Block 7 (Freitag 23.2.2024)

VIDEO: `video06e_linear_congruential_r`

8.3 Pseudozufallszahlen: Linear kongruenter Generator

[Selbsttest]

Welche Möglichkeiten kennen Sie, Zufallszahlen im Rechner zu erzeugen?

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

$$I_{n+1} = (aI_n + c) \bmod m \tag{41}$$

Zufallszahlen x_n gleichmäßig im Intervall $[0, 1)$: $x_n = I_n/m$. Beliebige Verteilungen: s.u.

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 (Ferrenberg et al., 1992) [1].

Programm `linear_congruential.c` erzeugt Zufallszahlen und erstellt ein Historgramm 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. 11), aber sehr regelmäßig. 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* $O(m^{1/k})$ (B.J.T. Morgan, Elements of Simulation, 1984) [2]. Obige Zahlenkombination → wenige Ebenen.

Ergänzungen/Änderungen zur Messung Zweierkorrelation:

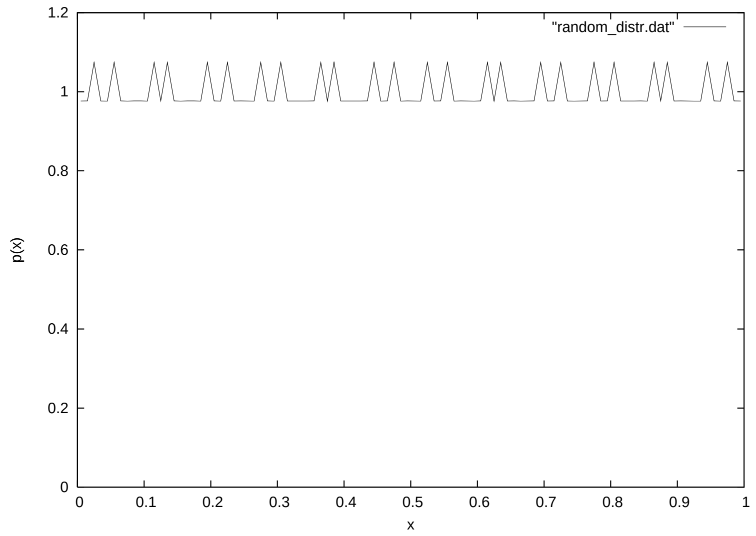

Figure 11: Verteilung von Zufallszahlen im Intervall $[0, 1)$, erzeugt mit einem linear kongruenten Generator mit Parametern $a = 12351, c = 1, m = 2^{15}$.

```
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;
}
```

[Selbsttest]

Erzeugen Sie Zufallszahlen in $[0, 1)$ mit dem zur Verfügung gestellten Programm auf Ihrem Laptop für:

a) $a = 12351, c = 1, m = 2^{15}$ und $I_0 = 1000$

b) $a = 12349$, $c = 1, m = 2^{15}$ und $I_0 = 1000$

Plotten sie die zweier Korrelationen mit gnuplot.

Bild a)

Bild b)

Hinweise: Die *GNU Scientific Library* (GSL) bietet hochwertige Generatoren wie den *Mersenne Twister*. Für kleine Experimente kann man sich in Unix mit `drand48()` behelfen.

VIDEO: `video06f_inversion_r`

8.4 Inversions Methode

Gegeben: `drand48()` (MS: `((double) rand())/(RAND_MAX)`) generiert gleichverteilte Zahlen in $[0,1)$, bezeichnet mit U.

Ziel: Zufallszahlen Z gemäß W.-Dichte $p(z)$ mit Verteilung

$$P(z) \equiv \text{Prob}(Z \leq z) \equiv \int_{-\infty}^{z} dz' p(z') \tag{42}$$

Idee: suche Funktion $g()$ mit $Z = g(U)$. Annahme: g ist streng monoton steigend, d.h. kann invertiert werden $\rightarrow$

$$P(z) = \text{Prob}(Z \leq z) = \text{Prob}(g(U) \leq z) = \text{Prob}(U \leq g^{-1}(z)) \tag{43}$$

Mit
1) $\text{Prob}(U \leq u) = F(u) = u$ wenn U gleichverteilt in $[0,1)$
2) Identifizierung u mit $g^{-1}(z)$
$\Rightarrow P(z) = g^{-1}(z) \Rightarrow z = g(u) = P^{-1}(u)$. (links+rechts invertiert)

Funktioniert, wenn P (evtl. numerisch) berechnet und invertiert werden kann.

