Computer Simmulation

A practical approach to simulation

Semra Gündüç

gunduc@ankara.edu.tr

Ankara University Faculty of Engineering,

Department of Computer Engineering

Pseudo Random Numbers

Random Number Generation

Properties of Random Numbers

- Important desired properties:
  - No correlations
  - Long periods
  - Follow well-defined distribution
  - Fast implementation
  - Reproductibility
  - Uniformity

Random Number Generation

Generation of Pseudo-Random Numbers

- Randomness is a part of nature:
  - radioactive decay,
  - positions of the stars,
  - intervals between passing individuals or viechiles,
  - ...
- Since computers are working with deterministic rules true random number generation is imposible.
- Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

John von Neuman (1951)

Random Number Generation

Generation Random or of Pseudo-Random Numbers

- Hardware ramdom number generators exist:
  - Electrical flicker noise
  - Photon emission from a semiconductor
- Software pseudo-random number generators:
  - Find some simple deterministic formulas (pseudo-random number generator) whose results immitate random numbers.
  - Check if this pseudo-random number generator produce desired properties of random numbers,
  - The true chek is to use this pseudo-random number generator to simulate a physical phenomena.

Random Number Generation

Generation of Pseudo-Random Numbers

- "Pseudo", because generating numbers using a known method removes the potential for true randomness.
- Goal: To produce a sequence of numbers in $[0, 1]$ that simulates, or imitates, the ideal properties of random numbers (RN).
- Important considerations in RN routines:
  - Fast
  - Portable to different computers
  - Have sufficiently long cycle
  - Replicable
  - Closely approximate the ideal statistical properties of uniformity and independence

Random Number Generation

Generation of Pseudo-Random Numbers

- Since the formula gives a deterministic relation between the given integers,
  - starting with the same seed result the same sequence.
  - This may seem a disadvantage; Infact this provide repeatibility, for debugging, comparissons and testing.
  - In order to avoid repeating the same sequence extra care must be given to choose different initial value(s) at each run of the same program. The reasons of running the same program repeatedly will be discussed in the Monte-Carlo Simulation section

Random Number Generation

Commonly Used Simple Pseudo Random Number Algorithms:

- Congruential (Lehmer, 1948)
- Lagged-Fibonacci (Tausworth,1965)
- **Referances:**
  - Numerical Recipes in C
  - D.E.Knuth: „ The Art of Programming: Seminumerical Algorithms“ 3rd ed. (Addison – Wesley, 1997) Vol. 2, Chapt. 3.3.1
  - J.E. Gentle, „Random Number Generation and Monte Carlo Methods“ (Springer, 2003)

Random Number Generation

Linear Congruential Generator

One of the most popular algorithm is the Linear Congruential Generator. Linear Congruential Generator implemented on most computers which generates a sequence of pseudo-random numbers according to the algorithm:

(1) $$x_{n+1} = (ax_n + c) mod M$$

where, x_n, M, a and c are integers. x_0 is called seed of the sequence.

Random Number Generation

Example of congruential RNG

- Park and Miller (1988):

```
const int M = 2147483647;
const int a = 16807;
int irnd = 77; //seed [0,M)
double rnd;// rnd,  [0,1)
irnd = (a * irnd) \% M;
rnd=(double) irnd/(double) M;
print irnd,rnd;
```

Random Number Generation

Maximum Period of a Pseudo-random Number Generator

- Since all integers are less than M the sequence must repeat after at least $M - 1$ iterations,
- i.e. the maximal period is $M - 1$.
- if $c = 0$, $x_0 = 0$ is a fixed point and cannot be used.
- Robert D. Carmichael proved (1910) that,
  - one gets the maximal period if M is a Mersenne prime number and,
  - The smallest integer number a for which $a^(M - 1)$ mod $M = 1$

Random Number Generation

Maximum Period of a Pseudo-random Number Generator

Mersenne prime numbers

The selection of the values for a, c, and M, and X_0 drastically affects the statistical properties and the cycle length.

Marin Mersenne, 1626

(2) $$M_q = 2^q - 1 \quad q \ \text{prime}$$

- For 32-bit word computers $M = 2^{31} = 12147483647$, which is a prime number, gives long sequence of random numbers. (Used also for random() and rand().)

