CENG 222 Probability and Statistics

Computer Simulations and Monte Carlo Methods

Nesli Erdoğmuş & Burak Galip Aslan

Contents

Introduction

Computer simulations refer to a regeneration of a process by writing a suitable computer program and observing its results. **Monte Carlo methods** are those based on computer simulations involving random numbers.

The main purpose of simulations is estimating such quantities whose **direct computation is**

- **complicated,**
- **risky,**
- **consuming,**
- **expensive, or**
- **impossible**.

Introduction

- Monte Carlo methods are mostly used for the computation of probabilities, expected values, and other distribution characteristics.
- Recall that probability can be defined as a **long-run proportion.** With the help of random number generators, computers can actually **simulate a long run**. Then, probability can be estimated by a mere computation of the associated **observed frequency**.
- The **longer run** is simulated, the **more accurate result** is obtained. Similarly, one can estimate expectations, variances, and other distribution characteristics from a long run of simulated random variables.

Introduction

Monte Carlo methods and Monte Carlo studies inherit their name from Europe's most famous *Monte Carlo casino* in Figure 5.1 located in Principality of Monaco on the Mediterranean coast since the 1850s. Probability distributions involved in gambling are often complicated, but they can be assessed via simulations. In early times, mathematicians generated extra income by estimating vital probabilities, devising optimal gambling strategies, and selling them to professional gamblers.

Applications and examples

- Let us briefly look at a few examples, where **distributions are rather complicated**, and thus, Monte Carlo simulation appears simpler than any direct computation.
- Notice that in these examples, instead of generating the actual devices, computer systems, networks, viruses, and so on, we **only simulate the associated random variables**.
- For the study of probabilities and distributions, this is entirely sufficient.

Applications and examples

Example 5.1 (Forecasting)

Given just a basic distribution model, it is often very difficult to make reasonably remote predictions. Often a one-day development depends on the results obtained during all the previous days. Then prediction for tomorrow may be straightforward whereas **computation of a one-month forecast is already problematic**.

Applications and examples

Example 5.1 (Forecasting)

On the other hand, simulation of such a process can be easily performed day by day (or even minute by minute). Based on present results, **we simulate the next day**. Now we “know it,” and thus, **we can simulate the day after that, etc**.

For every time $\boldsymbol{n}$, we simulate $\boldsymbol{X_{n+1}}$ based on already known $\boldsymbol{X_1}$, $\boldsymbol{X_2}$, ..., $\boldsymbol{X_n}$. Controlling the length of this do-loop, we obtain forecasts for the next days, weeks, or months. Such simulations help predict future paths of storms and hurricanes as well as possibilities of flash floods.

Applications and examples

Example 5.2 (Percolation - *"sızma"***)**

Consider a network of nodes. Some nodes are connected, say, with transmission lines, others are not (mathematicians would call such a network a graph). A signal is sent from a certain node. Once a node $\boldsymbol{k}$ receives a signal, it sends it along each of its output lines with some probability $\boldsymbol{p_k}$. After a certain period of time, one desires to estimate the proportion of nodes that received a signal, the probability for a certain node to receive it, etc.

Applications and examples

Example 5.2 (Percolation - *"sızma"***)**

This general *percolation* model describes the way many phenomena may *spread*. The role of a signal may be played by a computer virus spreading from one computer to another, or by rumors spreading among people, or by fire spreading through a forest, or by a disease spreading between residents.

Technically, simulation of such a network reduces to generating Bernoulli random variables with parameters $\boldsymbol{p_i}$. Line $\boldsymbol{i}$ transmits if the corresponding generated variable $\boldsymbol{X_i} = 1$. In the end, we simply count the number of nodes that got the signal, or verify whether the given node received it.

Applications and examples

Example 5.3 (Queuing)

A queuing system is described by a number of random variables. It involves:

- spontaneous arrivals of jobs,
- their random waiting time,
- assignment to servers, and,
- their random service time and departure.

In addition, some jobs may exit prematurely, others may not enter the system if it appears full, and also, intensity of the incoming traffic and the number of servers on duty may change during the day.

Applications and examples

Example 5.3 (Queuing)

When designing a queuing system or a server facility, it is important to evaluate its vital performance characteristics like:

- the job's average waiting time,
- the average length of a queue,
- the proportion of customers who had to wait, or that exit prematurely or could not enter,
- the proportion of jobs spending more than a time limit,
- the expected usage of each server,
- the average number of available (idle) servers at the time when a job arrives, and so on...

Applications and examples

Example 5.4 (Markov chain Monte Carlo)

There is a modern technique of generating random variables from rather complex, often intractable distributions, as long as conditional distributions have a reasonably simple form.

