ ist.
- $LCG(a, 0, 2^a, y_0)$ wobei M wieder eine Zweierpotenz, $a \equiv 5$ (mod 8) und y_0 gerade ist.

Die beiden letztgenannten Generatortypen werden auch multiplikative Kongruenzgeneratoren genannt. Für eine theoretisch Diskussion dieser Klassifizierung siehe [Ni92] und [Ri87]. Für eine Berechnung der Parameter des ersten Typs siehe [PaMi88].
Die beiden wichtigsten empirischen Anforderungen an einen Generator

- Gleichverteilung: die erzeugten Pseudozufallszahlen sollen im Intervall $U([0; 1[)$ gleichverteilt sein
- Unabhängigkeit: im Abstand d (d beliebig) aufeinanderfolgende Zufallszahlen sollen unkorreliert sein.

meistert die Zufallszahlenfolge des LCG für gut gewählte Parameter und einfache Tests bekanntermaßen ohne Probleme. In Kapitel 5 wurden im Rahmen des Vergleichstests ein χ^2-Test und ein Autokorrelationstest durchgeführt, die diese Behauptungen bestätigen.
Aufgrund dieser Eigenschaften wurde und wird der LCG als der Standardgenerator in allen Simulationspaketen eingesetzt. Untersucht man dagegen den LCG noch etwas genauer, so entdeckt man einige, längst bekannte, aber trotzdem kaum beachtete Defizite.
Der LCG verursacht vor allem zwei wesentliche Probleme:

- der LCG reagiert nicht nur sehr sensibel auf die Wahl der Parameter sondern auch auf die Art der Auswahl seiner Teilfolgen (streams).
- bei der Betrachtung von Lattice-Strukturen im s-dimensionalen Hyperraum (Lattice Test mit überlappenden s-Tupeln $(x_n, x_{n+1}, ..., x_{n+s})$) versagt jede Form des LCG, z.T. schon bei relativ kleinen Werten von s, d.h. die Unabhängigkeit der Zufallszahlenfolge ist nicht so gut wie angenommen.

Als Beispiel für ersteres Problem soll hier der „minmal standard generator" dienen, der in unserer Schreibweise dem $LCG(16807, 0, 2^{31} - 1, y_0)$ entspräche.
Hellekalek et. al. (vgl. [HMW94]) führten mit diesem weit verbreiteten Generator einen Runs-Up Test mit K=100 sukzessiven Proben und einer festen Größe der Samples N durch. Diese 100 Werte sollten anschließend annähernd χ^2-verteilt sein. Die horizontalen Linien in den Diagrammen stellen dabei die Quantile des auf diese 100 Werte angewandten Kolmogoroff-Smirnov-Tests für ein Signifikanzniveau von 0.05 und 0.01 dar. Die x-Achse repräsentiert die Größe der Proben von 2^{12}bis 2^{21}, die y-Achse zeigt die Werte des Tests von Kolmogoroff-Smirnov.

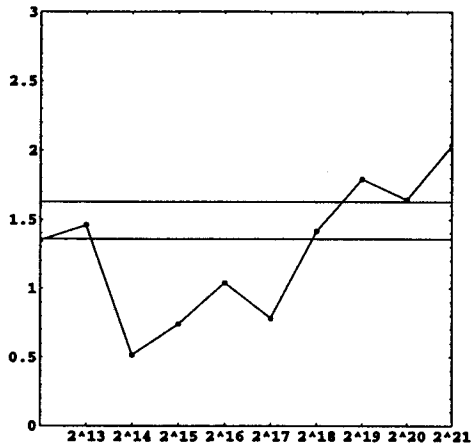

Bild 1
Substream $(x_{77\cdot n})$

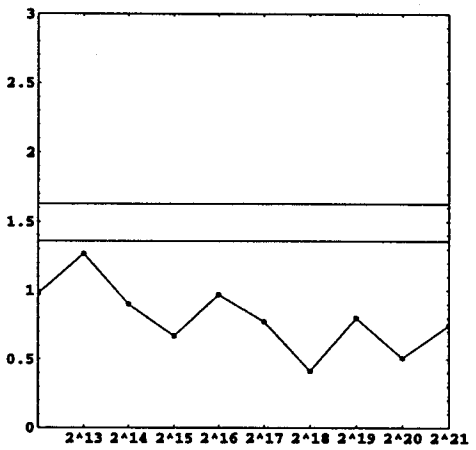

Bild 2
Substream $(x_{16\cdot n})$

Während die Teilfolge (x_{16n}) den Test meistert, versagt die Teilfolge (x_{77n}) völlig. Dieser Effekt ist dabei nicht auf die relativ groß gewählten Proben zurückzuführen. Weiterführende Untersuchungen von K. Entacher (vgl. [En94]) förderten bei jedem der untersuchten LCGs dasselbe Resultat zu Tage. Diese Tatsache behindert vor allem Maßnahmen zur Parallelisierung des LCG, da die so erzeugten Streams z.T. keineswegs die erwarteten Eigenschaften aufweisen.
Auch zweites oben angesprochenes Problem, die Defizite im Lattice-Test, können für tiefgreifende Verfälschung der Simulationsergebnisse verantwortlich sein.
Bei Tupeln $(x_n, ..., x_{n+s})$ für kleines s ist zunächst noch eine schöne feinkörnige Gitterstrukur erkennbar.

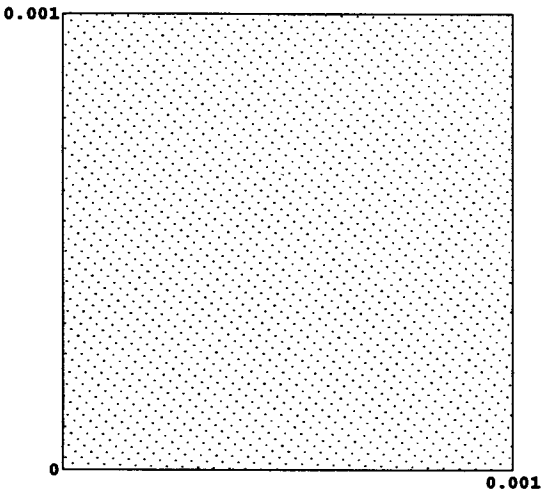

Bild 3
$LCG(950706376, 0, 1, 2^{31} - 1)$

Unweigerlich kommt aber für jeden LCG ein Wert σ, sodaß der Lattice Test für überlappende Tupel $(x_n, ...x_{n+\sigma})$ große Lücken im Hyperraum der Dimension σ hinterläßt (vgl. [Le95] [HMW94] [We94]). Das spektakulärste Beispiel in diesem Zusammenhang ist ohne Zweifel der Generator RANDU von IBM (entspricht: $LCG(65539, 0, 1, 2^{31})$). Bereits bei einer Betrachtung der Tripel (x_n, x_{n+1}, x_{n+2}) ist von einer feinkörnigen Sättigung des dreidimensionalen Hyperraums nichts mehr zu erkennen.

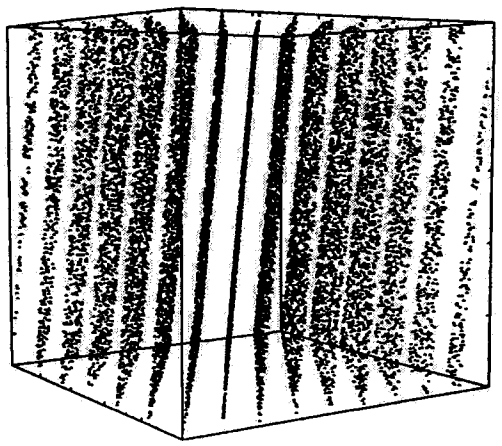

Bild 4
LCG$(65539, 0, 1, 2^{31})$ „RANDU"

Deutlich sind hier die 15 Ebenen zu erkennen, die Marsaglia bereits 1968 in seiner bekannten Veröffentlichung: „Random numbers fall mainly in the planes" (vgl. [Ma68]) beobachtete und untersuchte. Für weitere Beispiele zu dieser Thematik sei wiederum auf die Arbeiten von Hellekalek et. al. (vgl. [HMW94]) verwiesen, wo die beobachteten Korrelationen anhand von weiteren statistischer Verfahren genauer betrachtet wurden.
Im Gegensatz zu diesen empirischen Tests bieten die Arbeiten von Hannes Leeb (vgl. [Le95]) eine direkte mathematische Analyse der entstehenden Lattice-Strukturen.
Beides würde in diesem Zusammenhang aber zu weit führen, da in dieser Arbeit Zufallszahlengeneratoren aus der Sicht der Simulationstechnik gesehen werden sollen.
Als unangenehm, wenn auch nicht so bedeutend wie die beiden o.g. Phänomene, müssen die „long-range correlations" also die statistische Korrelation von Zufallszahlen x_n und x_{n+d} in großer, aber fester Entfernung d angesehen werden. Diese Tatsache limitiert natürlich die Länge der in der Simulation verwendbaren Zufallszahlenfolge. Weiterführende Informationen zu diesem Thema bieten die Arbeiten von A. de Matteis und S. Pagnutti (vgl. [MaPa90]).
Alles in allem läßt sich sagen, daß die einfache und regelmäßige Rekursionsvorschrift des LCG, die ihn zum am weitesten verbreiteten Zufallszahlengenerator werden lies, letztendlich die auch die Ursache für o.g. Probleme ist.
Aus diesem Grund kam der Gedanke nach andern Erzeugungsvorschriften auf. Die Entwicklung ging dabei hauptsächlich in zwei Richtungen:

- Abkehr von der linearen Rekursion und Suche nach neuartigen Methoden der Zufallszahlengenerierung. Beispiele hierfür werden im nächsten Kapitel in Form von inversem- und explizit-inversem Kongruenzgenerator behandelt.
- Weiterentwicklung der linearen Rekursion zur systematischen Ausmerzung der Nachteile, ohne auf die Vorteile der linearen Rekursion zu verzichten. Ein Vertreter dieser Gattung stellt der Lagged-Fibonacci-Generator dar, der im folgenden Abschnitt vorgestellt werden soll.

