Inhaltsverzeichnis

Zufallszahlen nach dem MLKG Algorithmus

```
function random : real;
const
       a = 16807;
       m = 2147483647;          { maxLongInt }
       q = 127773;              { ! q :=  m  DIV  a  }
       r = 2836;                { ! r :=  m  MOD  a  }
var
       k : longint;
begin
       k :=  s DIV q;
       s :=  a * ( s - k * q )    -  k * r ;
       if  s < 0   then   s := s + m ;
       random :=  s / m;
end;
```

Der multiplikative lineare Kongruenzgenerator (MLKG)

... ist der zur Zeit am meisten entwickelte, untersuchte und genutzte Algorithmus zur Erzeugung von Pseudo-Zufallszahlen! Das Verfahren ist sehr einfach und gerade deswegen genial:

Mit den vorgegebenen natürlichen Zahlen m (der Modul), a (der Multiplikator) und s:=s[0] (der Startwert; engl. 'seed', s<m) bildet man die Folge

```
s[k+1] := ( a · s[k] ) modulo m . . . . k = 0,1,2,...
```

und erhält durch Normieren daraus eine Folge x[k] := s[k] / m , die bei geeigneter Wahl von m und a gut als Folge von Pseudo-Zufallszahlen interpretiert werden kann.

Nach welchen Gesichtspunkten sollte man m und a nun wählen?

- Dazu muß zunächst festgestellt werden, daß jeder MLKG nur endlich viele verschiedene Werte x[k] aus dem Intervall (0,1) liefern kann. Sobald der Generator erstmals - angenommen nach n+p Schritten - einen Wert liefert, der in der Folge {s[k]} schon einmal vorkam (z.Bsp. s[n+p] = s[n]), dann werden sich offensichtlich alle weiteren Folgeelemente genau in der damaligen Reihenfolge wiederholen (d.h., es gilt dann x[n+p+k] = x[n+k] für alle k). Die Zahl p nennt man die Periode des Generators.
- Eine naheliegende Forderung ist, daß der MLKG eine möglichst große Periode haben sollte. Die maximal mögliche Periode eines MLKG ist sein Modul m - es empfiehlt sich daher, als Modul eine im Rahmen des unterstützten Zahlenbereiches möglichst große Zahl zu wählen, z.Bsp.

m = MaxLongInt = 2^31 - 1 = 2 147 483 647.

- Ist der Modul m eine Primzahl (maxLongInt ist eine!), dann kann man stets einen passenden Multiplikator a derart finden, daß der MLKG die volle Periode p = m hat! Damit ist dann auch der

Seed s[0] beliebig wählbar.

Schlaue Mathematiker haben herausgefunden, wie man einen solchen Multiplikator finden kann: a sollte eine sogenannte primitive Wurzel von m sein (Das bedeutet, daß a^(m-1) durch m teilbar ist und a^k für alle k<m-1 nicht durch m teilbar ist).

Für den Fall m = maxLongInt gibt es einen einfacheren Weg: a ist genau dann primitive Wurzel von maxLongInt, wenn eine Zahl b gefunden werden kann, die *selbst eine Primzahl ist, *kein Primfaktor von maxLongint - 1 = 2147483646 ist und *a = 7^b modulo m gilt.

b=5 erfüllt alle drei Bedingungen, folglich ist

a = 7^5 = 16 807

eine primitive Wurzel von maxLongInt .

Alle MLKG's haben den Nachteil, daß k-Tupel aufeinanderfolgender Werte - als Punkte im k-dimensionalen EUKLIDischen Raum aufgefaßt - auf parallelen Hyperebenen liegen. Für k=2 (also in der Ebene) bedeutet das, daß alle Punkte auf parallelen Geraden liegen; die Zwischenräume werden also nie getroffen! Man ist deshalb bestrebt, wenigstens den Abstand der Hyperebenen so klein wie möglich zu halten, d.h. einen Multiplikator a derart zu finden, daß für kleine k der minimal mögliche Ebenenabstand annähernd erreicht wird. Es gibt also selbst unter den primitiven Wurzeln noch 'gute' und 'schlechte'. Sollte die Hyperebenenstruktur bei speziellen Anwendungsfällen problematisch erscheinen, benutzt man eine Linearkombination mehrerer MLKG's!

Neben den genannten theoretischen Eigenschaften sollte den Nutzer vor allem interessieren, ob die vom MLKG gelieferte Zahlenfolge {x[k]} tatsächlich die gleichen statistischen Eigenschaften wie eine Folge 'echter' Zufallszahlen hat: Es ist zu untersuchen, ob die x[k] als Realisierungen von unabhängigen, im Intervall [0,1] gleichverteilten Zufallsvariablen aufgefaßt werden können.

Während die Uniformität (Intervalle gleicher Länge werden mit der gleichen Wkt. getroffen) einfach mittels Häufigkeitsanalyse überprüft werden kann, ist der vollständige Nachweis der Unabhängigkeit recht schwierig. Praktisch begnügt man sich meist mit dem Nachweis, daß zwischen k benachbarten Folgeelementen keinerlei Korrelation besteht. Zur Aufdeckung derartiger Korrelationen ist der Poker-Test recht gut geeignet. Weitere leicht zu implementierende und dennoch recht aussagekräftige Verfahren sind der Run-Test und der Gap-Test.