Computergestütztes wissenschaftliches Rechnen SoSe 2004

Alexander K. Hartmann, Universität Göttingen

28. April 2004

2.4 Numerik

2.4.1 Zahlendarstellung

- Analogrechner (Rechenschieber, Op-Amp): Zahlen entsprechen physikalische Größen, Genauigkeit = Messgenauigkeit/Rauschen
- Digitalrechner (keine physikalische Rechenungenauigkeit)
  - Ganze Zahlen (`int`, `long`): Zahlendarstellung exakt, außer wenn außerhalb des Bereichs (±`INT_MAX`, in `limits.h`). Alle Rechenoperationen exakt, nur der Rest bei der Division entfällt.
  - Festkommazahlen, gebräuchlich bei kaufmännischen Anwendungen, nicht in Standard C
  - Fließkommazahlen (`float`, `double`): Format

$$s \times 0.M \times B^e \tag{1}$$

    s= Vorzeichen, M=Mantisse, B=Basis (im Rechner üblich $B = 2$, hier zur Darstellung $B = 10$), e=Exponent, alles ganze Zahlen. Genauigkeit bestimmt durch n_M und n_E, die Zahl der Stellen von Mantisse und Exponent.

    Normalisierte Darstellung: 1. Stelle von M ist $\neq 0$ (außer bei 0). Bsp. 0.0034 wird als $+0.34 \times 10^{-2}$ dargestellt.

2.4.2 Numerische Fehler [1]

Umwandlung Realzahl $x \rightarrow$ Fließkommazahl $\tilde{x} = \mathrm{rd}(x)$ z.B. durch Rundung.

Maschinengenauigkeit = kleinste Zahl ϵ_f, so dass die normalisierte Darstellung von $\mathrm{rd}(1+\epsilon_f) \neq \mathrm{rd}(1)$. Also bei Rundung, $B = 10$, $\epsilon_f = 5 \times 10^{-n_M}$. Im Folgenden $n_M = 8$.

$\Rightarrow \mathrm{rd}(x) = x(1 + \epsilon)$ mit $|\epsilon| \leq \epsilon_f$ mit $\epsilon = \epsilon(x)$

Ergebis einer arithmetischen Operation ist nicht unbedingt eine Fließkommazahl, auch wenn beide Operanden Fließkommazahl.

Bsp: $x = 0.12345678 \times 10^0$, $y = 0.5000000 \times 10^{-8}$. Es ist $\mathrm{rd}(x+y) = 0.12345679 \times 10^0 \neq 0.123456785 \times 10^0 = x + y$.

Daher: Operationen im Rechner = *Fließkommaoperationen*, bezeichnet z.B. mit $+^*$. Der Fehler dabei: $x +^* y = (x + y)(1 + \epsilon_1)$ mit $|\epsilon_1| \leq \epsilon_f$, $\epsilon_1 = \epsilon_1(x, y)$.

Für Fließkommaoperationen gelten die üblichen Gesetze für aritmetische Operation nicht!

Beispiel: Assotiativgesetz, Addition
Für

$$\begin{aligned} a &:= 0.23371258 \times 10^{-4} \\ b &:= 0.33678429 \times 10^{2} \\ c &:= -0.33677811 \times 10^{2} \end{aligned}$$

ist

$$\begin{aligned} a +^* (b +^* c) &= 0.23371258 \times 10^{-4} +^* 0.6180000 \times 10^{-3} \\ &= 0.64137126 \times 10^{-3} \\ (a +^* b) +^* c &= 0.33678452 \times 10^{2} -^* 0.33677811 \times 10^{2} \\ &= 0.64100000 \times 10^{-3} \end{aligned}$$

Exakt ist $a + b + c = 0.641371258 \times 10^{-3}$. □

Erklärung: *Fehlerfortpflanzung* des Rundungsfehlers. Analyse der $+^*$ Operation:

$$\begin{aligned} (a +^* b) +^* c &= (a + b)(1 + \epsilon_1) +^* c \\ &= [(a + b)(1 + \epsilon_1) + c][1 + \epsilon_2] \\ &= (a + b + c)(1 + \frac{a + b}{a + b + c}\epsilon_1(1 + \epsilon_2) + \epsilon_2) \\ &\approx (a + b + c)(1 + \frac{a + b}{a + b + c}\epsilon_1 + \epsilon_2) \\ &=: (a + b + c)(1 + \epsilon_{12}) \end{aligned}$$

Günstig: Die zuerst addierten Zahlen $a + b$ klein gegen die Gesamtsumme, wie im ersten Teil des Beispiels.
Ein Ziel numerischer Mathematik: Entwicklung von Algorithmen, so dass Gesamtfehler möglichst klein (d.h. Algorithmus möglichst *numerisch stabil*). Analyse mittels allgemeiner Fehlerfortpflanzung.