Lagged-Fibonacci-Generator

Ein weiter Vertreter der Klasse von Generatoren die auf die Methode der linearen Kongruenzen vertraut ist der Lagged-Fibonacci-Generator.
Entwickelt wurde der Generator vom Lehrstuhl für Kommunikationsnetze der RWTH Aachen.

Einführung und Schreibweise

Bei der Erzeugung von Zufallszahlen durch den Lagged-Fibonacci-Generator LFG sind folgende Parameter zu wählen:

- M ... Modulus, die Menge $\{0, 1, ..., M-1\}$ stellt die Grundmenge des Generators dar
- d ... eine additive Konstante
- e_0 ... ein Startwert für eine Folge von Hilfswerten
- $(y_n)_{n=0}^{n\geq 97}$ ... die Seedfolge von mindestens 97 Werten, mit dem der Generator initialisiert wird.

Die Wahl obiger Parameter wird durch die Definition der Rekursionsvorschrift des Lagged-Fibonacci-Generators klar:

$z_{n+1} = (y_{n-97+1} - y_{n-33+1}) \pmod{M}$
$e_{n+1} = (e_n - d) \pmod{M}$
$y_{n+1} = (z_{n+1} - e_{n+1}) \pmod{M}$

Gemäß oben vereinbarter Schreibweise ist y_{n+1} die aktuell zu erzeugende Zufallszahl, z_{n+1} und e_{n+1} hierfür benötigte Hilfswerte.
Dieser Generator wird durch die Schreibweise $LFG(e_0, d, M, (y_n)_{n=0}^{N})$ eindeutig charakterisiert.
Die übliche Normalisierung $x_n = \frac{y_n}{M}$ erzeugt daraus dann die gewünschte Sequenz $(x_n)_{n\geq 0}$ in $U([0;1[)$.

Eigenschaften

Betrachtet man die Rekursionsvorschrift, so wird eines klar: Im Vergleich zum LCG ist der LFG zwar auch ein linearer, aber keineswegs ein einfacher Kongruenzgenerator, da zur Erzeugung einer Zufallszahl die vorangegangenen 97 Zufallszahlen parat sein müssen.
Zudem fällt auf, daß die multiplikative Konstante a des LCG hier nicht auftaucht, bzw. auf eins gesetzt ist. Die ausschließliche Verwendung der Addition in der Erzeugungsvorschrift ermöglicht grundsätzlich eine extrem schnelle Berechnung. Dies ist vor allem deshalb gegeben, da die Modulo-Rechnung bei einfacher Summenbildung auf einen Vergleich zurückgeführt werden kann. Dieser Vorteil wird teilweise durch den größeren Aufwand zur Verwaltung der Datenstruktur aufgefangen. Näheres dazu findet man im Kapitel 7 bei der Beschreibung der Implementierung.
Als weiteres Argument gegen den LFG wird oftmals der erhöhte Speicherplatzbedarf zur Abspeicherung von aktuellen Streams oder des Seeds angeführt. Bei der Abspeicherung von Streams ist es nötig, neben dem direkten Vorgänger, wie beim LCG, auch noch 96 weitere Zahlen aus der bisherigen Zufallszahlenfolge im Speicher unterzubringen. Selbst wenn man sich vor Augen hält, daß die verwendete Zahlendarstellung eines solchen Generators mindestens 4 Byte erfordert, und daß moderne Simulationen ohne weiteres 100 oder mehr Streams erfordern können, ist dieser Speicheraufwand bei modernen Rechnern völlig vernachlässigbar und macht sich in der Praxis keinesfalls bemerkbar.
Auch hinsichtlich der Reproduzierbarkeit der Zufallszahlenfolge ist der LFG genauso problemlos wie der LCG.
Es gibt einen signifikanten Unterschied in den Eigenschaften beider Generatoren. Auf verschiedenartige Auswahl von Teilstreams aus der Zufallszahlenfolge reagiert der Lagged-Fibonacci-Generator wesentlich unempfindlicher.
Verschieden Runs-Tests beweisen, daß die Qualität der Substreams nicht von deren Auswahl aus der Zufallszahlenfolge beeinflußt wird (vgl. [Go96]).

Nichtlineare Kongruenzgeneratoren

Wie schon im letzten Abschnitt erwähnt wurde nicht nur versucht auf direktem Weg den LCG zu 'tunen', sondern man wandte sich auch vollkommen neuen Erzeugungsmethoden für Zufallszahlen zu. Dies bedeutete im konkreten Fall eine Abkehr von der linearen, hin zur nichtlinearen Rekursion.
Bevor eine Form dieser neuen Erzeugungsart, die inverse Rekursion, konkret besprochen wird, seien zunächst einige Anmerkungen zu nichtlinearen Kongruenzgeneratoren erlaubt. Die Idee, nichtlineare Kongruenzen zur Zufallszahlenerzeugung einzusetzen, wurde erstmals von Eichenauer et. al. (vgl. [EGL88]) vorgestellt. Die durch lineare Rekursion verursachten Gleichmäßigkeiten in der LCG-Zufallszahlenfolge sollen hierdurch behoben werden.

Nichlineare Kongruenzgeneratoren

Nachdem es sich hier, ähnlich wie beim LCG um eine einfache Kongruenzmethode handelt, sieht die Erzeugung von Zufallszahlen durch einen nichtlinearen Generator (nonlinear congruential generator, NLCG) zunächst ganz ähnlich aus.
Wir wählen eine großen Primzahl M, etwa $M = (2^{31} - 1)$ und generieren dann eine Folge von natürlichen Zahlen $y_1, y_2, \ldots$ aus der Menge $\mathbb{N}_p = \{0, 1, \ldots, M-1\}$ durch die Rekursion

$y_{n+1} = f(y_n) \pmod{p}$

mit Hilfe einer nichtlinearen Funktion, die so gewählt wird, daß die Folge $y_0, y_1, \ldots$ periodisch mit minimaler Periodenlänge M ist. Diese Funktion stellt also eine Selbstabbildung der abgeschlossenene Menge $\mathbb{N}_p$ auf sich selbst dar. Anschließend werden nichtlineare kongruente Pseudozufallszahlen (x_n)wieder durch die Normalisierung $x_n = \frac{y_n}{M}$ erzeugt.
Ziel ist es also, eine derartige nichtlineare Funktion zu finden.

Inverser Kongruenzgenerator

Einführung und Schreibweise

Inverse Kongruenzgeneratoren verwenden hierfür die Berechnung der multiplikativen Inverse (mod M). Für ein $c \in \mathbb{N}_M$ definieren wir die mulitplikative Inverse $\overline{c}$ als

- im Fall von $c = 0$ sei $\overline{c} = 0$
- sonst muß gelten $c\overline{c} \stackrel{!}{=} 1 \pmod{p}$, wobei 1 hier für das Einselement, also das neutrale Element bzgl. der Multiplikation steht.(in $\mathbb{R}$ würde gelten: $\overline{c} = \frac{1}{c}$)

Bei der Erzeugung von Zufallszahlen durch einen inversen Kongruenzgenerator (inversive congruential generator, ICG) sind folgende Parameter zu wählen:

- M ... Modulus, $\{0, 1, \ldots, M-1\}$ stellt die Grundmenge des Generators dar
- a ... der Vorfaktor
- b ... die additive Konstante
- y_0 ... der Startwert, mit dem der Generator initialisiert wird.

Der inverse Kongruenzgenerator wird dann definiert durch die Rekursion

$y_{n+1} = a \cdot \overline{y_n} + b \pmod{M}$, $n \geq 0$.

Dieser Generator wird durch die Schreibweise $ICG(a, b, M, y_0)$ eindeutig charakterisiert.
Die übliche Normalisierung $x_n = \frac{y_n}{M}$ erzeugt daraus dann die gewünschte Sequenz $(x_n)_{n \geq 0}$ in $U([0;1[)$.

Das multiplikativ-inverse Element

Schon an der Rekursionsvorschrift ist erkennbar, daß die Berechungskomplexität im Vergleich zum LCG höher liegen wird. Verursacht wird dieser Anstieg durch die benötigte „Invertierung" der Zufallszahl.
Gesucht wird zu einer Zahl n multiplikativ-inverse Element n' über der Menge $\{0, 1, \ldots, M-1\}$, also die Zahl n', sodaß $n \cdot n' = 1 \pmod{M}$ gilt.
In der angegebenen Menge könnte dies zunächst durch die Vorschrift $\overline{c} = c^{M-2} \pmod{p}$ erfolgen, bei gewünschten Periodenlängen von $M \leq 2^{31}$ ist der dazu nötige Berechnungsaufwand keinesfalls akzeptabel.
Aus diesem Grund wird in der Praxis der Algorithmus von Euklid verwendet, der eingesetzt zur Ermittlung des größten gemeinsamen Teilers $ggT(n, M)$ als erfreuliches Nebenprodukt das multiplikativ inverse Element liefert. Dieser Algorithmus dürfte aus den Standardwerken der Analysis hinreichend bekannt sein (vgl. z.B. [He91]). Gegenüber anderen Verfahren, wie z.B. der Primfaktorzerlegung, hat dieser den Vorteil, sich automatisieren zu lassen (vgl. Kapitel 6 über die Implementierung der Generatoren).
Die durch die Inversion zusätzlich anfallende Zeiteinbuße ist damit im schlechtesten Fall um den Faktor drei höher als beim herkömmlichen LCG, wenn man die Anzahl der Multiplikationen als Maßstab wählt. Selbst unter dem Gesichtspunkt, daß umfangreiche Simulationen große Mengen von Zufallszahlen benötigen, und der Zufallszahlengenerator hier durchaus Einfluß auf die Laufzeit der Simulation haben kann, wird dieser geringfügig höhere Aufwand in der Praxis so gut wie nicht spürbar. Im praktischen Einsatz in mehreren Simulationsmodellen unterschiedlicher Komplexität und mit unterschiedlichen Parametern waren selbst im