Beispiel: Gleichverteilung in $[2,4]$: $p(z) = 0.5$ für $z \in [2,4]$, 0 sonst. $\Rightarrow$ $P(z) = 0.5 \times (z-2)$ für $z \in [2,4]$. Mit u gleichsetzen und nach z auflösen, also: erzeuge gleichverteilte Zahl u und wähle $z = 2 + 2 \times u$.

[Selbsttest]

Wie funktioniert die Generation gemäß der Exponentialverteilung: $p(z) = \lambda \exp(-\lambda z)$, $z \in [0, \infty)$?
Herleitung:

Programm `exponential.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUM_BINS 100

int main(int argc, char *argv[])
{
  int histo[NUM_BINS];                                     /* histogram */
  int bin;
  double start_histo, end_histo;                  /* range of histogram */
  double delta;                                         /* width of bin */
  int t;                                                /* loop counter */
  int num_runs;                    /* number of generated random numbers */
  double lambda;                           /* parameter of distribution */
  double number;                                    /* generated number */

  num_runs = atoi(argv[1]);                          /* read parameters */
  sscanf(argv[2], "%lf", &lambda);
  for(t=0; t<NUM_BINS; t++)                     /* initialise histogram */
      histo[t] = 0;
  start_histo = 0.0; end_histo = 10.0/lambda;
  delta = (end_histo - start_histo)/NUM_BINS;

  for(t=0; t<num_runs; t++)                                /* main loop */
  {
    number = -log(drand48())/lambda;     /* generate exp-distr. number */
```

```
    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);
}
```

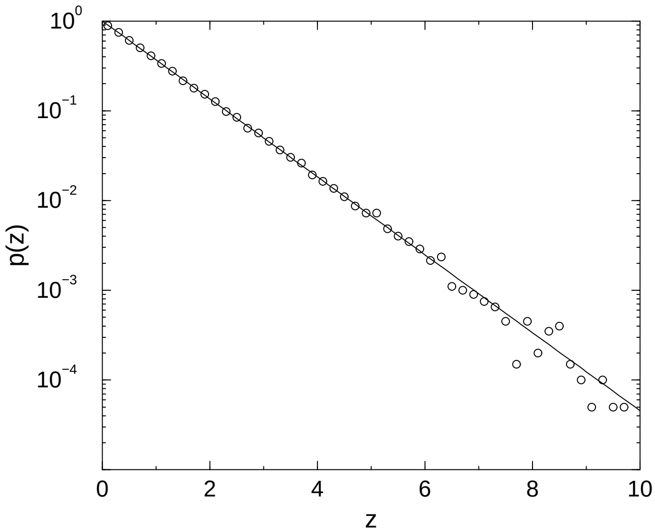

Figure 12: Histogramm von Zufallszahlen, generiert für eine Exponential Verteilung ($\lambda = 1$) verglichen mit W.-Dichte in logarithmischer Auftragung.

VIDEO: `video06g_reject_r`

8.5 Zurückweisungsmethode

Für (analytisch) nicht-integrable W.-Dichten, oder (analytisch) nicht-invertierbare Verteilungen.

Bedingung: W-Dichte $p(x)$ passt in Kasten $[x_0, x_1) \times [0, p_{\max}]$, d.h. $p(x) = 0$ für $x \notin [x_0, x_1]$ und $p(x) \leq p_{\max}$.

Grundidee: Erzeuge zufällige Paare (x, y), gleichverteilt in $[x_0, x_1) \times [0, p_{\max})$. Akzeptiere nur die x mit $y \leq p(x)$, d.h. die Paare unterhalb $p(x)$, siehe Abb. 13.

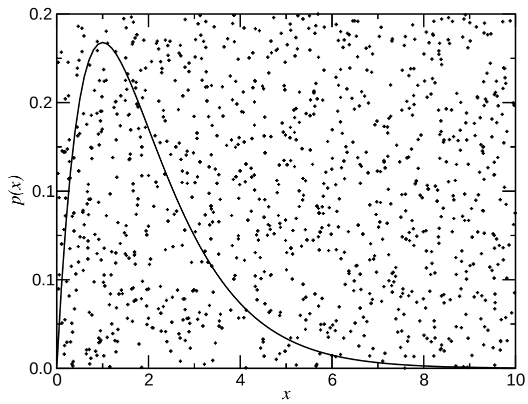

Figure 13: Zurückweisungsmethode: Punkte (x, y) sind gleichmäßig in dem Rechteck verteilt. Die Wahrscheinlichkeit, dass $y \leq p(x)$ ist proportional zu $p(x)$.

Implementierung als Funktion (Programm `reject.c`):

