
Vorlesung 6 (Freitag 23.2.2018)

6.1.2 Kontinuierliche Zufallsvariablen

Definition: Für eine ZV X mit kontinuierlicher VF F_X, ist die Wahrscheinlichkeitsdichte (engl. probability density function, pdf) $p_X : \mathbb{R} \to \mathbb{R}$:

$$p_X(x) = \frac{dF_X(x)}{dx} \tag{22}$$

Damit:

$$F_X(x) = \int_{-\infty}^{x} p_X(\tilde{x})\, d\tilde{x} \tag{23}$$

Definition:

- Erwartungswert

$$\mathrm{E}[X] = \int_{-\infty}^{\infty} dx\, x\, p_X(x) \tag{24}$$

- Varianz

$$\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2] = \int_{-\infty}^{\infty} dx\, (x - \mathrm{E}[X])^2 p_X(x) \tag{25}$$

- Median $x_{\text{med}} = \mathrm{Med}[X]$

$$F_X(x_{\text{med}}) = 0.5 \tag{26}$$

Definition: Gleichverteilung mit Parametern $a < b$, beschreibt ZV X mit pdf

$$p_X(x) = \begin{cases} 0 & x < a \\ \frac{1}{b-a} & a \le x < b \\ 0 & x \ge b \end{cases} \tag{27}$$

Schreibweise: $X \sim U(a,b)$.
Mittels $g(X_{01}) = (b-a) * X_{01} + a$ erhält man $g(X_{01}) \sim U(a,b)$ falls $X_{01} \sim U(0,1)$.

[Selbsttest]

Berechnen sie E[X] und Var[X].

Am wichtigesten:
Definition: Die Gaußverteilung oder Normalverteilung mit Parametern μ und $\sigma > 0$, beschreibt die ZV X mit pdf

$$p_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{28}$$

Schreibweise: $X \sim N(\mu, \sigma^2)$.
Eigenschaften: $\mathrm{E}[X] = \mu$, $\mathrm{Var}[X] = \sigma^2$.
Mittels $g(X) = \sigma X_0 + \mu$ erhält man $g(X_0) \sim N(\mu, \sigma^2)$ falls $X_0 \sim N(0, 1)$.

Zentraler Grenzwertsatz:
Für unabhängige ZVen $X^{(1)}, X^{(2)}, \ldots, X^{(n)}$ mit jeweils $E[X^{(i)}] = \mu$ und $\mathrm{Var}[X^{(i)}] = \sigma^2$ gilt:

$$X = \sum_{i=1}^{n} X^{(i)} \tag{29}$$

ist für große n $X \sim N(n\mu, n\sigma^2)$.
Dichten weiterer wichtiger Verteilungen:
Definition:

- Exponentialverteilung (für $x \geq 0$)

$$p_X(x) = \frac{1}{\mu} \exp\left(-x/\mu\right) \tag{30}$$

[Selbsttest]

Berechnen Sie die Verteilungsfunktion für die Exponentialverteilung

---

Definition:

- Potenz-Gesetz Verteilung oder Pareto Verteilung

$$p_X(x) = \begin{cases} 0 & x < 1 \\ \frac{\gamma}{\kappa}(x/\kappa)^{-\gamma-1} & x \geq 1 \end{cases} \tag{31}$$

Für $\gamma > 1$ existiert Erwartungswert $\mathrm{E}[X] = \gamma\kappa/(\gamma-1)$, für $\gamma > 2$ $\mathrm{Var}[X] = \frac{\kappa^2\gamma}{(\gamma-1)^2(\gamma-2)}$

$$F_X(x) = 1 - (x/\kappa)^{-\gamma} \quad (x \geq 1) \tag{32}$$

- Fisher-Tippett Verteilung

$$p_X(x) = \lambda e^{-\lambda x} e^{-e^{-\lambda x}} \tag{33}$$

(auch Gumbel Verteilung für $\lambda = 1$) $\mathrm{E}[X] = \nu/\lambda$, $\nu \equiv 0.57721\ldots$, Maximum bei $x = 0$, Verschiebung durch $x \to (x - \mu)$

[Selbsttest]

Können Sie die Verteilungsfunktion ablesen ?

6.2 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.

$$I_{n+1} = (aI_n + c) \bmod m \tag{34}$$

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[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. 12), aber sehr regelmäßig.

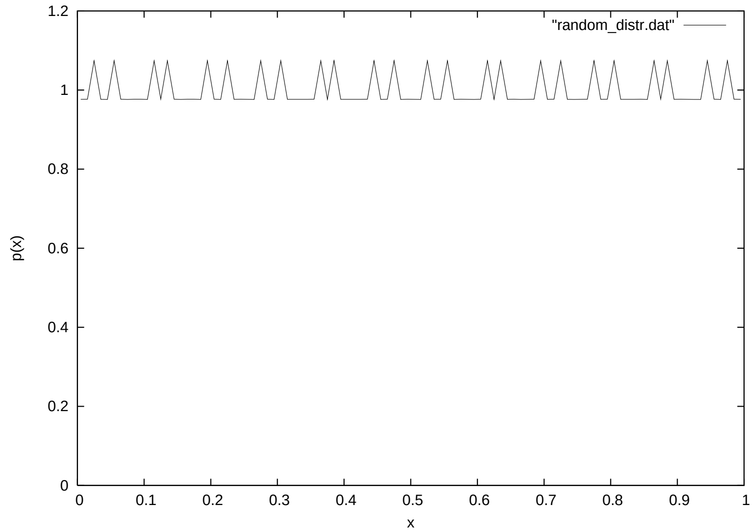

Figure 12: Verteilung von Zufallszahlen im Intervall $[0, 1)$, erzeugt mit einem linear kongruenten Generator mit Parametern $a = 12351, c = 1, m = 2^{15}$.