schlechtesten Fall Verzögerungen von höchstens Faktor 2 im Vergleich zum konkurrierenden LCG hinzunehmen.
Hiermit ist das letzte Wort noch nicht gesprochen. Zusätzliche Verbesserungen lassen sich durch Verwendung von zusammengesetzten Generatoren (compound generators) und der digital inversen Methode erreichen. Die Idee, die hinter letzterem Verfahren steht ist die folgende:
Man führt die Berechnung von $\overline{c} = c^{M-2}$ auf die von $\left(c^{2^{\lfloor \frac{M}{2} \rfloor - 1}}\right)^2$ zurück, was durch die Identitäten

$$a^{2^m-1} = \left(a^{2^{\frac{m}{2}}}\right)^{2^{\frac{m}{2}}} a^{2^{\frac{m}{2}}-1} \text{ für gerades m}$$

$$a^{2^m-1} = \left(a^{2^{\frac{(m-1)}{2}}-1}\right)^{2^{\frac{(m+1)}{2}}} \left(a^{2^{\frac{(m-1)}{2}}-1}\right)^2 a \text{ für ungerades m}$$

erfolgen kann. Für detailliertere Informationen sei auf Arbeiten von Niederreiter (vgl. [Ni92] und [Ni..]) verwiesen.

Eigenschaften

In Bezug auf Speicherplatzbedarf und Reproduzierbarkeit von Zufallszahlenfolgen steht der ICG dem klassischen LCG in nichts nach. Auch die Frage der gewachsenen Berechnungskomplexität ist in der Praxis kein Problem, wie wir eben gesehen haben.
Die Frage, ob ein ICG volle Periode hat, läßt sich aufgrund des folgenden Theorems entscheiden (vgl. [We94]):

> **Theorem**: Ein ICG hat volle Periode dann und nur dann, wenn der Quotient aus den Wurzeln des Polynoms $x^2 - bx - a$ über $\mathbb{N}_M$ multiplikative Ordnung $(p+1)$ in der abgeschlossenen Menge mit M^2 Elementen hat.

Dies bedeutet, daß es die Forderung nach Primitivität des Polynoms $x^2 - bx - a$, eine hinreichende Bedingung für die volle Periodenlänge eines ICG ist.
Ist dies der Fall, so erreicht die Folge (x_n) alle rationalen Zahlen aus der Menge $U([0;1[)$ mit Nenner M.
Diese Forderung nach Primitivität eines Polynoms mag zunächst sehr kompliziert klingen, es gibt aber weitreichende Studien und Algorithmen solcher IMPs, einer Oberklasse der primitiven Polynome.
Vorgestellt wurde diese neue Klasse von Polynomen von Flahive und Niederreiter (vgl. [FlNi92]), der erste Algorithmus für das Auffinden von IMP-Polynomen über einer abgeschlossenen Menge wurde von Chou (vgl. [Ch93]) veröffentlicht. Hellekalek et. al (vgl. [He94] und [HMW94]) schließlich modifizierten diesen Algorithmus und führten Läufe durch, sodaß heute umfangreiche Parametertabellen für die Implementierung von inversen Generatoren zur Verfügung stehen. (siehe Anhang)
Die einfachen empirischen Tests auf Gleichverteilung und Unabhängigkeit meistert der ICG genauso exzellent wie der LCG. An dieser Stelle möchte sei der Leser wieder auf Kapitel 5 verwiesen, wo anhand von χ^2-Test und Autokorrelationstest diese Eigenschaften nochmals bestätigt wurden.
Im Gegensatz zum LCG sind aber auch Lattice-Tests für den ICG kein Problem. Betrachtet werden wieder die überlappenden s-Tupel

$z_n = (x_n, x_{n+1}, ..., x_{n+s-1})$ für n≥ 0

oder die nichtüberlappenden s-Tupel

$z_n = (x_{ns}, x_{ns+1}, ..., x_{ns+s-1})$ für n≥ 0

im s-dimensionalen Hyperraum $[0;1[^s$.
Sind die zugrundeliegenden Zufallszahlen Realisationen einer unabhängigen, $U([0;1[)$-gleichverteilten Zufallsvariable, so unterliegen die Vektoren z_n in jedem Fall einer Gleichverteilung im Hyperraum $U([0;1[^s)$. Für ein sample $P = (z_n)_{n=0}^{N-1}$ kann die Analyse sowohl mit zahlentheoretischen als auch mit statistischen Methoden erfolgen, um die vorliegende empirische Verteilung zu ermitteln.
In der Zahlentheorie nennt man diesen Test „star discrepancy" (vgl. [Ni92] und [Ni..]). Dieser Test erlaubt sogar eine Größenordnungabschätzung der Diskrepanz und die Ergebnisse sind zuverlässiger, da sie normalerweise für die ganze Periode des Generators gelten und nicht nur für die kleinen Teile, die in der Praxis benutzt werden. Der ICG überzeugt in dieser Disziplin vollständig.
Diese o.g. Untersuchung förderte noch eine weitere interessante Eigenschaft zu Tage: die Unabhängigkeit der erzeugten Zufallszahlen ist von der Wahl der Parameter völlig unabhängig. Mit anderen Worten: Wurden die Parameter a und b so gewählt, daß volle Periodenlänge induziert wird, sind diese hervorragenden statistischen Eigenschaften immer gewährleistet.
Natürlich können theoretische Tests, wie der letztgenannte, niemals garantieren, daß diskrete Proben der Zufallszahlenfolge dieses Generators auch wirklich den entsprechenden statistischen Tests genügen. Dies liegt

daran, daß man in den theoretischen Tests das Verhalten sehr langer Proben analysiert, während empirische Tests relativ kurze Folgen untersuchen (vgl. dazu Abschnitt 3.1 aus [He95]).
Aus diesem Grund soll ein empirischer Test, der, wie man sehen wird, das obige Ergebnis bestätigt, folgen. Eine geeignete statistische Methode hierfür ist der der klassische „goodness-of-fit Test", der zweiseitige Kolmogoroff-Smirnov-Test. Bekanntermaßen vergleicht man hierbei direkt die empirische mit der erwarteten Verteilung der z_n. Hier bestätigt der ICG die obigen Ergebnisse und passiert den Test ohne Probleme (vgl. [Ni92] und [Ei94]).
Eine interessante Untersuchung zu diesem Thema wurde von Peter Hellekalek auf der Basis des M-Tupel-Tests durchgeführt (vgl. [He95]).
Eichenauer-Herrmann hat in seiner gleichnamigen Veröffentlichung aus dem Jahre 1991 gezeigt: „Inversive congruential random number avoid the planes". (vgl. [Ei91])
Dieses Ergebnis wird bestätigt durch Arbeiten von Niederreiter (vgl. [Ni92] und [Ni..]), der durch umfangreiche Tests zeigte, daß der ICG Lattice-Tests in Dimensionen besteht, die für den LCG völlig undenkbar wären.
Auch im Bereich der „long-range"-Korrelation ist der ICG überlegen, wie Tests von Entacher (vgl. [En..a] und [En..b]) belegen. Das Testszenario ist dabei wieder das gleiche, wie beim Runs-Test des LCG. Die beiden horizontalen Linien repräsentieren wieder die kritischen Werte der Testgröße des Kolmogoroff-Smirnov-Tests für ein Konfidenzniveau von 0.05 bzw. 0.01, die x-Achse veranschaulicht wieder die Größe der Proben. Betrachtet wird wieder die Teilfolge (x_{77n}).

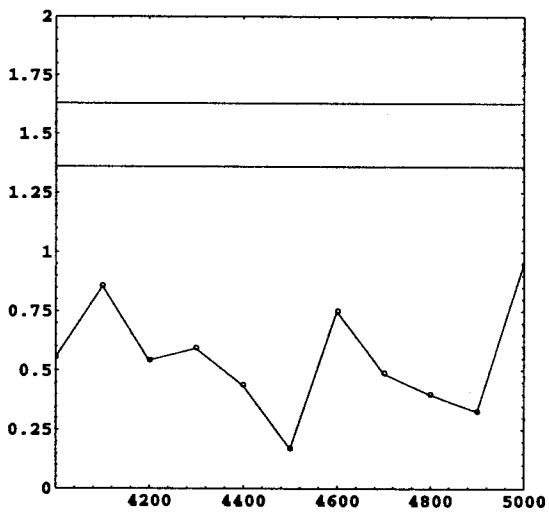

Bild 5
$ICG(1, 1, 1, 2^{31} - 1)$ „short ICG"

Der Unterschied wird beim Vergleich mit der Grafik des LCG deutlich.
Für eine detailliertere Untersuchung der Generatortypen mit dem Runs-Test sei der interessierte Leser auf o.g. Literaturstellen verwiesen.

Explizit inverser Kongruenzgenerator

Ein weiterer Generator der nach dem gleichen Prinzip arbeitet, ist der explizit inverse Kongruenzgenerator (explicit inverse congruential generator, EICG). Auch er benutzt das die Berechnung der multiplikativen Inversen um die o.g. Nachteile des LCG zu überwinden. Der Unterschied zum ICG liegt darin, daß der EICG kein rekursiver Generator ist.

Einführung und Schreibweise

Bei der Erzeugung von Zufallszahlen durch den explizit inversen Kongruenzgenerator (explicit inversive congruential generator, EICG) sind folgende Parameter zu wählen:

- M ... Modulus, $\{0, 1, ..., M-1\}$ stellt die Grundmenge des Generators dar
- a ... der Vorfaktor
- b ... die additive Konstante
- n_0 ... der Startindex $n_0 \in \{0, 1, ..., M-1\}$, der zugleich Index der ersten erzeugten Zufallszahl der Zufallszahlenfolge (x_n)ist.