```
/** gerenates random number for 'pdf' in the range **/
/** ['x0', 'x1'). condition: pdf(x)<='p_max' in    **/
//* the range ['x0', 'x1')                         **/
double reject(double p_max, double x0, double x1,
              double (* pdf)(double))
{
  int found;                /* flag if valid number has been found */
  double x,y;                /* random points in [x0,x1]x[0,p_max] */
  found = 0;
  while(!found)                  /* loop until number is generated */
  {
    x = x0 + (x1-x0)*drand48();             /* uniformly on [x0,x1] */
    y = p_max *drand48();                 /* uniformly in [0,p_max] */
    if(y <= pdf(x))                                    /* accept ? */
        found = 1;
  }
  return(x);
}
```

Beispiel:

```
/** artifical pdf **/
double pdf(double x)
{
  if( (x<0)||
      ((x>=0.5)&&(x<1))||
      (x>1.5) )
      return(0);
  else if((x>=0)&&(x<0.5))
      return(1);
  else
      return(4*(x-1));
}
```

ergibt bei 100000 Zufallszahlen das Ergebnis von Bild 14.
Nachteil: Es müssen unter Umständen viele Zufallszahlen weggeworfen werden. Effizienz $1/(2p_{\max}(x_1 - x_0))$. (Faktor 1/2 da man mind. 2 Zahlen (x, y) für eine Zufallszahl braucht).

VIDEO: `video06i_schaetzwerte_r`

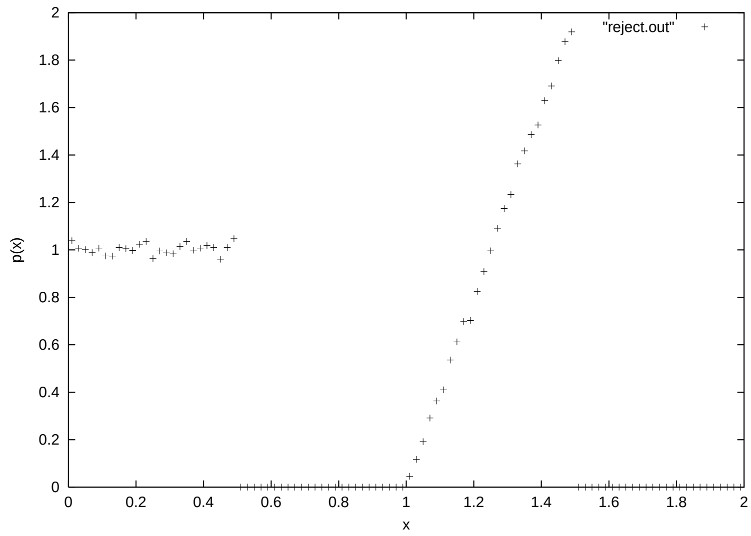

Figure 14: Zurückweisungsmethode: Histogramm für die angegebene W. Dichte.

8.6 Grundlagen Datenanalyse

Gegeben: n Messpunkte ("Stichprobe") $\{x_0, x_1, \ldots, x_{n-1}\}$

Problem: zugrundeliegende Verteilung $F(x)$ meistens unbekannt.

8.6.1 Schätzwerte

Schätzwerte $h = h(x_0, x_1, \ldots, x_{n-1})$ sind selber Zufallsgrößen: $H = h(X_0, X_1, \ldots, X_{n-1})$

- <u>Mittelwert</u> (MW)

$$\overline{x} \equiv \frac{1}{n}\sum_{i=0}^{n-1} x_i \tag{44}$$

- <u>Stichprobenvarianz</u>

$$s^2 \equiv \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \overline{x})^2 \tag{45}$$

- <u>Stichproben Standardabweichung</u>

$$s \equiv \sqrt{s^2} \tag{46}$$