3 (Pseudo-) Zufallszahlen

Anwendung von Zufallszahlen bei Computersimulationen:

- Systeme mit zufälligen Wechselwirkungen ("Spingläser")
- Simulatione bei endlichen Temperaturen mit Monte-Carlo Verfahren

- Randomisierte Algorithmen (aus deterministischen Algorithmen entstanden)

Zufallszahlen im Computer möglich (z.B. Schwankungen der Spannungen an einem Transistor durch thermisches Rauschen). Vorteil: Zufällig. Nachteil: Statistische Eigenschaften unbekannt und nicht kontrollierbar.
Daher: Pseudozufallszahlen = nicht zufällig, aber *möglichst* gleiche statische Eigenschaften (Verteilung, Korrelationen).

3.1 Linear kongruenter Generator

Erzeugt Folge $I_1, I_2, \ldots$ von Werten zwischen 0 und $m-1$, startend von gegebenen I_0.

$$I_{n+1} = (aI_n + c) \bmod m \tag{2}$$

Zufallszahlen r gleichmäßig im Intervall $[0, 1)$: $x_n = I_n/m$. Beliebige Verteilungen: nächste Vorlesung

Einfache Iterationsgleichung, analog zur Populationsdynamik. Hier ist möglichst "chaotisches" Verhalten erwünscht. Ziel: Wahl der Paramater a, c, m (und I_0), so dass der Generator "gut" ist → Kriterien benötigt. Achtung: Öfters waren Ergebnisse von Simulationen falsch, w.g. schlechter Zufallszahlengeneratoren[2].

Programm `linear_congruential.c` erezugt Zufallszahlen und erstellt ein Histogramm der Häufigkeiten:

```
/*** Linear congruential generator                                  ***/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NUM_BINS 100
int main(int argc, char *argv[])
{
  int a, c, m, I;              /* parameter of random-number generator */
  double number;                                  /* generated number */
  int num_runs;                  /* number of generated random numbers */
  int histo[NUM_BINS];            /* histogram to measure distribution */
  double start_histo, end_histo;                 /* range of histogram */
  double delta;                                        /* width of bin */
  int bin;
  int t;                                               /* loop counter */

  m = 32768; c = 1; I = 1000;

  sscanf(argv[1], "%d", &num_runs);                 /* read parameters */
  sscanf(argv[2], "%d", &a);
  for(t=0; t<NUM_BINS; t++)                    /* initialise histogram */
      histo[t] = 0;
  start_histo = 0.0; end_histo = 1;
  delta = (end_histo - start_histo)/NUM_BINS;
```

```
  for(t=0; t<num_runs; t++)                              /* main loop */
  {
    I = (a*I+c)%m;                     /* linear congruential generator */
    number = (double) I/m;                        /* map to interval [0,1) */
    bin = (int) floor((number-start_histo)/delta);
    if( (bin >= 0)&&(bin < NUM_BINS))                /* inside range ? */
       histo[bin]++;                                    /* count event */
  }

  for(t=0; t<NUM_BINS; t++)               /* print normalized histogram */
      printf("%f %f\n", start_histo + (t+0.5)*delta,
             histo[t]/(delta*num_runs));
  return(0);
}
```

Beispiel: $a = 12351, c = 1, m = 2^{15}$ und $I_0 = 1000$ (und durch m teilen). Verteilung: ist "gleichmäßig" in $[0, 1)$ verteilt (Fig. 1), aber sehr regelmäßig.

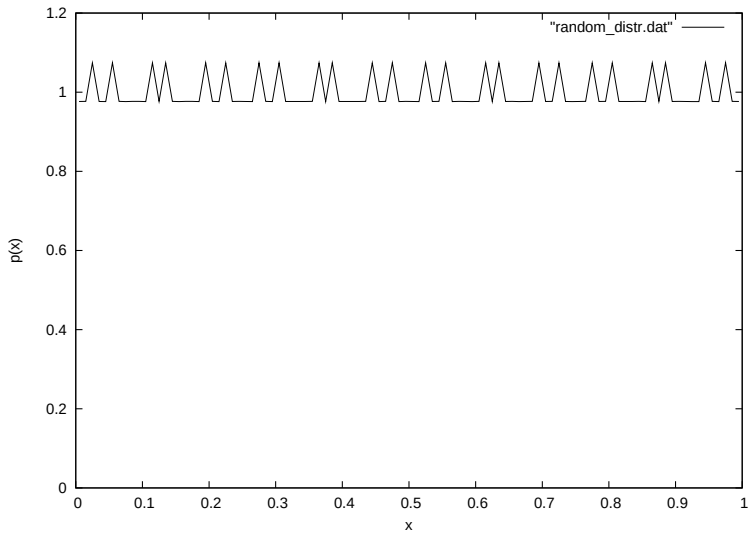

Abbildung 1: Verteilung von Zufallszahlen im Intervall $[0, 1)$, erzeugt mit einem linear kongruenten Generator mit Parametern $a = 12351, c = 1, m = 2^{15}$.