Der explizit inverse Kongruenzgenerator wird dann definiert durch

$$y_n = \overline{a \cdot (n - n_0) + b} \pmod{M},\ n \geq 0$$

Dieser Generator wird durch die Schreibweise $EICG(a, b, M, n_0)$ eindeutig charakterisiert.
Die übliche Normalisierung $x_n = \frac{y_n}{M}$ erzeugt daraus dann die gewünschte Sequenz $(x_n)_{n \geq 0}$ in $U([0;1[)$.

Eigenschaften

Nachdem der EICG quasi den kleinen Bruder des ICG darstellt, werden seine Eigenschaften hier nur noch kurz zusammengefaßt, und der interessierte Leser möge für weiterführende Informationen die entsprechend angegebene Literatur zu Rate ziehen.
Zunächst fällt auf, daß die additive Konstante eigentlich überflüssig ist. Die Generatoren $EICG(a, b, M, n_0)$ und $EICG(a, 0, M, m_0)$ mit $m_0 = n_0 + \overline{a}b \pmod{M}$ produzieren identischen Output.
Die maximal mögliche Periodenlänge ist M. Um einen Generator mit maximaler Periode zu erhalten, ist es aus gerade erläuterten Eigenschaften, lediglich nötig, den Parameter $a \neq 0$ und M als Primzahl zu wählen.
Reproduzierbarkeit und Berechnugskomplexität sind, ähnlich wie beim ICG, keinerlei Hindernisse dar. Auch hier bietet sich natürlich die Möglichkeit von zusammengesetzten Generatoren, bzw. der Einsatz der digital-inversen Methode (vgl. Abschnitt über den ICG) an, falls zusätzliche Leistungssteigerungen erwünscht sind.
Lattice-Test und „long-range"-Korrelationen stellen für den EICG, aus den gleichen Gründen wie beim ICG, keinerlei Schwierigkeiten dar. Fundiert wird diese Tatsache durch weitreichende empirische Tests, im Bezug auf Lattice-Tests durch die Arbeiten von Niederreiter (vgl. [Ni94]), im Fall der Korrelationen durch Eichenauer-Herrmann und Niederreiter (vgl. [Ni94], [Ei93] und [EiNi94]), Tests wie z.B. Runs-Tests wurden von Hellekalek durchgeführt (vgl. [He95]).
Im Bezug auf Art und Größe von Substreams gelten für den EICG die gleichen Aussagen wie für den ICG.

Sonstige Generatoren

Puran-2-CD

Eine völlig Abkehr von herkömmlichen Methoden stellt der letzte hier vorzustellende Zufallszahlengenerator dar. Hierbei werden die Zufallszahlen nicht durch mathematische Methoden berechnet, sondern physikalisch gemessen und abgespeichert. Im vorliegenden Fall diente dazu eine CD.
In der ersten Ausgabe der Puran-CD wurden dabei Messungen aus dem radioaktiven Zerfall benutzt, zeigten nicht die gewünschte Qualität der Ergebnisse. Aus diesem Grund und um die komplizierten Sicherheitsmaßnahmen bei Messungen aus radioaktivem Zerfall möglichst zu vermeiden, wurde bei der zweiten Fassung der Puran-CD eine andere Datenquelle, nämlich ein Semiconductor Random Noise Generator gewählt.

Einleitung

Folgende Skizze (entnommen aus der Beschreibung der Puran-2-CD) veranschaulicht den Versuchsaufbau.

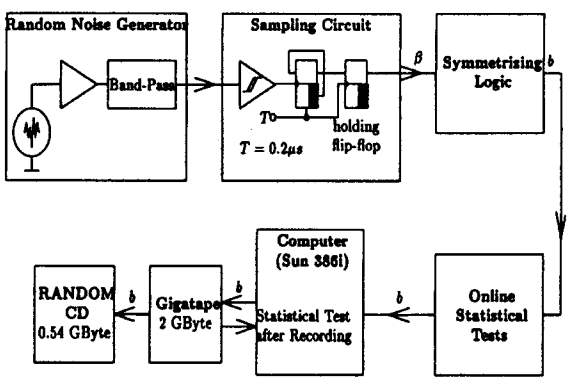

Bild 6

Das Signal des Tongenerators wurde durch den Bandpass auf ein Frequenzspektrum von $0..13GHz$ gefiltert.
Durch schnelle GaAs-Flip-Flops, Verwendung nur jedes 20. Bits des holding flip-flops und lange Meßdauer wurden evtl. Einflüsse des Sampling Circuit auf die Güte der Ergebnisse verhindert.
Durch Veränderung des Bit-Streams wurde mit Hilfe folgender Regel

$\beta_i\beta_{i+1}$	01	10	11	00
b_j	0	1	--	--

Tabelle 1
Bit-Operationen zur Erzeugung
eines symmetrischen Musters

aus dem zunächst unsymmetrischen Muster (β_i) ein symmetrisches Muster (b_j) gemacht.
Als Konsequenz aus den Erfahrungen mit der ersten Puran-CD wurde die Aufnahme der Random-Bits ständig durch statistische Tests überwacht. Ein Teil, etwa 0.54 Gbyte, der gemessenen 2 Gbyte an Zufalls-Bits wurde dann auf der Puran-2-CD gespeichert.
Weiterführende Informationen über Versuchsanordnungen dieser Art finden sich in [Ri92], die verwendete Datenstruktur und sonstige technische Details entnehme man dem Abschnitt über die Implementierung der Generatoren.

Eigenschaften

Durch weitreichende theoretische und empirische Untersuchungen wurde die Qualität der Zufallszahlenfolge dokumentiert (vgl. [Sc86] und [Sc87]), zusätzlich dazu sei der Leser auf das folgende Kapitel über den Vergleichstest der Generatoren verwiesen, wo anhand zweier ausgewählter Tests die Eigenschaften der erzeugten Zufallszahlenfolge hinsichtlich Gleich- verteilung und Korrelation noch einmal überprüft wurden.
Über die guten empirischen Eigenschaften sollte man eines nicht vergessen: Der eingeschränkte Platz einer CD der heutigen Generation (ISO 9660) von etwa 650 MB läßt in Verbindung mit der 32-bit Repräsentation der Zufallszahlen, nur sehr geringe Periodenlängen zu. Um Lesefehler des CD-Laufwerkes korrigieren zu können, wird bei der Puran-2-CD der vorhandene Platz durch Speicherung von Parity-bits noch weiter eingeschränkt.

Zusammenfassung

Der Lagged-Fibonacci-Generator stellt also eine durchaus gelungene Weiterentwicklung der linearen Kongruenzmethode dar. Durch geschickte Wahl der Rekursionsvorschrift ist es gelungen einen der wesentlichen Nachteile des LCG zu beseitigen.
Inwieweit sich der Lagged-Fibonacci-Generator etablieren wird, wird vor allem darauf ankommen, wie gut er das zweite wesentliche Problem der LCGs, die regelmäßige Lattice-Struktur in höherdimensionalen Räumen, in den Griff bekommen kann. Bei Abschluß dieser Arbeit waren mir hierzu keine Erkenntnisse zugänglich.
Zusammenfassend ist zu sagen, daß die oben erwähnten Generatoren eine klare Bereicherung auf dem Sektor der Zufallszahlengenerierung versprechen. ICG und EICG bieten für einen geringfügig höheren Berechnungsaufwand empirische und strukturelle Vorteile gegenüber dem LCG. Wie in der Einleitung bereits angemerkt, wird es keinen Zufallszahlengenerator geben, der alle Anforderungen der Simulationstechnik zu lösen vermag. Sicherlich wird man in Zukunft Spezialfälle finden, die die inversen Generatoren vor Probleme stellen. Deshalb ist es wichtig diese neuartigen Generatoren nicht als Konkurrenz, sondern als Bereicherung zu klassischen Methoden der Zufallszahlenerzeugung, wie dem LCG, zu sehen.
Schließlich kann es nur von Vorteil für die Güte von Simulations- und Analyseergebnissen sein, wenn die Möglichkeit zur Verifikation durch Läufe mit verschiedenen Zufallszahlengeneratoren gegeben ist.

Vergleich der Generatoren

Expect the unexpected!
- Douglas Adams, The Hitch Hiker's Guide to the Galaxy

Obwohl niemand die Notwendigkeit von Tests zur Ermittlung der Eigenschaften von Zufallszahlengeneratoren leugnen kann, gehen die Meinungen über die Aussagekraft solcher Verfahren sehr weit auseinander. Betrachtet man einige Veröffentlichung, so glaubt man sich in den Bereich der Esoterik versetzt. Die Meinungsverschiedenheiten beginnen schon bei der essentiellen Frage, ob denn nun theoretische oder empirische Tests das richtige Mittel wären. Theoretische Tests eines Pseudozufallszahlengenerators beschäftigen sich aufgrund der benötigten mathematischen Grundlagen mit sehr großen Proben, in der Regel also mit der gesamten Periode des Generators. In empirischen Tests konzentriert man sich auf die statistische Analyse relativ kleiner Samples aus der Zufallszahlenfolge.
Leider ist die Mathematik bis heute die Integration beider Verfahrensweisen schuldig geblieben, d.h. die theoretische Analyse großer Folgen von Zufallszahlen läßt keinerlei Rückschlüsse auf die Ergebnisse der empirischen Tests des gleichen Generators zu und umgekehrt.
Trotzdem haben sich in über vier Jahrzehnten einige Verfahren zum Test von Pseudozufallszahlengeneratoren zumindest soweit etabliert, daß sie allgemein als verläßliche Indikatoren gelten. Aus der Sicht der theoretischen Analyse wäre hier z.B. der Spektral-Test zu nennen, als verläßliche empirische Tests gelten unter anderem die beiden hier vorgestellten:

- der χ^2 –Test, als ein typischer Test auf Gleichverteilung
- der Autokorrelationstest.