MW: Zur Schätzung des Erwartungswertes $\mu = \mathrm{E}[X]$. MW entspricht ZV $\overline{X} = \frac{1}{n}\sum_{i=0}^{n-1} X_i$. $\Rightarrow$

$$\mu_{\overline{X}} \equiv \mathrm{E}[\overline{X}] = \mathrm{E}\left[\frac{1}{n}\sum_{i=0}^{n-1} X_i\right] = \frac{1}{n}\sum_{i=0}^{n-1}\mathrm{E}[X_i] = \frac{1}{n} n\,\mathrm{E}[X] = \mathrm{E}[X] = \mu \tag{47}$$

$\rightarrow$ der Mittelwert ist <u>erwartungstreu</u>.
Verteilung der $\overline{X}$ hat Varianz:

$$\begin{aligned}
\sigma^2_{\overline{X}} &\equiv \mathrm{Var}[\overline{X}] = \mathrm{Var}\left[\frac{1}{n}\sum_{i=0}^{n-1} X_i\right] \overset{\mathrm{Var}[\alpha X]=\alpha^2\mathrm{Var}[X]}{=} \frac{1}{n^2}\sum_{i=0}^{n-1}\mathrm{Var}[X_i] \\
&= \frac{1}{n^2} n\,\mathrm{Var}[X] = \frac{\sigma^2}{n}
\end{aligned} \tag{48}$$

$\rightarrow$ wird schmaler für wachsendes n
$\rightarrow$ Schätzung wird genauer (wobei σ^2 unbekannt)

$\rightarrow$ gesucht: erwartungstreuer Schätzwert für σ^2 Versuch für $S^2 = \frac{1}{n}\sum_{i=0}^{n-1}(X_i - \overline{X})^2$:

$$\begin{aligned}
\mathrm{E}[S^2] &= \mathrm{E}\left[\frac{1}{n}\sum_{i=0}^{n-1}(X_i - \overline{X})^2\right] = \mathrm{E}\left[\frac{1}{n}\sum_{i=0}^{n-1}(X_i^2 - 2X_i\overline{X} + \overline{X}^2)\right] \\
&\overset{\sum_i X_i = n\overline{X}}{=} \frac{1}{n}\left(\sum_{i=0}^{n-1}\mathrm{E}[X_i^2] - n\,\mathrm{E}[\overline{X}^2]\right) \overset{\mathrm{E}[Y^2]=\sigma_Y^2+\mu_Y^2}{=} \frac{1}{n}\left(n(\sigma^2+\mu^2) - n(\sigma^2_{\overline{X}} + \mu^2_{\overline{X}})\right) \\
&\overset{\sigma^2_{\overline{X}} = \frac{\sigma^2}{n}}{=} \frac{1}{n}\left(n\sigma^2 + n\mu^2 - n\frac{\sigma^2}{n} - n\mu^2\right) = \frac{n-1}{n}\sigma^2
\end{aligned} \tag{49}$$

S^2 ist <u>nicht</u> erwartungstreu, wohl aber $\frac{n}{n-1}S^2$

Forgeschrittene Themen umfassen:
Resampling, Hypothesentests, Clustering, Fits .... (siehe A.K. Hartmann, *Big Practical Guid to Computer Simulations*, (World-Scientific, 2015) [3])