Random Number Generation

Maximum Period of a Pseudo-random Number Generator

- One can use different set of a, c and M.
- An Linear Congruential Generator with parameters a, c and M has perion length M if and only if,
  1. gcd(c,M) = 1;
  2. $a \; Mod \; p = 1)$ for every prime p dividing M;
  3. $a \; Mod \; 4 = 1$ if M ia a multiple of 4

$$(a \times x_{k-1} + c) \leq (2M) \quad (a \sim x_i \sim (M)),$$

Random Number Generation

Maximum Period of a Pseudo-random Number Generator

Some sets of good a, c and M combinations are given by the table:

M	a	c
259200	7141	54773
134456	8121	28411
243000	4561	51349
714025	1366	150889

Table 1: Some good M, a and c combinations

Random Number Generation

Simple Linear Congruential Generator

```
#include <stdio.h>
#include <math.h>
int main(){
   int i;
   long int x =1237,ic=417,ia=117,M=4717;
   float r;
   for ( i=1; i<=10; i++ ){
       x = ( x * ic + ia ) % M ;
       r = (double) x / (double) M;
       printf("%3d   %7.5f\n ",i,r);
     }
   return(0);
}
```

Random Number Generation

Using System Provided Random Numbers

- C programming language provides a random number generator based on Linear Congruential algorithm with $M = 2^{31} - 1$.
- This generator can be activated by calling either one of these programs:
  - random()
  - rand()
- Both random() and rand() use a fixed seed.
- Sequence of random numbers are generated within the calling program and subprograms.
- When the same program is activated again the same sequence of random numbers can be reproduced.

Random Number Generation

Using System Provided Random Numbers

- Changing random number sequence is possible by calling the program,
  - srandom(unsigned long)
  - srand(unsigned long)
- srand() takes a long integer to initiate the random nember sequence.
- In order to further randomize the initial seed one can use the time() function which is defined in the hader file time.h
- After including time.h into your program, to set a new random number seed each time program is runed:
  - srand(time(NULL));

Random Number Generation

Using System Provided Random Numbers

Using srand, srandom(), random() and rand()

```
//This program Produce integer random numbers
//between $0$ and $2^{31}-1$
#include <stdio.h>
int main(){
   int i;
   srandom(127717); //set seed
   for ( i=0; i < 10; i++ )
     printf(" %d  %d \n ",i,rand());
   srandom(127717); //set seed
   for ( i=0; i < 10; i++ )
     printf(" %d  %d \n ",i,random());
}
```

Random Number Generation

Using System Provided Random Numbers

- Integer random numbers have some limitted use.
- RAND_MAX which is the largest possible number to be produced, is provided in both stdlib.h and math.h
- Dividing rand() or random() by this number one can obtain linearly distributed random numbers in the range $[0, 1)$

Random Number Generation

Using System Provided Random Numbers

```
//This program Produce integer random numbers
//and random numbers between $0$ and $1$
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define frand() ((double)random()/(double)RAND_MAX
int main(){
  int i;
  srandom(time(NULL)); //set seed
  for ( i=0; i < 10; i++ ){
    printf(" %d  %d ",i,random());
    printf("% f ",(double)rand()/(double)RAND_MAX);
    printf(" %f \n ",frand());
  }
}
```

Random Number Generation

A set of integer random numbers (rand())

1553815959	556437479	573295497	339398630
207228801	2090041015	68418236	708757024
891197014	1893431560	688619835	1157187795
1463216227	188270770	1769319866	988937994
1163497225	64224048	1500957925	1945850552
129706340	769155680	1051505658	35530749
121144548	292193613	31819671	1994671593
476584321	289859384	1312514152	137185691
447083025	2030400280	846296863	1885809649
2071902895	654311826	1972957647	914715099

Random Number Generation

A set of floating point random numbers

0.17664	0.04163	0.00099	0.00000
0.48722	0.09230	0.09133	0.36460
0.83129	0.23318	0.45443	0.52675
0.05083	0.55609	0.56806	0.93173
0.29820	0.25236	0.01891	0.76705
0.51543	0.92026	0.53156	0.87598
0.57061	0.88631	0.18842	0.81043
0.11835	0.98489	0.81527	0.07677
0.25331	0.10092	0.78448	0.89391
0.68092	0.67888	0.37838	0.01984

Random Number Generation

Tests for random numbers

- It is wise to test a random-number generator to obtain a numerical measure of its uniformity and randomness before using in any scienticic project.
- In the litearure there are various examples of ambaresment due to some unexpected behaviour of the random numbers.
- In fact, some tests are simple enough to perform before actually using the random number generator.
- Even the following list of test are not complete,
- Best test is actual calculation of a known quantity.

