Anhang B

Erzeugung von Zufallszahlen

Die Erzeugung von Zufallszahlen ist bei Monte–Carlo–Simulationen von zentraler Bedeutung. Deshalb ist es nicht verwunderlich, dass regelmässig Veröffentlichungen entstehen, die sich ausschliesslich mit diesem Problem beschäftigen (als Beispiel erwähnen wir [73]). Aus diesem Grund ist es sinnvoll, sich auch hier ein paar Gedanken zur Erzeugung der Zufallszahlen zu machen.

Da herkömmliche Rechenmaschinen von ihrem Konzept her deterministisch sind, sind sie unfähig, Folgen von echten Zufallszahlen zu erzeugen. Das Ziel aller Algorithmen zur Generierung von sogenannten Pseudozufallszahlen ist es, eine Folge zu erzeugen, die dem Betrachter den Eindruck der Zufälligkeit vermittelt. Wir sprechen von Algorithmen zur Erzeugung deterministischer Pseudozufallszahlen. In einer solchen Folge können unvermeidliche Korrelationen zu fehlerhaften Resultaten führen [73].

Generatoren von Pseudozufallszahlen gibt es viele. Zur Erzeugung einer Zufallsfolge der Zahlen 0 und 1 werden in der Regel Algorithmen verwendet, die Bitinformationen von Registern miteinander verknüpfen [73] und dadurch den Vorteil einer hohen Geschwindigkeit haben. Zur Erzeugung von Zufallsfolgen ganzer Zahlen in einem bestimmten, festlegbaren Intervall, oder von Gleitkommazahlen im halboffenen Intervall $[0; 1)$ werden typischerweise Generatoren verwendet, die durch die lineare kongruente Formel

$$x_{n+1} = (ax_n + c)[\text{mod } m] \ , \qquad n \geq 0$$

iterativ berechnet werden. Dabei werden die Parameter a, c, m sowie der Wert x_0 vor einer ersten Ausführung der Iteration festgelegt.

Über den von uns verwendeten Zufallszahlengenerator drand48() können wir auszugsweise in den elektronischen Manuel–Seiten (man–pages) lesen:

```
SYNOPSIS

  #include <stdlib.h>
  double drand48(void);
  void srand48 (
          long seed_val);

PARAMETERS

  seed_val  Specifies the initialization value to begin
```

```
            randomization. Changing this value changes the
            randomization pattern.

DESCRIPTION

The drand48() and erand48() functions return nonnegative,
double-precision, floating-point values uniformly distributed
over the range of y values such that 0 <= y < 1.0.

The srand48(), seed48(), and lcong48() functions initialize the
random-number generator. Programs should invoke one of them
before calling the drand48(), lrand48(), or the mrand48()
functions. (Although it is not recommended practice, constant
default initializer values are supplied automatically if the
drand48(), lrand48(), or mrand48() functions are called without
first calling an initialization function.) The erand48(),
nrand48(), and jrand48() functions do not require that an
initialization function be called first.

All the functions work by generating a sequence of 48-bit
integer values, X[i], according to the linear congruential
formula:   X[n + 1] = (aX[n] + c)[mod m] n>= 0

The parameter m equals 2^48 hence 48-bit integer arithmetic is
performed. Unless lcong48() has been invoked, the multiplier
value a and the addend value c are given by

a = 5DEECE66D (hex) = 273673163155 (octal)
c = B (hex) = 13 (octal)

The values returned by the drand48(), erand48(), lrand48(),
nrand48(), mrand48(), and jrand48() functions are computed by
first generating the next 48-bit X[i] in the sequence. Then the
appropriate bits, according to the type of data item to be
returned, are copied from the high-order (most-significant)
bits of X[i] and transformed into the returned value.

The drand48(), lrand48(), and mrand48() functions store the
last 48-bit X[i] generated into an internal buffer, which is
why they must be initialized prior to being invoked.

The initializer function seed48() sets the value of X[i] to
the 48-bit value specified in the array pointed to by the seed_16v
parameter. In addition, seed48() returns a pointer to a 48-bit
internal buffer that contains the previous value of X[i] which
is used only by seed48(). The returned pointer allows you to
restart the pseudorandom sequence at a given point. Use the
```

```
pointer to copy the previous X[i] value into a temporary array.
To resume where the original sequence left off, you can call
seed48() with a pointer to this array.
```

Bei der Diskussion von Generatoren für Pseudozufallszahlen werden gleichzeitig auch immer Testmethoden für solche Generatoren besprochen (Vgl. beispielsweise [73]). Dabei steht die präzise Messung von Korrelationslängen von Sequenzen der Pseudozufallszahlen im Vordergrund. Keine Testmethode kann jedoch beweisen, dass ein gegebener Generator verlässlich ist in allen Anwendungen.

Als Testmethode für Zufallszahlengeneratoren finden wir in [74] einen einfach zu realisierenden Vorschlag. Die Gitterplätze eines Kubus der Grösse L^3 werden aufgefüllt, indem $t \cdot L^3$ Mal zufällig ein Gitterplatz ausgewählt und vom Anfangswert 0 auf den Wert 1 verändert wird, wobei der Faktor t von der Grössenordnung 10 sein soll. Theoretisch sollte bei einem guten Generator der Anteil der auf 0 verbleibenden Gitterplätze mit $\exp(-t)$ abnehmen. Der Test mit unserem Generator drand48() zeigt das verlangte Verhalten, was den Schluss zulässt, dass er für unsere Zwecke genügt.