Da beide Tests als weitverbreitete Standardverfahren gelten können, wird im folgenden auf eine nochmalige Beschreibung dieser Tests weitgehend verzichtet. Die Auswertung der Tests wurde mit Hilfe des Programmpakets Octave (vgl. [Ea95]) durchgeführt.

Test auf Gleichverteilung

Testszenario

Der χ^2-Test wurde bereits im Jahr 1900 von K. Pearson veröffentlicht. Als verwendete Parameter sind:

- $x(i)$ die erwartete Verteilung, die mit der gemessenen verglichen werden soll.
- n die Größe der gemessenen Probe („Sample").
- k die Anzahl der Intervalle; der Definitionsbereich der Verteilung wird also zerlegt in $[a_0, a_1), [a_1, a_2), \ldots, [a_{k-1}, a_k)$.
- N_j die Anzahl von Zufallszahlen aus dem gewählten Sample, die in das j-te Intervall, also in $[a_{j-1}, a_j)$ fallen, es gilt $\sum_{j=1}^{k} N_j = n$.
- P_j die anhand der erwarteten Verteilung ermittelte Anzahl von Zufallszahlen, die in das j-te Intervall fallen sollten, im diskreten Fall gilt also $P_j = \sum_{i=a_{j-1}}^{a_j} x(i)$.
- χ^2 die Testgröße, definiert durch $\chi^2 = \sum_{j=1}^{k} \frac{(N_j - nP_j)^2}{nP_j}$, die im Falle einer Übereinstimmung der beiden Funktionen einer χ^2-Verteilung mit n-1 Freiheitsgraden folgt.
- H_0 die Nullhypothese: Die gemessenen Zufallszahlen sind Realisationen einer $x(i)$ –verteilten Zufallsvariable.

Für ein gegebenes Konfidenzniveau α wird die Nullhypothese H_0 abgelehnt, falls

$$\chi^2 > \chi^2_{k-1,1-\alpha}$$

und nicht abgelehnt falls

$$\chi^2 \leq \chi^2_{k-1,1-\alpha}.$$

Im vorliegenden Fall soll ein Test auf Gleichverteilung im Intervall [0; 1) durchgeführt werden.

Als wohl ältester „goodness-of-fit"-Test eignet sich dieses Verfahren für einen solchen Test aus zwei Gründen besonders gut:

- Die erwartete Verteilung $x(i)$ ist vollständig bekannt, d.h. es muß kein einziger Parameter geschätzt werden. Dies erspart uns die Frage nach der Nichtablehnung der Nullhypothese im Bereich $\chi^2_{k-1,1-\alpha} \leq \chi^2 \leq \chi^2_{k-m-1,1-\alpha}$, falls m Parametern zu schätzen gewesen wären.
- Für sinnvolle Aussagen braucht man auf jeden Fall einen „unbiased" Schätzer, d.h. die Signifikanz muß größer sein als die Wahrscheinlichkeit für einen Fehler vom Typ I. Um dies beim χ^2-Test sicherstellen zu können, ist es nötig den „equiprobable approach" zu wählen, d.h. die Intervalle $[a_{j-1}, a_j)$ für $1 \leq j \leq k$ so festzulegen, daß $P_1 = P_2 = ... = P_k$ auch für den diskreten Fall annähernd gilt. Bei einer erwarteten Gleichverteilung bedeutet dies aber nichts anderes, als daß das Intervall $[0; 1)$ in k äquidistante Teile zu zerlegen

Dazu wurde ein Wert von $k = 2^{12} = 4096$ gewählt. Dieser Wert erwies sich als geeignet, da weitere Verfeinerung die Signifikanz des Tests nicht verbessert. Zudem müssen von jeder Zufallszahl nur die „most-significant" 12 Bit ausgewertet werden, was einen entsprechend schnellen Test ermöglicht. Das für einen zuverlässigen Test notwendige Verhältnis von $\frac{n}{k} \geq 5$ wird durch die im vorliegenden Fall gewählte Samplegröße in Höhe von $2^{18} = 262144$ weit überschritten ($\frac{n}{k} = \frac{262144}{4096} = 64$). Da auch der absolute Wert von k im vorliegenden Fall groß genug gewählt wurde, konnte folgende Approximation benutzt werden:

$$\chi^2_{k-1,1-\alpha} \approx (k-1) \cdot \left[1 - \frac{2}{9 \cdot (k-1)} + z_{1-\alpha} \sqrt{\frac{2}{9 \cdot (k-1)}}\right]^3$$

wobei $z_{1-\alpha}$ der obere $1-\alpha$ kritische Punkt einer $N(0;1)$ –Verteilung ist.
Für das Konfidenzniveau wurden die Werte $\alpha = 0.10$und $\alpha = 0.05$ gewählt.

Ergebnisse

Die benötigten kritischen Punkte aus der Standard-Normalverteilung sind:

kritischer Punkt	$z_{1-\frac{\alpha}{2}}$
Konfidenzniveau $\alpha = 0,10$	1,282
Konfidenzniveau $\alpha = 0,05$	1,645

Tabelle 2
kritische Punkte

Dies führt zu den folgenden Quantilen:

Quantil	$\chi^2_{4095,0.90}$	$\chi^2_{4095,0.95}$
Werte	4211,40	4145,59

Tabelle 3
Quantile

Die Ergebnisse dieses Tests lassen sich anhand folgender Tabelle veranschaulichen

Generator	LCG	LFG	ICG	EICG	Puran
χ^2 –Teststatistik	4097,30	4069,31	4058,69	4121,10	4147,70

Tabelle 4
Ergebnisse

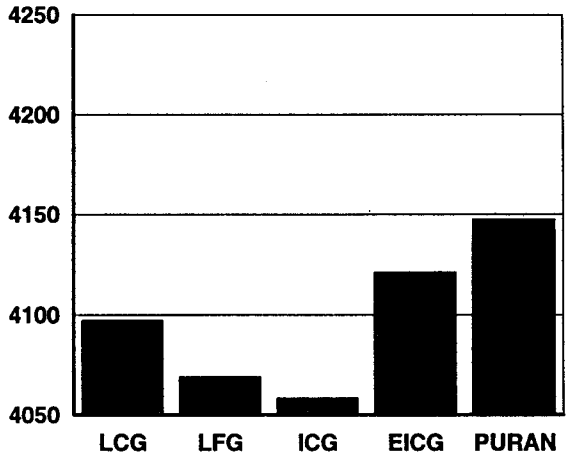

Bild 7
Ergebnisse des χ^2 –Tests

Die zugehörige graphische Darstellung zeigt deutlich, daß bei Konfidenzniveaus von $\alpha = 0,1$ und $\alpha = 0,05$ alle Generatoren gute Ergebnisse liefern. Einzig der Puran-Generator fällt in dieser Disziplin etwas ab und verfehlt bei einem Konfidenzniveau von $\alpha = 0,05$ das Ziel knapp.

Test auf Korrelation

Testszenario

Der Autokorrelationstest ist die einfachste und direkteste Art, Korrelationen zwischen Paaren von Zufallsvariablen zu erkennen. Hierzu werden folgende Parameter benötigt:

- $X_1, X_2, ..., X_n$ sei eine Folge von n Zufallsvariablen
- C_j ist die Kovarianz zwischen Folgengliedern im Abstand j, definiert durch $C_j = Cov(x_i, x_{i+j}) = E(x_i x_{i+j})$
- C_0 ist die Varianz der Zufallsvariable X_i
- $\rho_j = \frac{C_j}{C_0}$ ist dann die Korrelation von der Zufallsvariablen im Abstand j („lag" j)

In unserem Fall ist $X_i = x_i$ zu setzen, jede generierte Zufallszahl wird also als Ausprägungen einer gleichverteilten Zufallsvariable angesehen. Natürlich wird die ein kovarianz-stationäres System angenommen, d.h. Mittelwert und Varianz sind über die gesamte Zeitdauer hinweg als konstant anzusehen. Unter der Hypothese, daß die x_i's gleichverteilt in $U([0;1[)$ sind, gilt:

- $E[x_i] = \frac{1}{2}$
- $Var[x_i] = \frac{1}{12}$

und damit nach obiger Formel

- $C_j = E(x_i x_{i+j}) - \frac{1}{4}$
- $C_0 = \frac{1}{12}$

Also gilt

$$\rho_j = 12 \cdot E(x_i x_{i+j}) - 3.$$

Bei einer Folge von generierten Zufallszahlen kann der Korrelationsschätzer $\hat{\rho}_j$ also direkt durch den Mittelwertschätzer berechnet werden. Es ergibt sich damit:

$$\hat{\rho}_j = \frac{12}{h+1} \sum_{k=0}^{h} x_{1+kj} \cdot x_{1+(k+1)\cdot j} - 3 \text{ mit } h = \left\lfloor \frac{n-1}{j} \right\rfloor - 1$$

Für lange Folgen von Zufallszahlen ist obige Berechnung von $\hat{\rho}_j$ allerdings sehr zeitaufwendig. Der Autokorrelationskoeffizient wurde daher durch Verwendung der Fast-Fourier-Transformation berechnet (vgl. [Ro97]).

Unter der weitere Annahme, daß die x_i's unabhängig voneinander wären, also insbesondere $\rho_j = 0$ wäre, würde gelten

$$Var(\hat{\rho}_j) = \frac{13 \cdot h + 7}{(h+1)^2}$$

und damit wäre die normalverteilte Testgröße

$$A_j = \frac{\hat{\rho}_j}{\sqrt{Var(\hat{\rho}_j)}}$$

sogar standard-normalverteilt.

Die Nullhypothese H_0, daß die Korrelation $\rho_j = 0$ ist, wird also zu gegebenem Konfidenzniveau α abgelehnt, falls die Testgröße $|A_j| > z_{\frac{1-\alpha}{2}}$ ist. $z_{\frac{1-\alpha}{2}}$ ist dabei wieder das $\frac{1-\alpha}{2}$ –Quantil der $N(0;1)$ –Verteilung.