Random Number Generation

Tests for random numbers

There are a large class of tests. The most elementary test is to check if the random number generator is a biased one or not.

A list of simple random number tests:

- Average is 0.5
- Check distribution
- Correlations should vanish
- n-cube-test
- χ^2 test: partial sums follow a Gaussian
- Average of each bit is 0.5
- Spectral test: no peaks in Fourier transform

Random Number Generation

Tests for random numbers

- The most obvious test for randomness and uniformity is to look at the numbers generated.
- Observe that
  - wether they all lie between 0 and 1,
  - they appear to differ from each other, and
  - there is no obvious pattern (like 0.3333).

Random Number Generation

Tests for random numbers

- The most obvious test for randomness and uniformity is to look at the numbers generated.
- Observe that
  - wether they all lie between 0 and 1,
  - they appear to differ from each other, and
  - there is no obvious pattern (like 0.3333).

Random Number Generation

Tests for random numbers

- Average test:
  - Generate uniformly distributed random numbers in the range 0 and 1.
  - Sum all numbers and sum the squares of the random numbers.
  - Calculate the average and the standart deviation,

$$< r > = \frac{1}{N} \sum_{i=1}^{N} r_i$$

$$sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (r_i - < r >)^2$$

Random Number Generation

Tests for random numbers

- Distribution Test
  - calculate the index according to the value of the random number, sampled from a uniform distribution between 0 and 1 .
  - Calculate the range and add one to the range.

$$ind = N_{Bins} \times r_i$$
$$Distribution[ind] = Distribution[ind] + 1$$

  - Repeat the same process as many times as possible.

Random Number Generation

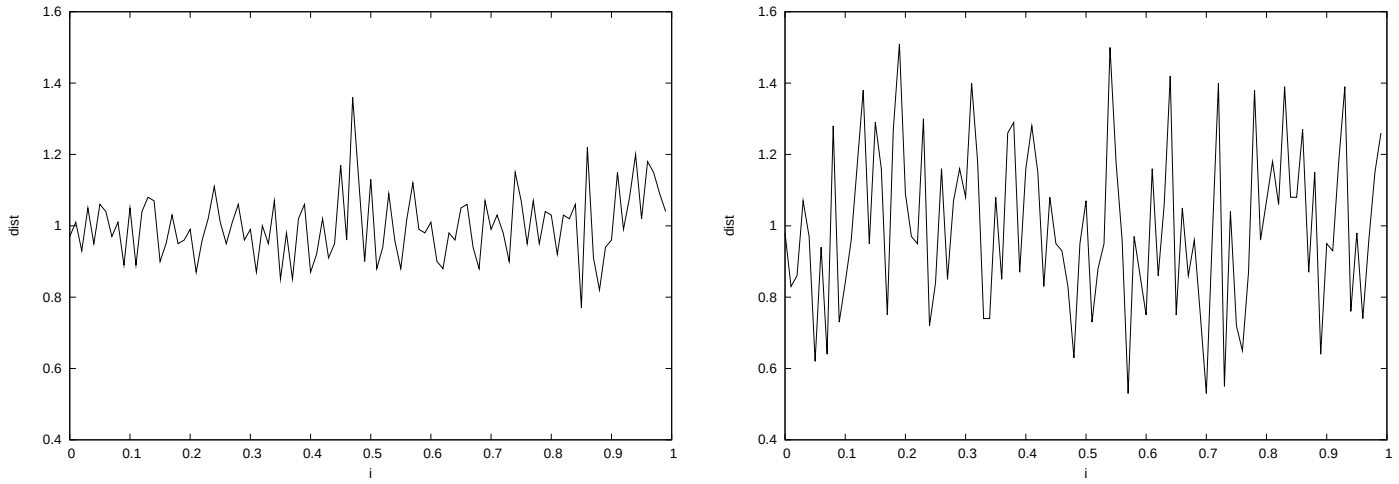

Figure 1: Good and bad random number generators.

