Random number generators

Anna Simon

September 4, 2015

Random number generator

A **pseudo-random number generator** (PRNG) is a program that takes a starting number (seed) and performs mathematical operations on it to transform it to another number. If the algorithm is complex enough the numbers will seem random.
Features of a good PRNG:

- the numbers should be generated with uniform probability
- the method generating a sequence should NOT be obvious or predictable
- the numbers should have a good dimensional probability (low, medium and high values should be all mixed together and not generated in "packages")
- it should have a long period
  All PRNG are periodic, the longer the period the less likely it is for the numbers to repeat)

PRNG on the market

There are several PRNG available:

- Mersenne Twister: period of $219937 - 1 \approx 10^{6001}$. The MT implementation mt19937( ) is part of many languages and scientific libraries such as Matlab, R, Python. Best on the market.
- WELL generators: The name stands for Well Equidistributed Long-period Linear.
- drand48: The Unix built-in family of generators drand48( ) is actually based on a linear congruential generator with 48-bit arithmetic. Period of $\approx 10^{14}$. (Not very good numerical simulations)

rand()

C++ provides a build in PRNG implemented as two functions in `cstdlib`:

- `srand()` sets the initial seed value. It is called only once
- `rand()` starting from the seed provided by `srand()` generates the next random number.

Usually the system time is used as an initial seed: `srand(time(0))`.

`rand()` is not the best generator, the range of simulated numbers is limited to the size of int (e.g., 32767 (on 16-bit systems)):

- cannot generate large numbers
- when generating values between 0 and 1. gives 'poor resolution'

How to use rand()

```cpp
#include <iostream>
#include <cstdlib> // for rand() and srand()
#include <ctime> // for time()
using namespace std;

int main()
{
    // print out the maximum value that can be generated
    cout << RAND_MAX<< endl;

    srand(time(0)); // set initial seed value to system clock

    for (int nCount=0; nCount < 100; ++nCount)
    {
        cout << rand() << "\t";

        if ((nCount+1) % 5 == 0)
            cout << endl;
    }
    return 0;
}
```

Set `srand(1)` and run the code multiple times.

Examples

How to generate a "histogram":

```
1  int hist[100] = {0}; //initialize the array for the histogram
2  int randNumber;
3
4  //generate a number in a range 0 to 1000
5  randNumber = rand() % 1000;
6
7  // scan through the array and increment the histogram "bin"
8  for (int ii=0; ii<100;ii++)
9  {
10     //histogram binned in 10.
11     if (randNumber>=ii*10 && randNumber<(ii+1)*10)
12     {
13         hist[ii]++;
14         //debug: check if the number is saved in the bin we want
15         //cout << randNumber << " " << ii << endl;
16         break;
17     }
18 }
```

- generate 10,000 events using the code above, copy the output to e.g. Excel and draw the generated histogram.
- Modify the code to generate random numbers in the range: 0 to 1.

How to generate various distributions

Generating random variables in 3 steps:

- Generate a random number R, between 0 and 1 (with a random number generator).
- Inverse the Probability Distribution Function (PDF) of your distribution. Suppose $G(z)$ is the inverse function.
- Your random number which obeys the specific distribution is $x = G(R)$.

PDF: $y = exp(-x)$, inverse function: $x = -ln(y)$
using PRND generate y and using inverse function calculate x

- for Gaussian distribution Box–Muller typically transformation is used

Examples

Uniform distribution around `mean` value:

```
1 float Uniform(float mean){
2     float R;
3     R = (float)rand()/(float)(
          RAND_MAX);
4     return 2*mean*R;
5 }
```

Exponential distribution with a given decay constant:

```
1 float Exponential(float lambda){
2     float R;
3     R = (float)rand()/(float)(
          RAND_MAX);
4     return -log(R)/lambda;
5
6 }
```

Gaussian distribution with a given centroid and standard deviation:

```
1 float Normal(float mean, float
      stdev)
2 {
3     float R1;
4     R1 = (float)rand()/(float)(
          RAND_MAX);
5     float R2;
6     R2 = (float)rand()/(float)(
          RAND_MAX);
7     return mean + stdev*cos(2*3.14*R
          1)*sqrt(-log(R2));
8
9 }
```