Block 7 (Freitag 23.2.2024)

VIDEO: `video06e_linear_congruential_r`

8.3 Pseudo Random Numbers: Linear Congruential Generator (LCG)

[Activator]

Which approaches do you know to generate random numbers in a computer?

LCG: Generate series $I_1, I_2, \ldots$ of values between 0 and $m-1$, starting from given value I_0.

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

$\to$ (Pseudo) random numbers x_n uniformly in interval $[0,1)$: $x_n = I_n/m$. Arbitrary distribution: see below

Wanted: "chaotic" behavior. Aim: chose parameters a, c, m (and I_0), such that generator is "good" $\to$ criteria needed. Attention: Frequently results of simulations turned out to be (slightly) wrong due to bad random number generators (Ferrenberg et al., 1992) [1].

Program `linear_congruential.c` generates random numbers and creates histogram of the frequencies of occurrence:

```
/*** 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);
}
```

Example: $a = 12351, c = 1, m = 2^{15}$ and $I_0 = 1000$ (values divided by m).
Distribution: is "uniform" in $[0, 1)$ (Fig. 11), but very regular.
Thus: correlations. Analysis: k-tuples of k successive random numbers $(x_i, x_{i+1}, \ldots, x_{i+k-1})$. Small correlation: k-dim space uniformly covered. LCGs: tuples are located on $k-1$-dim planes, their number is *at most* $O(m^{1/k})$ (B.J.T. Morgan, Elements of Simulation, 1984) [2]. Above parameter combinations → very few planes.

Change of program to measure 2-tuple correlations:

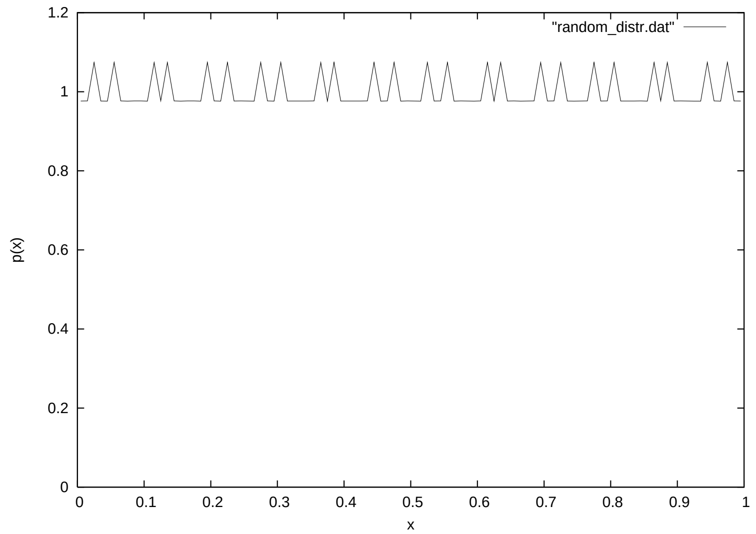

Figure 11: Distribution of random numbers in interval $[0, 1)$, generated by a linear congruential generator with parameters $a = 12351, c = 1, m = 2^{15}$.

```
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;
}
```

[Activator]

Generate random numbers in $[0, 1)$ with the provided program for :
a) $a = 12351, c = 1, m = 2^{15}$ and $I_0 = 1000$
b) $a = 12349$, $c = 1, m = 2^{15}$ and $I_0 = 1000$
Plot the 2-correlations using gnuplot.

Fig. a)

Fig. b)

Remarks:: The *GNU Scientific Library* (GSL) offers high-quality generators like the *Mersenne Twister*. For small experiments one can also use in Unix the `drand48()`.

VIDEO: `video06f_inversion_r`

8.4 Inversion Method

Given: `drand48()` (MS: `((double) rand())/(RAND_MAX)`) generates uniformly numbers in $[0, 1)$, denoted as U.

Target: random numbers Z distributed according to pdf $p(z)$, i.e. with distribution

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

Idea: look for function $g()$ with $Z = g(U)$. Assumption: g is strongly monotonous growing, i.e. it can be inverted→

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

With
1) $\mathrm{Prob}(U \le u) = F(u) = u$ if U uniformly in $[0, 1)$
2) Identification u with $g^{-1}(z)$
$\Rightarrow u = P(z) = g^{-1}(z) \Rightarrow z = g(u) = P^{-1}(u)$. (invert left and right)

Works if P can be obtained and inverted (possibly numerically).

Example: uniform distribution in $[2,4]$: $p(z) = 0.5$ for $z \in [2,4]$, 0 else. $\Rightarrow$ $P(z) = 0.5 \times (z-2)$ for $z \in [2,4]$. Equate to u and resolve with respect to z, thus: generated uniformly distributed number u and choose $z = 2 + 2 \times u$.

