Step 1B Linear Congruential Generator

1. Requirements for simulating stochastic systems
   1. **Random number requirements for simulation.** Inherent random elements: arrival times, service times, failure of system

Requirement of simulation: produce a sequence of numbers, that are, or appear to be, random

* 1. **Pseudo-random numbers**: Remark on random “observations” vs. “variables”. The goal is determine a sequence of numbers, u1, u2, u3.,, un, .., uN. with the properties

un+1 is completely determined by un

* 1. Generate U(0,1), then generate other non-uniform distribution

4. Desirable properties of generators

* + - 1. Appear (statistically) to be u(0,1) and uncorrelated/ independent, Turing test: human vs. computer
      2. Fast with low memory requirements
      3. Reproducible for debugging, for more precise comparison of alternative system designs
      4. Able to produce separate streams

B. Formula:

xn+1 = (*a* xn +c) **mod** m

where un is the [sequence](http://en.wikipedia.org/wiki/Sequence) of pseudorandom values, and

m  the "[**modulus**](http://en.wikipedia.org/wiki/Modulo_operation)"

a, 0<a<m : "**multiplier**"

c 0<=c<m: the "**increment**"

xn=0  the "seed" or "start value"

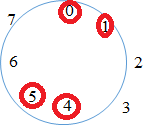
un=xn/m: uniformly distributed (0,1) value

example: The secret key in the DES encryption

2. example: m=8, a =3, c =1, x0 =1

xi possible values = {0,1,4,5}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 3xn-1+1 | *X*n+1 | ui =xn/m |  |
| 0 |  | 1 | 0.125 |  |
| 1 | 4 | 4 | 0.500 |  |
| 2 | 13 | 5 | 0.625 |  |
| 3 | 16 | 0 | 0.000 |  |
| 4 (repeated | 1 | 1 | 0.125 |  |
| 5 | 4 | 4 | 0.500 |  |
| 6 | 13 | 5 | 0.625 |  |



Full cycle: travel direction, full coverage, appeal random?

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | ***m*** | **(multiplier) *a*** | **(increment) *c*** |
| [MMIX](http://en.wikipedia.org/wiki/MMIX) by [Donald Knuth](http://en.wikipedia.org/wiki/Donald_Knuth) | 264 | 6364136223846793005 | 1442695040888963407 |
| C/C++ | 232 | 22695477 | 1 |
| [Microsoft Visual/Quick C/C++](http://en.wikipedia.org/wiki/Visual_C%2B%2B) | 232 | 214013 (343FD16) | 2531011 (269EC316) |

1. The LCG with m=231-1, a = 75, c =0 has full period, and good statistics property: the period for this LCG is about 2.1\*109

But it can gen exhausted in less than 30 min on a 1GHz CPU.

Solutions: => multiple recursive generators and composite generators: MRGs

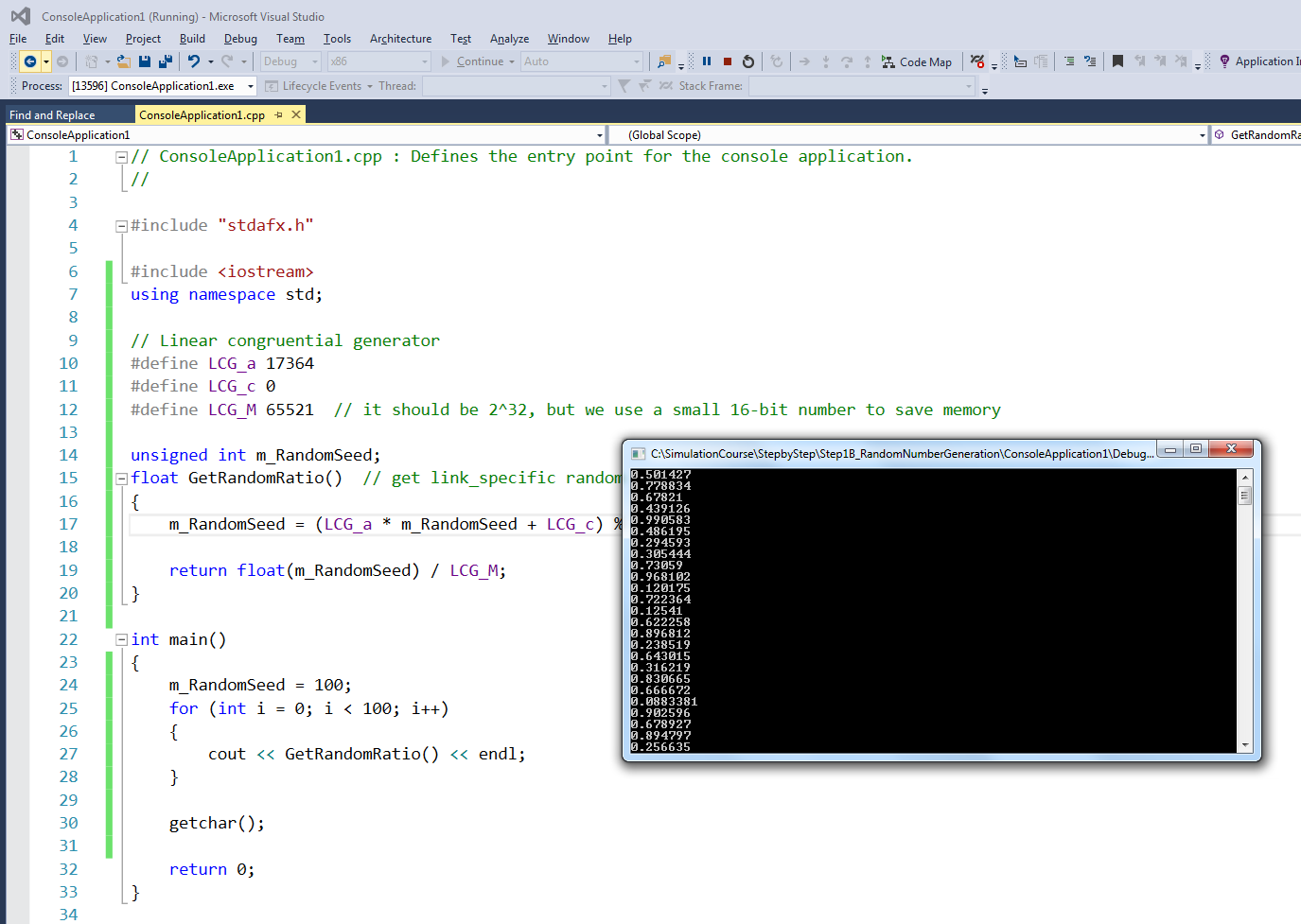
xn+1 = (*a1* xn+ *a2* xn-1  +…+ *aq* xn-q+1++c) **mod** m