Im vorliegenden Fall wurden die gesamte Zufallszahlenfolge, sowie die Substreams $(x_{5 \cdot n})$ bis $(x_{10 \cdot n})$ jedes Generators, bezüglich Korrelation für lags $1, .., 20$ untersucht, die Sample-Größe betrug wieder $2^{18} = 262144$, die betrachteten Konfidenzniveaus waren 0.1 und 0.05. Die Ergebnisse dieses Test, angewandt auf die genannten Generatoren ist auf den folgenden Seiten dargestellt.

Ergebnisse

Die Darstellung der Autokorrelationsfunktion für die gesamte Folge ergibt sich wie folgt:

	LCG	LFG	ICG	EICG	Puran
Lag 0	1,0000E+00	1,0000E+00	1,0000E+00	1,0000E+00	1,0000E+00
Lag 1	2,8291E-04	-1,5095E-03	-1,6503E-03	-1,9747E-03	-1,1779E-03
Lag 2	1,2360E-03	9,5553E-04	1,0233E-03	1,1650E-03	1,6951E-03
Lag 3	-9,8549E-05	-1,4860E-03	1,1612E-03	5,1249E-04	-5,1049E-04
Lag 4	1,3841E-03	9,5586E-04	-1,0570E-03	-2,9810E-03	-3,3548E-03
Lag 5	-5,1391E-04	-7,4669E-04	1,4864E-03	-3,5174E-03	-2,3197E-03
Lag 6	-1,6678E-03	9,4755E-04	2,4976E-03	1,5925E-03	4,9906E-04
Lag 7	1,6038E-03	-3,4491E-03	-2,5676E-04	-1,8712E-03	-5,7327E-04
Lag 8	3,2353E-03	2,4788E-03	1,5241E-03	-1,6631E-03	2,3519E-03
Lag 9	-2,9832E-03	-3,4533E-03	1,3085E-03	7,3252E-04	-3,0741E-03
Lag 10	-4,2835E-04	9,4515E-04	-8,7698E-04	-1,6114E-03	2,9514E-03

Tabelle 5
Autokorrelationsfunktion

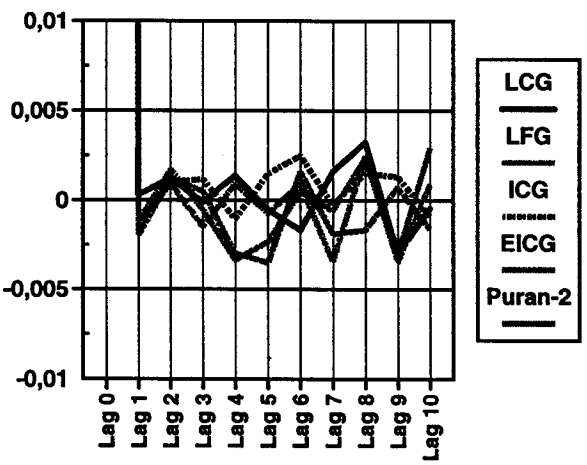

Bild 8
Autokorrelationsfunktion

Schon an der Autokorrelationsfunktion wird sichtbar, daß keine der getesteten Zufallszahlenfolgen merkliche Korrelation aufweist. Der Plot der Testgröße bestätigt dies eindeutig, da im Vergleich zu den Quantilen aus der $N(0;1)$ –Verteilung, keiner der Werte signifikant von der 0 abweicht.

	LCG	LFG	ICG	EICG	Puran
Lag 0	1,0000E+00	1,0000E+00	1,0000E+00	1,0000E+00	1,0000E+00
Lag 1	3,3713E-03	-1,7990E-02	-1,9670E-02	-2,3530E-02	-1,4040E-02
Lag 2	1,2390E-02	9,5750E-03	1,0250E-02	1,1670E-02	1,6990E-02
Lag 3	-8,9260E-04	-1,3460E-02	1,0520E-02	4,6420E-03	-4,6240E-03
Lag 4	1,3570E-02	9,3690E-03	-1,0360E-02	-2,9220E-02	-3,2880E-02
Lag 5	-4,0950E-03	-5,9510E-03	1,1850E-02	-2,8030E-02	-1,8490E-02
Lag 6	-1,2700E-02	7,2150E-03	1,9020E-02	1,2130E-02	3,8000E-03
Lag 7	1,1530E-02	-2,4790E-02	-1,8450E-03	-1,3450E-02	-4,1200E-03
Lag 8	2,2870E-02	1,7520E-02	1,0770E-02	-1,1760E-02	1,6630E-02
Lag 9	-2,0410E-02	-2,3620E-02	8,9510E-03	5,0110E-03	2,1030E-02
Lag 10	-2,8700E-03	6,3340E-03	-5,8770E-03	-1,0800E-02	1,9780E-02

Tabelle 6
Plot der Testgröße

Im Vergleich dazu die Quantile der $N(0;1)$ –Verteilung:

kritischer Punkt	$z_{1-\frac{\alpha}{2}}$
Konfidenzniveau $\alpha = 0,10$	1,282
Konfidenzniveau $\alpha = 0,05$	1,645

Tabelle 7
Quantile

Um eine bessere Übersichtlichkeit zu gewährleisten wurde der Wertebereich der folgenden Grafik auf ein sinnvolles Maß eingeschränkt. Die weit oberhalb und unterhalb liegenden Quantile sind deshalb nicht eingezeichnet.

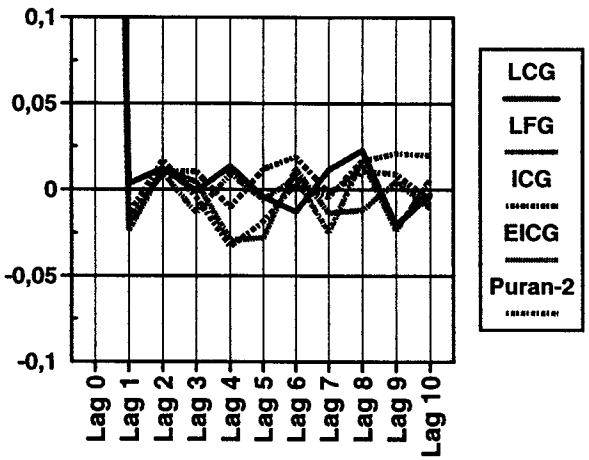

Bild 9
Testgröße A_j

Die Teilfolgen erzielten die gleichen guten Ergebnisse, von der Darstellung aller dieser Kurven wurde aus Gründen der Übersichtlichkeit Abstand genommen.
Dies bedeutet, daß keiner der Generatoren im getesteten Bereich Korrelationen aufweist. Wir haben bereits oben, bei der Vorstellung des LCG gesehen, daß aber auch „long-range"-Korrelationen auftreten können, die in diesem Test natürlich nicht berücksichtigt wurden.

Fazit

Natürlich haben die Ergebnisse obiger Standardtests, wie bereits wiederholt betont, nur ein begrenztes Aussagevermögen für eine Gesamtbeurteilung der einzelnen Generatoren. Zusammen mit den in der Einleitung aufgeführten Eigenschaften kristallisiert sich folgendes Bild heraus: Entgegen dem am Kapitelanfang gebrauchten Zitat haben sich die bereits in der Einführung genannten Qualitäten der getesteten Generatoren weitgehend bestätigt. Keiner der getesteten Generatoren weißt in der produzierten Zufallszahlenfolge nennenswerte Korrelation auf. Bis auf den Puran-2-Generator ist auch keine signifikante Abweichung von der Gleichverteilung erkennbar. Nachdem letztgenannter Generator auch in der maximalen Periodenlänge gegenüber den anderen Generatoren etwas abfällt, ist er für die Simulation umfangreicher Systeme nur bedingt geeignet.
Abschließend läßt sich feststellen, daß die neuen Generatoren in jeder Beziehung eine Bereicherung für die Praxis der Simulationstechnik darstellen. Auch wenn der LCG wohl auch in Zukunft das „Arbeitspferd" der Simulationstechnik bleiben wird, so sind die neuen Generatoren doch eine sinnvolle Erweiterung für Gebiete, wo der konventionelle LCG aufgrund seiner o.g. Schwächen nur bedingt oder überhaupt nicht einsetzbar ist. Auch der Einsatz der neuen Generatoren im Bereich der Verifikation von Simulationsergebnissen sollte dabei nicht unterschätzt werden. Simulationsläufe mit den neuen Generatoren erleichtern die Eliminierung und Entdeckung von Effekten, die durch den Generator erst erzeugt wurden bedeutend.
Nach der Präsentation dieser Ergebnisse soll nun noch kurz auf die Implementierung der Generatoren eingegangen werden.

Implementierung der Generatoren

Zielsetzung

Von Anfang an stand fest, diese Generatoren nicht nur zu implementieren, um die für die Tests notwendigen Outputs zu erhalten. Erwünscht war außerdem die Einbindung dieser neuen Generatoren in das Programmpaket Delphi, um deren Qualitäten auch in großen Simulationsprojekten zu testen.