[Activator]

How does the generation look like for the exponential distribution: $p(z) = \lambda \exp(-\lambda z)$, $z \in [0, \infty)$?
Calculation:

Program `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);
}
```

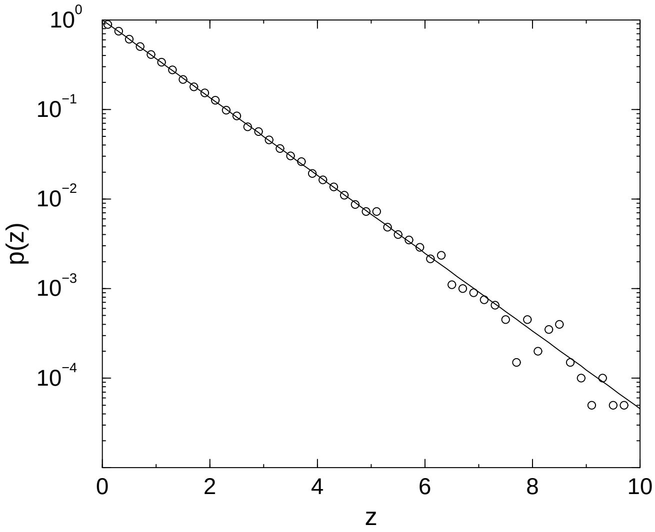

Figure 12: Histogram of random numbers, generated for exponential distribution ($\lambda = 1$) compared to pdf with logarithmic y-axis.

VIDEO: `video06g_reject_r`

8.5 Rejection Method

For (analytically) non-integrable pdfs, or (analytically) non-invertable distributions.

Simple variant: Condition: pdf $p(x)$ fits into box $[x_0, x_1) \times [0, p_{\max}]$, i.e. $p(x) = 0$ for $x \notin [x_0, x_1]$ and $p(x) \leq p_{\max}$.

Basic idea: generate random pairs (x, y), distributed uniformly in $[x_0, x_1) \times [0, p_{\max})$. Accept only those x with $y \leq p(x)$, i.e. the pairs below $p(x)$, see Fig. 13. The x value is the generated number for the pair.

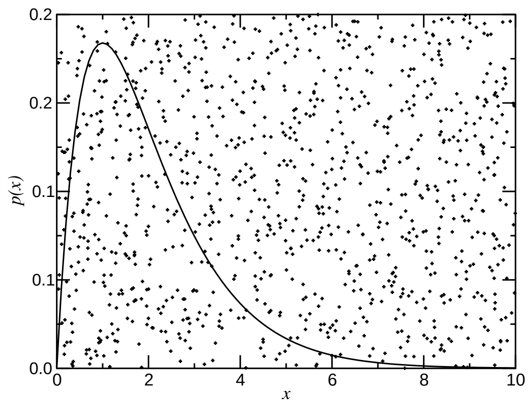

Figure 13: Rejection method: points (x, y) are uniformly distributed in rectangle. The probability that $y \leq p(x)$ is proportional to $p(x)$.

Implementierung as function (program `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));
}
```

results for 100000 random numbers is shown in Fig. 14.
Disadvantage: Possibly many random numbers are thrown away. Efficiency $1/(2p_{\max}(x_1-x_0))$. (Factor 1/2 because at least two numbers (x,y) are needed for one final random number).

VIDEO: `video06i_schaetzwerte_r`

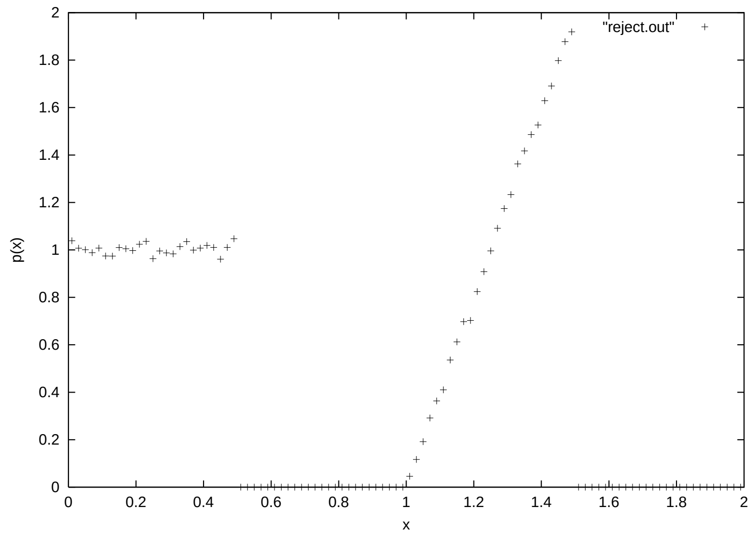

Figure 14: Rejection method: Histogram for the sample pdf.

8.6 Basic Data Analysis

Given: n data points ("sample") $\{x_0, x_1, \ldots, x_{n-1}\}$

Problem: underlying distribution $F(x)$ usually unknown.

8.6.1 Estimators

Estimators $h = h(x_0, x_1, \ldots, x_{n-1})$ are random variables as well: $H = h(X_0, X_1, \ldots, X_{n-1})$

- Mean (MW)

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

- Sample variance

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

- Sample standard deviation

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

MW: To estimate the expectation value $\mu = \mathrm{E}[X]$. MW corresponds to RV $\overline{X} = \frac{1}{n} \sum_{i=0}^{n-1} X_i$. $\Rightarrow$

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

$\rightarrow$ the mean unbiased.
Distribution for $\overline{X}$ has variance:

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

$\rightarrow$ gets narrower for increasing n
$\rightarrow$ estimation gets more precises (while σ^2 unknown)

$\rightarrow$ wanted: unbiased estimator for σ^2 Attempt for $S^2 = \frac{1}{n}\sum_{i=0}^{n-1}(X_i - \overline{X})^2$:

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

S^2 is biased, but $\frac{n}{n-1}S^2$ is unbiased.

Advanced subjects: confidence intervals, resampling, hypothesis tests, principal component analysis, clustering, fits, ...
(see A.K. Hartmann, *Big Practical Guid to Computer Simulations*, (World-Scientific, 2015) [3])