period is as large as mq-1, and we now have q seeds.

Composite generator:

un= (δ*1* xn+δ*2* xn-1  +…+δ*q* xn-q+1++c)/m:

Different statistics test for independence, randomness, coverage



1.    Random number seeds

2.    Calculate mean, variances,

3.    PDF, and CDF based on samples

// ConsoleApplication1.cpp : Defines the entry point for the console application.

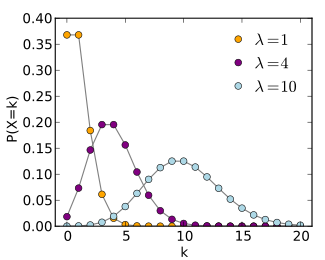
//

|  |  |
| --- | --- |
| // Linear congruential generator  #define LCG\_a 17364  #define LCG\_c 0  #define LCG\_M 65521 // it should be 2^32, but we use a small 16-bit number to save memory  unsigned int m\_RandomSeed; | Global definitions |
| float GetRandomRatio() // get link\_specific random seed  {  m\_RandomSeed = (LCG\_a \* m\_RandomSeed + LCG\_c) % LCG\_M; //m\_RandomSeed is automatically updated.  return float(m\_RandomSeed) / LCG\_M;  } | External function |
| int main()  {  m\_RandomSeed = 100;  for (int i = 0; i < 100; i++)  {  cout << GetRandomRatio() << endl;  }  getchar();  return 0;  } | Call random number generators |
|  |  |

1. Inversion methods for generating nonuniform random variables

Poisson distribution discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event.

In [probability theory](https://en.wikipedia.org/wiki/Probability_theory) and [statistics](https://en.wikipedia.org/wiki/Statistics), the **exponential distribution** (also known as **negative exponential distribution**) is the [probability distribution](https://en.wikipedia.org/wiki/Probability_distribution) that describes the time between events in a [Poisson process](https://en.wikipedia.org/wiki/Poisson_process), i.e. a process in which events occur continuously and independently at a constant average rate.



\!f(k; \lambda)= \Pr(X=k)= \frac{\lambda^k e^{-\lambda}}{k!},

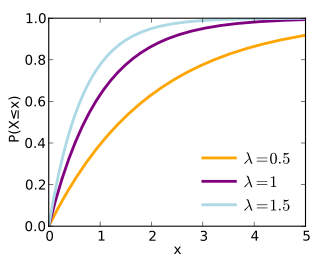
F(x) as CDF(x)

0<F(x)<1

Let u = F(x)

Then x = F-1(u)





Example: suppose x ~ exp(λ)

PDF(x) = λexp(-λx) for x>=0

F(x) = 1- exp(-λx)

Let u = 1- exp(-λx)

X = -1/λ\*ln(1-u)

Of course, 1-u between 0, 1

Example, P(X=x)

λ = 1

u= 0.4, x = -1\*LN(0.6)= 0.510826

u= 0.8, x = -1\*LN(0.2)= 1.609438

u= 0.9, x = -1\*LN(0.1)= 2.302585

λ = 0.5

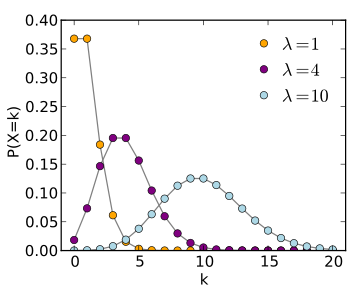
u= 0.4, x = -1/λ\*LN(0.6)= 1.021651

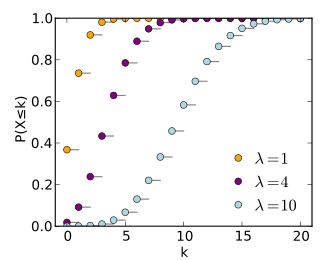
u= 0.8, x = -1/λ\*LN(0.2)= 3.218876

u= 0.9, x = -1/λ\*LN(0.1)= 4.60517

1. Example of Poisson Distribution (from Wiki)

Suppose someone typically gets 4 pieces of mail per day. There will be, however, a certain spread: sometimes more pieces of mail arrive, sometimes a fewer, and occasionally, no mail is received at all.[[](http://en.wikipedia.org/wiki/Poisson_distribution#cite_note-umass1-2) Given only the average rate, for a certain period of observation (pieces of mail per day, phonecalls per hour, etc.), and assuming that the process, or mix of processes, that produces the event flow is essentially random, the Poisson distribution specifies how likely it is that the count will be 3, or 5, or 10, or any other number, during one period of observation.





Other methods