Programmpaket Delphi

Das Simulationspaket Delphi wurde zur Simulation umfangreicher Warteschlangensysteme entwickelt. Heute wird Delphi unter dem Namen „Factory Explorer" von seinem Entwickler F. Chance kommerziell vertrieben, für Informationen hierzu siehe *www.chances.com.*
Die letzte, für Universitäts- und Lehrzwecke frei erhältliche Version von Delphi, Version 8, Level 29, basiert auf der Programmiersprache C (vgl. [Ch91]). Um die Einbindung möglichst reibungslos zu gestalten, wurde deshalb bei der Programmierung der Generatoren ebenfalls auf die Vorteile der objektorientierten Programmierung verzichtet.
Delphi war in der vorliegenden Version nur mit einem linearen Kongruenzgenerator ($LCG(2^{31}-1, a, b, y)$) ausgerüstet. Die zur Verfügung gestellten Funktionen im Bezug auf den Zufallszahlengenerator erklärt umseitiger Auszug aus dem Delphi-Code:

```
Usage: (Three functions)

        0. Before executing any of the following functions,
           execute
               kinit(last_stream)
           where last_stream is any number from 1 to 21,474.  This
           initializes the seeds for each of the streams from
           1 to last_stream.

        1. To obtain the next U(0,1) random number from stream "stream,"
           execute
               u = krand(stream);
           where krand is a double function.  The double variable u will
           contain the next random number.

        2. To set the seed for stream "stream" to a desired value zset,
           execute
               krandst(zset, stream);
           where krandst is a void function and zset must be a long set to
           the desired seed, a number between 1 and 2147483646 (inclusive).
           Default seeds for all 1000 streams are given in the code.

        3. To get the current (most recently used) integer in the sequence
           being generated for stream "stream" into the long variable zget,
           execute
               zget = krandgt(stream);
           where krandgt is a long function.
```

Folgende Funktionen stehen im Einzelnen dazu zur Verfügung

```
• long krandnt(void);                        /* Return Number of times called. */
• void kinit(int last_stream);               /* Initialize streams. */
• double krand(int stream);                  /* Generate next random number */
• void  krandst(long zset, int stream);      /* Set stream's current position. */
• long  krandgt(int stream);                 /* Get stream's current position. */
• void krand_save_streams(void);             /* Save stream information. */
• void krand_restore_streams(void);          /* Restore stream information. */
```

Aus diesem Grund war bis dato eine Datenstruktur in Form eines eindimensionalen Array of Long für Speicherungs- und Initialisierungszwecke völlig ausreichend. Der jeweilige Stream wurde über den Index des Array, die jeweils aktuelle Zahl des Streams über den zugehörigen Wert abgefragt.

Implementierung

Diese Datenstruktur erwies sich für die neuen Generatoren als völlig ungeeignet. Insbesondere der Fibonacci-Generator, der zur Erzeugung der nächsten Zufallszahl auf die Kenntnis der 97 letzten angewiesen ist, erfordert eine Datenstruktur, die es möglich macht mindestens jene letzten 97 Elemente eines Streams stets parat zu haben. Aus Effizienzgründen war auch in Hinblick auf den Puran-2-Generator eine solche Datenstruktur wünschenswert, da bei der geringen Zugriffsgeschwindigkeit auf ein Massenspeichermedium wie CD-drive oder Harddisk eine Pufferung der Daten unerläßlich ist. Um schließlich die gegebene Funktionalität voll beibehalten zu können, mußte die Möglichkeit geschaffen werden ganze Streams abzuspeichern bzw. zu laden.

gewählte Datenstruktur

Um diesen Anforderungen zu genügen und im Hinblick auf evtl. Erweiterungen möglichst flexibel zu sein, wurde eine doppelte Listenstruktur gewählt. Folgende Skizze gibt einen Überblick:

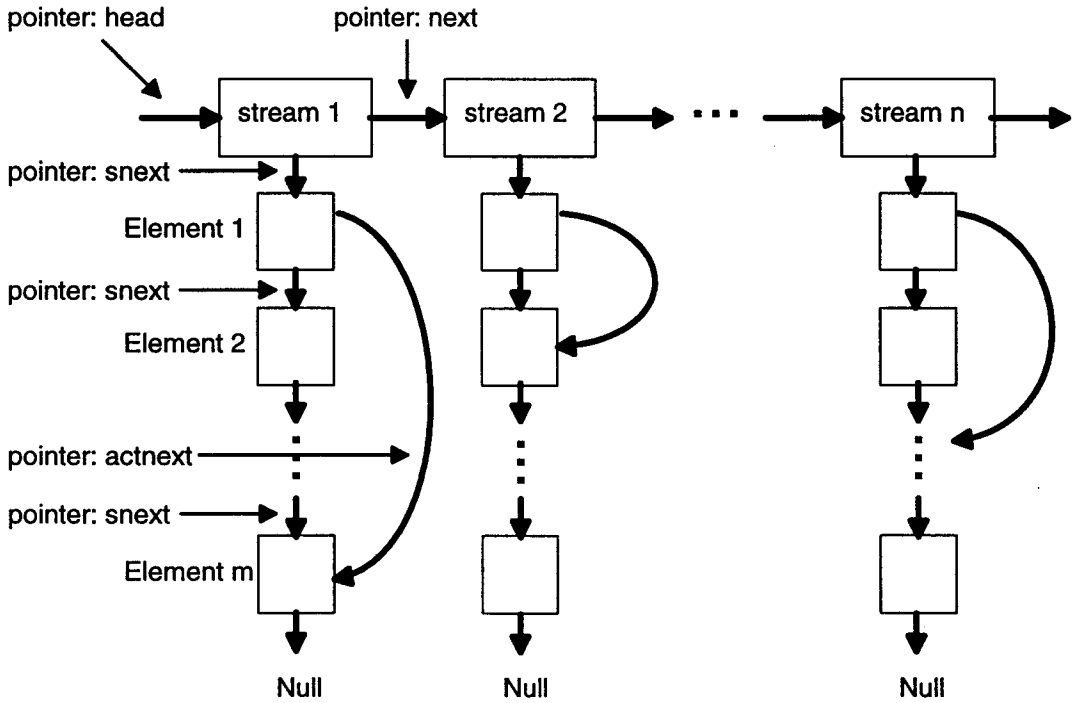

Bild 10
Datenstruktur

Die grundsätzliche Struktur wird dabei durch die Headerelemente *„stream 1"* bis *„stream n"* gebildet. Jedes Headerelement wurde neben der Streamnummer mit drei Pointern versehen. Der erste, bezeichnet als *„next"* stellt die Verbindung zum nächsten Stream dar, zeigt also wieder auf ein Headerelement. Die beiden anderen Pointer zeigen auf die Elemente des zugehörigen Streams. Der Pointer *„snext"* zeigt auf das aktuell erste Element des Streams, der Pointer *„actnext"* auf erste, noch nicht benutzte Element des Streams. Dieser Pointer muß also beim Auslesen einer Zufallszahl aus dem Stream jeweils weitergesetzt werden. Hierdurch wird der Aufwand zur Kennzeichnung oder Indizierung bereits gebrauchter Zufallszahlen des Streams eingespart, die Datenstruktur organisiert sich also weitgehend selbst.
Die einzelnen Streamelement bilden eine einfach verkettete Liste und enthalten jeweils den hierfür notwendigen *„snext"*-Pointer sowie die entsprechende Zufallszahl in *long*- und *double*-Repräsentation. Zudem wurde ein Feld für die Hilfsvariable des Lagged-Fibonacci-Generators vorgesehen.
Im folgenden soll nun nur noch kurz auf die Implementierung der einzelnen Generatoren eingegangen werden.

Linearer Kongruenzgenerator

Das Programmpaket Delphi war in der vorliegenden Version mit einem konventionellen LCG ausgerüstet. Dieser Generator wurde in dieser Form weitestgehend beibehalten, Änderungen wurden nur durchgeführt wo dies im Hinblick auf die neue Datenstruktur notwendig wurde.

Folgender Programmausschnitt erklärt die Arbeitsweise des Generators und die verwendeten Parameter:

```
/* constants for the linear congruential generator */
#define LCG_MODLUS          2147483647
#define LCG_MULT1           24112
#define LCG_MULT2           26143

/* compute next value from old 'value'*/
lowprd = (value & 65535) * LCG_MULT1;
hi31   = (value >> 16) * LCG_MULT1 + (lowprd >> 16);
value  = ((lowprd & 65535) - LCG_MODLUS) +
  ((hi31 & 32767) << 16) + (hi31 >> 15);
if (value < 0)
value += LCG_MODLUS;
lowprd = (value & 65535) * LCG_MULT2;
hi31   = (value >> 16) * LCG_MULT2 + (lowprd >> 16);
value  = ((lowprd & 65535) - LCG_MODLUS) +
  ((hi31 & 32767) << 16) + (hi31 >> 15);
if (value < 0)
value += LCG_MODLUS;
```

Lagged-Fibonacci-Generator

Der obigen Beschreibung des LFG ist eigentlich nichts hinzuzufügen, als einer kurzen Bemerkung zur Initialisierungssequenz.

Durch Verwendung zweier Generatoren wird jeweils eine Zufallszahlenfolge (s_n) und (t_n) auf folgende Art und Weise erzeugt:

- $s_n = s_{n-3} \cdot s_{n-2} \cdot s_{n-1} \pmod{179}$
- $t_n = 53 \cdot t_{n-1} + 1 \pmod{169}$.

Beide Generatoren erzeugen jeweils $97 \cdot 32 = 3104$ Zufallszahlen. Die Generierung der ersten 97 Zufallszahlen des eigentlichen Streams erfolgt dann aufgrund der Berechnung von Zufallsbits durch

- $b_i = 0$ falls $s_i \cdot t_i \pmod{64} < 32$
- $b_i = 1$ sonst

und anschließendes Zusammensetzen der Bits zu den Zufallszahlen

$$y_1 = b_1 b_2 ... b_{32}$$

$$y_2 = b_{33} b_{34} ... b_{64}$$

...

$$y_{97} = b_{3073} b_{3074} ... b_{3104}$$

Als Startwert für e_0 dient im vorliegenden Fall die Zahl 15362436.

Folgende Ausschnitte zeigen nochmals Arbeitsweise und Parametersatz:

```
/* constants for the fibonacci generator FIB(97,33) */
#define FIB_MODLUS    4294967296.0
#define FIB_MULT1      362436069.0
#define FIB_HELP1       15362436.0

/* fibonacci generator Fib(97,33) */
value1=dmod(abs(stream_97->rn_1 - stream_33->rn_1), FIB_MODLUS);
value2=dmod(abs(value2 - FIB_MULT1), FIB_MODLUS);
value=dmod(value1 + value2,FIB_MODLUS);
/* + instead of - because value2 is abs(value2) */
```

stream_97 und stream_33 sind dabei Pointer auf die benötigten Elemente x_{n-97} und x_{n-33} im aktuellen Stream.