Random Number Generation

Tests for random numbers

- **Square Test**
  - Step 1: Calculate two consequetive random numbers r_i and r_{i+1}
  - Step 2: Plot x versus y, $x = r_i$ and $y = r_{i+1}$
  - Step 3: Repeat starting from step1.
- If there exist short range correlations in the sequence it will appear as paralel lines on the plot.

Random Number Generation

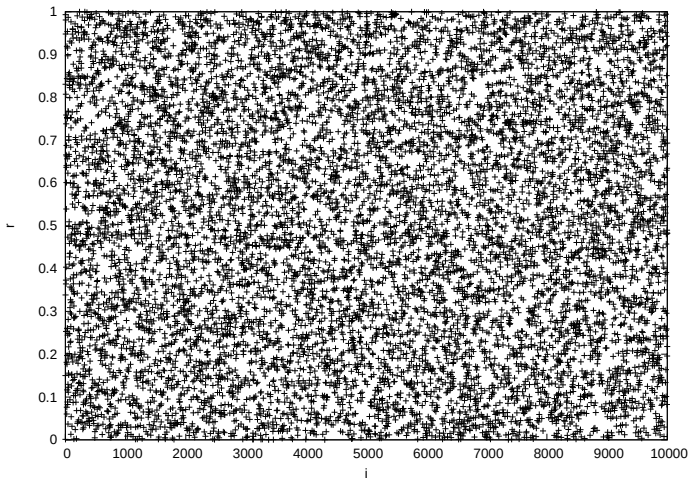

Figure 2: Good and bad random number generators.

Random Number Generation

Usually one requires random numbers in a certain range. Most of the applications require random numbers in the range 0 and 1. Once uniformly distributed random integers are obtained, uniformly distributed random numbers in the interval $[0, 1)$ is obtained by dividing the random integer with the largest number M,

(3) $$r_n = \frac{x_n}{M} \quad \text{for} \quad 0 \leq r_n < 1$$

if one require both limits to be included, then

(4) $$r_n = \frac{x_n}{M-1} \quad \text{for} \quad 0 \leq r_n \leq 1$$

will produce uniformly distributed random numbers in the closed interval, $[0, 1]$.

Random Number Generation

Changing the Interval of the Uniform Random Numbers

- It is easy to change the interval of the created uniform random numbers.
- This is simply done by choosing a new variable which takes values within the desired interval.
- If random numbers r_i's are created in the interval $[0, 1)$, then this interval can be changed to $[A, B)$ by choosing a new random variable x,

$$x_n = (B - A) \times r_n + A \quad 0 \leq r_n < 1 \tag{5}$$

where $A \leq x_n < B$.

Random Number Generation

Integer random numbers

- If random numbers r_i's are created in the interval $[0, 1)$, random sequence of integers, 0's and 1's can be created.
  - $$ir_n = (int)(r_n + 0.5)$$

    will give 0 for $r < 0.5$ and 1 for $r > 0.5$.
  - Since r is uniformly distributed, with equal probablility 0's and 1's are created.
- Similarly, integer random numbers consist of only 1 and -1 can be created.
- $$ir_n = 2 \times (int)(r_n + 0.5) - 1$$

Random Number Generation

Integer random numbers

- Random sequences of 0 (1) and 1 (-1) are used in simulations where the individual units can take only two values.
- This very common even in the modern society since many decissions require yes or no questions. Something is either on or off also a very common in daily life.

Random Number Generation

Changing the interval of the uniform random numbers

- In some situations integer random numbers are necessary.
- If the range of the integer random numbers is $[0, M)$
- For uniformly distributed random integer values between 0 and $N-1$ one can use the integer random numbers generated by using the relation given in Eq. (1),

(6) $$ir_n = x_n \% N \quad \text{where} \quad 0 \leq x_n < M$$

here ir_n takes all values $0, 1, \ldots, N-1$.