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* $O(m^{1/k})$ [2]. Obige Zahlenkombination → 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;
}
```

[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.

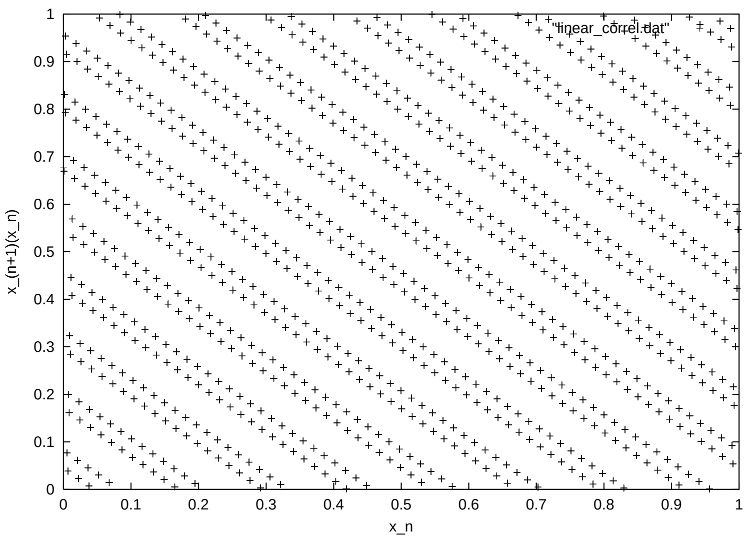

Figure 13: Zweipunkt Korrelation $x_{i+1}(x_i)$ zwischen aufeinanderfolgenden Zahlen x_i, x_{i+1}. Linear kongruenter Generator mit Parametern $a = 12351, c = 1, m = 2^{15}$.

Besser: $a = 12349$, Fig. 14.

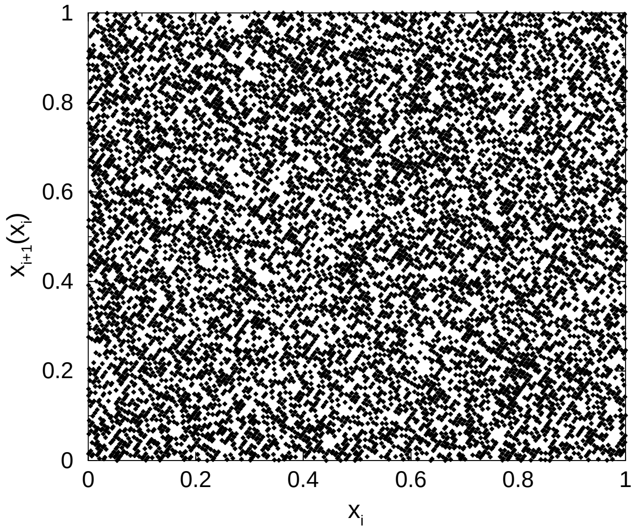

Figure 14: Zweipunkt Korrelation $x_{i+1}(x_i)$ zwischen aufeinanderfolgenden Zahlen x_i, x_{i+1}. Linear kongruenter Generator mit Parametern $a = 12349, c = 1, m = 2^{15}$.

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

6.3 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

$$P(z) \equiv \text{Prob}(Z \leq z) \equiv \int_{-\infty}^{z} dz' p(z') \tag{35}$$

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

$$P(z) = \text{Prob}(Z \leq z) = \text{Prob}(g(U) \leq z) = \text{Prob}(U \leq g^{-1}(z)) \tag{36}$$

Mit

1) $\text{Prob}(U \leq u) = F(u) = u$ wenn U gleichverteilt in $[0, 1)$

2) Identifizierung u mit $g^{-1}(z)$
$\Rightarrow P(z) = g^{-1}(z) \Rightarrow g(u) = P^{-1}(u)$. (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 $z \in [2,4]$. Damit Erzeuge gleichverteilte Zahl u und wähle $z = 2 + 2 \times u$.

[Selbsttest]

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

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);
}
```

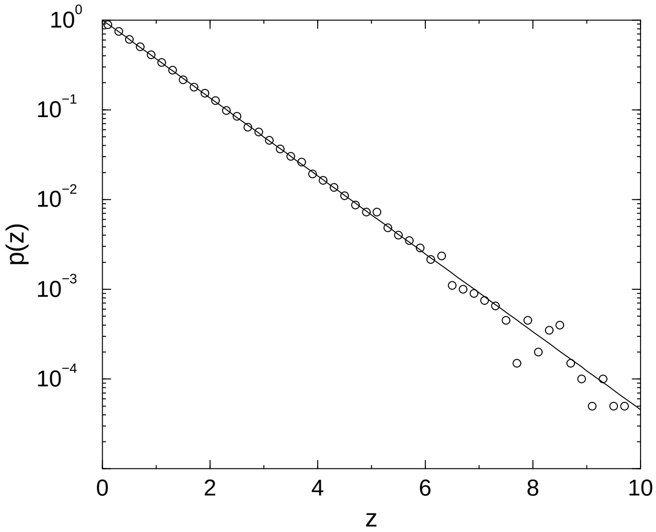

Figure 15: Histogramm von Zufallszahlen, generiert für eine Exponential Verteilung ($\lambda = 1$) verglichen mit W.-Dichte in logarithmischer Auftragung.