Inverser und explizit inverser Kongruenzgenerator

Sowohl inverser, als auch explizit inverser Generator sind 1:1 anhand ihrer Erzeugungsvorschrift zu implementieren.

```
/* constants for the inversive congruential generator 'short icg'*/
```

```
#define ICG_MODLUS      2147483647
#define ICG_MULT                1
#define ICG_ADD                 1

/* compute next value from old 'value'*/

value = inverse_gordon(value, ICG_MODLUS);
value = dmod(ICG_MULT * value, ICG_MODLUS) + ICG_ADD;
```

Die Berechnung des EICG läuft dann völlig analog. Einzig die Berechnung des mulitplikativ inversen Elements, die beiden Verfahren eigen ist, verdient einige Beachtung.
Wie bereits oben erwähnt eignet sich hierfür u.a. der Algorithmus von Euklid. In der vorliegenden Implementierung wurde ein leicht abgewandeltes Verfahren, der Algorithmus von Gordon, eingesetzt. Die ursprüngliche Implementierung stammt von Ottmar Lendl von der Universität Salzburg und wurde freundlicherweise für Forschungszwecke zur Verfügung gestellt. Der folgende Ausschnitt stammt aus diesem Original-Source-Code und verdeutlicht die Arbeitsweise:

```
/*
 * Copyright (c) 1996 Otmar Lendl (lendl@cosy.sbg.ac.at)
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted without a fee provided that the following
 * conditions are met:
 *
 * 1. This software is only used for private, research, or academic
 *    purposes.
 *
 * 2. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 3. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * 4. Any changes made to this package must be submitted to the author.
 *    The legal status of the submitted changes must allow their inclusion
 *    into this package under this license.
 *
 * 5. Publications in the field of pseudorandom number generation, which
 *    made use of this package must include a reference to this package.
 *
 * Modular Inversion. Based on Gordon's Algorithm (improvement of euclids)
 *
 *
 * Input:
 *      a,p    Two prng_num, gcd(a,p) should be 1 !
 *
 * Output:
 *      prng_num which satisfies:
 *             a' * a = 1    (for a != 0)
 *             a'     = 0    (for a == 0)
 *
 */

prng_num inverse_gordon(prng_num a,prng_num p)
{
s_prng_num INV,U,V,HCF,temp2;
prng_num temp;        /* must be UNSIGNED ! I need the bit ! */
 int EnterLoop,shifts;


if (a <= 1)           /* trivial cases */
 return(a);

HCF = p; INV = 0; V= 1; U = a;

do
 {
```

```
  shifts = -1; EnterLoop = FALSE;
  if (HCF<U)
        temp = 0;
   else
        {
        EnterLoop = TRUE; temp = U;
        while ( temp <= HCF)
              {
              shifts++;
              temp = temp << 1;
              }
        temp = temp >> 1;
        };

  temp2 = HCF - temp; HCF = U; U = temp2;
  temp2 = INV; INV = V;

  if (EnterLoop)
        {
        V = V << shifts;
        temp2 -= V;
        }
  V = temp2;
  }
while (! (0 == U) || (U == HCF));

if (INV < 0) INV += p;


if (HCF != 1)
  {
  fprintf(stderr,
        "inverse_gordon: HCF is %ld\n",HCF);
  }

return(INV);
}
```

Literatur

(Ch91) F. Chance. DELPHI: A C-based manufacturing simulator. Cornell University. Ithaca. NY. USA. Version 8. Level 29. 1991.

(Ch93) W.-S. Chou, On inversive maximal period polynomial over finite fields. Preprint. Austrian Academy of Sciences. Institute of Information Processing. Vienna. 1993.

(Ea95) J.W. Eaton. OCTAVE: A high-level interactive language for numerical computations. Free Software Foundation,Inc. Cambridge. MA. USA.Version 1.1.1. January 1995.

(Ei91) J. Eichenauer-Hermann. Inversive congruential pseudorandom numbers avoid the planes. Mathematical Computation. Volume 56. pp 315. 1991.

(Ei93) J. Eichenauer-Herrman. Statistical Independence of a New Class of Inversive Congruential Pseudorandom Numbers. Mathematics of Computation. Volume 60. Number 201. pp 375. January 1993.

(Ei94) J. Eichenauer-Herrmann. Improved lower bounds for the discrepancy of inversive congruential pseudorandom numbers. Mathimatical Computation. Volume 62. pp 783. 1994.

(EiEm94) J. Eichenauer-Herrman, F. Emmerich. A review of compound methods for pseudorandom number generation. Proceedings of the 1st Salzburg Minisymposium on Pseudorandom Number Generation and Quasi-Monte-Carlo Methods. Salzburg. Nov 1994.

(EiNi94) J. Eichenauer-Herrmann, H. Niederreiter. Bounds for the exponential sums and their applications to pseudorandom numbers. Acta Arith.. Volume 67. pp 269. 1994.

(EGL88) J. Eichenauer, H. Grothe and J.Lehn. Marsaglia's lattice test and non-linear congruential pseudo random number generators. Metrika. Volume 35. pp 241. 1988.

(En94) K. Entacher. Randum numbers and parallization: selected generators in the run-test. Preprint. Institut für Mathematik. Universität Salzburg. Austria. 1994.

(En..a) K. Entacher. Selected random number generators in the run test. Preprint. Department of Mathematics. University of Salzburg. Austria.

(En..b) K. Entacher. Selected random number generatores in the run test II: subsequence behavior. Article in preparation. Department of Mathematics. University of Salzburg. Austria.

(EnLe95) K. Entacher, H. Leeb. Inversive pseudorandom number generators: Empirical Results. Parallel Numerics 95. Sorento. Italy. September 1995.

(FlNi92) M. Flahive, H. Niederreiter. On inversive congruential generators for pseudorandom numbers. In G.L. Mullen, P.J.-S. Shiue. Finite Fields, Coding Theory, and Advances in Communications and Computing. pp 75. Dekker. New York. 1992.

(Go96) C. Görg. Lagged-Fibonacci-Generator. In Workshop Simulation. Universität Würzburg. Deutschland. Januar 1996.

(He91) H. Heuser. Lehrbuch der Analysis -- Teil I. 9.Auflage. Teubner. Stuttgart. 1991.

(He94) P. Hellekalek. Study of algorithms for primitive polynomials. Research Institute for Software Technology. Universität Salzburg. April 1994.

(He95) P. Hellekalek. Inversive Pseudorandom Number Generators: Concepts, Results and Links. Proceedings of the 1995 Winter Simulation Conference 1995.

(HMW94) P. Hellekalek, M. Mayer, A. Weingartner. Implementation of algorithms for IMP-polynomials. Research Institute for Software Technology. Universität Salzburg. Oktorber 1994.

(LaKe91) A. M. Law, W. D. Kelton. Simulation Modelling & Analysis. Second Edition. McGraw-Hill. New York. 1991.

(Le95) H. Leeb. Random Numbers For Computer Simulation. Diplomarbeit zur Erlangung des Magistergrades. Universität Salzburg. Januar 1995.

(LiNi83) R.Lidl, H. Niederreiter. Finite Fields. Addison Wesley. Reading. MA. 1983.

(Li95) F. Liebl. Simulation. 2. Auflage. Oldenbourg. München. 1995.

(Ma68) G. Marsaglia. Random numbers fall mainly in the planes. Proc. Nat. Acad. Sci. Volume 61. pp 25. 1968.

(MaPa90) A. De Matteis, S. Pagnutti. Long-range correlations in linear and non-linear random number generators. Parallel Computing. Volume 14. pp 207. 1990.

(Ni92) H. Niederreiter. Random Number Generation and Quasi-Monte-Carlo-Methods. SIAM. Philadelphia. USA. 1992.

(Ni94) H. Niederreiter. On a new class of pseudorandom numbers for simulation methods. Journal of Computer Applied Mathematics. Volume 56. pp 159. 1994.

(Ni95) H. Niederreiter. Some Linear and Nonlinear Methods for Pseudorandom Number Generation. Proceedings of the 1995 Winter Simulation Conference. pp 250. 1995.

(Ni..) H. Niederreiter. New developments in uniform pseudorandom number and vector generation. In H. Niederreiter,R.J.-S. Shiue. Monte Carlo and quasi-Monte Carlo methods in scientific computing. Springer. Berlin. to appear.

(Pa91) B. Page. Diskrete Simulation. Springer. Berlin. 1991.

(PaMi88) S.K. Park, K.W. Miller. Randum number generators: good ones are hard to find. Comm. ACM. Volume 31. pp 1192. 1988.

(Ri87) B.D. Ripley. Stochastic Simulation. John Wiley. New York. 1987.

(Ri92) F. Schreiber. Measurement of the correlation coefficient of the 2-node Markov chain. AEÜ. Volume 40. pp 402, 1986.

(Ro97) O. Rose. Traffic Modeling of Variable Bit Rate MPEG Video and ist Impact on ATM Networks. Dissertation zur Erlangung des Doktorgrades. Universität Würzburg. 1997.

(Sc86) F. Schreiber. Measurement of the stationary state probabilities of the 2-node Markov chain. AEÜ. Volume 41. pp 117. 1987.

(We95) S. Wegenkittl. Empirical Testing of Pseudorandom Number Generators. Diplomarbeit zur Erlangung des Magistergrades. Universität Salzburg. Oktober 1995.

(We94) A. Weingartner. Nonlinear congruential pseudorandom number generators. Diplomarbeit zur Erlangung des Magistergrades. Universität Salzburg. Juni 1994.