Pseudo Random Numbers II

Random Number Generation

Properties of Random Numbers

- Random Numbers, r_i, must be independently drawn from a distribution with :

$$\int_{-\infty} \infty p(x)\, dx = 1 \quad \text{and} \quad p(x) > 0.$$

- Examples:
  - Homogeneous (linear)
  - Gaussian,
  - Poisson,
  - Arbitrary distribution.
  - ...

Random Number Generation

Random Numbers Sampled According to a Desired Distribution
Given a uniform distribution ofrandom numbers, it is easy to sample any distribution specified by an integrable function by simple change of variables. If we define

$$y(x) = \int^x dx' P(x') \quad \text{then}$$

$$\frac{dy}{dx} = P(x) \tag{7}$$

Then we may write,

$$\int dx P(x) f(x) = \int dy f(x(y)) \tag{8}$$

Thus when the variable y is sampled uniformly the inverse

Random Number Generation

Examples:

$P(x) = 2x$

(9) $$\int_0^1 f(x)2xdx = \int_0^1 f(x)d(x^2) = \int_0^1 f(y^{1/2})dy$$

so that sampling values of y uniformly distributed on (0,1)

yields values of $x = y^{1/2}$ distributed with $P(x) = 2x$ on $(0, 1)$

Random Number Generation

Exponantial Distribution

Consider the integral,

(10)
$$I = \int_0^\infty dx e^{-x} g(x)$$

the most natural choise of weight is, e^{-x} then the distribution can be obtained,

$$dy = dx e^{-x}$$
$$y = \int_0^x e^{-x'} dx' = 1 - e^{-x}$$

(11)
$$x = -\ln(1 - y)$$

Random Number Generation

random Numbers with Normal Distribution

Similarly Gaussian distribution for which number of points in a differential area is proportional to

(12) $$e^{-\frac{1}{2}(x^2+y^2)}dxdy$$

In terms of polar coordinates $r = (x^2 + y^2)^{1/2}$, $\theta = \arctan \frac{y}{x}$ the distribution is,

(13) $$e^{-\frac{1}{2}r^2}rdrd\theta$$

if $u = 1/'r^2$ the distribution is

(14) $$e^{-u}dud\theta$$

Random Number Generation

If we generate u between 0 and ∞ with an exponential distribution and θ uniformly 0 and 2π then the corresponding values of

$$x = (2u)^{1/2} cos\theta$$
$$y = (2u)^{1/2} sin\theta \quad (15)$$

will distribute normally. Or if ξ_1 and ξ_2 are uniformly distributed random numbers,

$$x = (-2\ln\xi_1)^{1/2} cos(2\pi\xi_2)$$
$$y = (-2\ln\xi_1)^{1/2} sin(2\pi\xi_2) \quad (16)$$

gives normally distributed random numbers.

Random Number Generation

Random Numbers with Arbitrary Distribution

In above examples the weight function is choosen such that the exact integration was possible. If exact integration is not possible still there are some methods to solve this problem.

$i)$ Let us imagine tabulating the values $X^{(j)}$ for which the incomplete integral of w takes on a series of uniformly spaced values $Y^{(j)} \equiv j/N \;\; j = 0, 1, \ldots, N$ That spans the interval (0,1). Thus,

(17)
$$y^{(j)} = \frac{j}{N} = \int_0^{x^{(j)}} dx' W(x')$$

Random Number Generation

Through simple discritization,

$$\frac{y^{(j+1)}-y^{(j)}}{x^{(j+1)}-x^{(j)}} = W(x^{(j)})\; y^{(j+1)} - y^{(j)} = \frac{1}{N}$$

$$x^{(j+1)} = x^{(j)} + \frac{1}{NW(x^{(j)})} \tag{18}$$

a convenient initial value may be $x^{(0)} = 0$

Random Number Generation

Von Neumann Rejection Method:

Let F(x) be a positive function everywhere greater than or equal to a probablity distribution P(x) to be sampled. If points are generated uniformly in the plane below F(x) and only those which are also below P(x) are accepted, then accepted points will be distributed according to P(x). Operationally, a random variable x_i is selected with probablity proportional to F(x) (i.e. P(x)=F(x)/$\int$!F(x)) and a random number ξ uniformly distributed on the interval (0,1) is generated. The value x_i is accepted if $P(x_i)/F(x_i) > \xi$; otherwise it is rejected. Clearly this procedure is only efficient if