Daher: Korrelationen. Untersuche: k-Tupel aus k aufeinanderfolgenden Zufallszahlen $(x_i, x_{i+1}, \ldots, x_{i+k-1})$. Kleine Korrelationen: k-dim Raum uniform gefüllt. LKGs: Punkte liegen auf $k-1$-dim Ebenen, deren Zahl ist *maximal* $O(m^{1/k})$ [3]. Obige Zahlenkombination $\rightarrow$ wenige Ebenen.

Ergänzungen/Änderungen zur Messung Zweierkorrelation:

```
double number_old;

number_old = (double) I/m;

for(t=0; t<num_runs; t++)                                    /* main loop */
{
  I = (a*I+c)%m;                        /* linear congruential generator */
  number = (double) I/m;                          /* map to interval [0,1) */
  bin = (int) floor((number-start_histo)/delta);
  printf("%f %f\n", number_old, number);
  number_old = number;
}
```

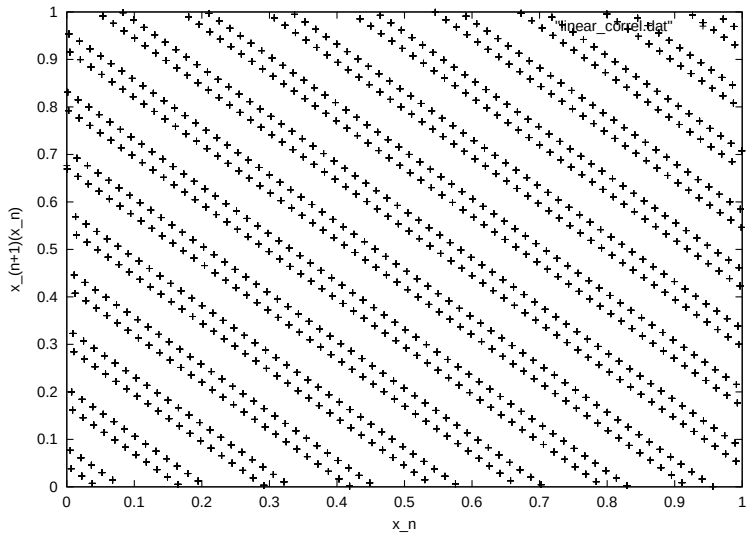

Abbildung 2: Zweipunkt Korrelation $x_{i+1}(x_i)$ zwischen aufeinanderfolgenden Zahlen x_i, x_{i+1}. Linear kongruenter Generator mit Parametern $a = 12351, c = 1, m = 2^{15}$.

Besser: $a = 12349$, Fig. 3.

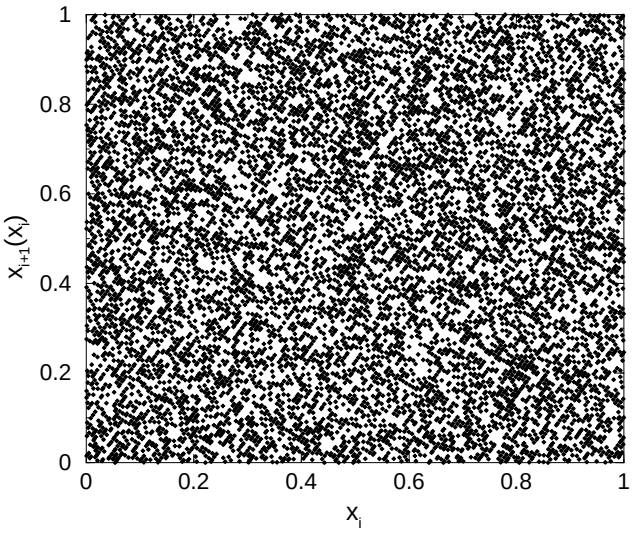

Abbildung 3: Zweipunkt Korrelation $x_{i+1}(x_i)$ zwischen aufeinanderfolgenden Zahlen x_i, x_{i+1}. Linear kongruenter Generator mit Parametern $a = 12349, c = 1, m = 2^{15}$.

Literatur

[1] J. Stoer, Numerische Mathematik, (Springer, Heidelberg 1989)

[2] A.M. Ferrenberg, D.P. Landau and Y.J. Wong, *Phys. Rev. Lett.* **69**, 3382 (1992); I. Vattulainen, T. Ala-Nissila and K. Kankaala, *Phys. Rev. Lett.* **73**, 2513 (1994)

[3] B.J.T. Morgan, *Elements of Simulation*, (Cambridge University Press, Cambridge 1984)