In semiconductor industry, for example, the joint distribution of good and defective chips on a produced wafer has a rather complicated correlation structure. On the other hand, the quality of each chip is predictable based on the quality of the surrounding, neighboring chips.

Applications and examples

Example 5.4 (Markov chain Monte Carlo)

Given its neighborhood, conditional probability for a chip to fail can be written, and thus, its quality can be simulated by generating a corresponding Bernoulli random variable with $\boldsymbol{X_i}$ = 1 indicating a failure.

According to the Markov chain Monte Carlo (MCMC) methodology, a long sequence of random variables is generated from conditional distributions. A wisely designed MCMC will then produce random variables that have the desired unconditional distribution, no matter how complex it is.

Applications and examples

In all the examples, we saw how different types of phenomena can be computer-simulated. However, **one simulation is not enough** for estimating probabilities and expectations.

After we understand how to program the given phenomenon once, we can embed it in a do-loop and repeat similar simulations a large number of times, generating **a long run**.

Since the simulated variables are random, we will generally obtain a number of different realizations, from which **we calculate probabilities and expectations as long-run frequencies and averages**.

Contents

Simulation of random variables

As we see, implementation of Monte Carlo methods reduces to **generation of random variables from given distributions**. Hence, it remains to design algorithms for generating random variables and vectors that will have the desired distributions.

Majority of computer languages have a random number generator that returns **only Uniformly distributed independent** random variables.

We will now discuss general methods of **transforming Uniform random variables**, obtained by a standard random number generator, i**nto variables with desired distributions**.

Random number generators

Obtaining a good random variable is not a simple task. How do we know that it is "truly random" and does not have any undesired patterns?

For example, quality random number generation is so important in coding and password creation that people design special tests to verify the "randomness" of generated numbers.

Random number generators

In the fields sensitive to good random numbers, their generation is typically related to an accurate measurement of some **physical variable**, for example, the computer time or noise.

Certain transformations of this variable will generally give the desired result.

e.g. atmospheric noise, www.random.org

Random number generators

More often than not, a pseudo-random number generator is utilized. This is nothing but a very long list of numbers.

A user specifies a random number seed that points to the location from which this list will be read. It should be noted that if the same computer code is executed with the same seed, then the same random numbers get generated, leading to identical results.

Often each seed is generated within the system, which of course improves the quality of random numbers.

Random number generators

```python
import random

random.seed(10)
print(random.random())
```

In the **random** module from the **Python** standard library, the **seed()** method is used to initialize the random number generator.

The random number generator needs a number to start with (a seed value), to be able to generate a random number.

By default the random number generator uses the current system time. Use the seed() method to customize the start number of the random number generator.

Note: If you use the same seed value twice you will get the same random number twice.

Random number generators

Instead of a computer, a table of random numbers is often used for small-size studies. For example, we can use Table A1 in Appendix.

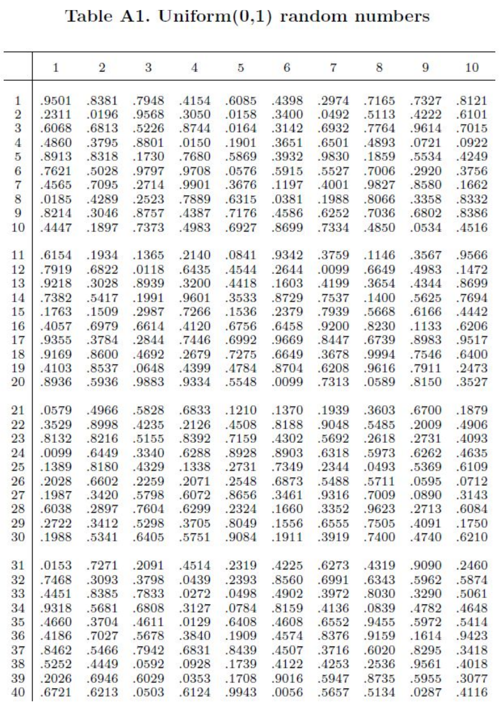


Table A1. Uniform(0,1) random numbers

	1	2	3	4	5	6	7	8	9	10
1	.9501	.8381	.7948	.4154	.6085	.4398	.2974	.7165	.7327	.8121
2	.2311	.0196	.9568	.3050	.0158	.3400	.0492	.5113	.4222	.6101
3	.6068	.6813	.5226	.8744	.0164	.3142	.6932	.7764	.9614	.7015
4	.4860	.3795	.8801	.0150	.1901	.3651	.6501	.4893	.0721	.0922
5	.8913	.8318	.1730	.7680	.5869	.3932	.9830	.1859	.5534	.4249
6	.7621	.5028	.9797	.9708	.0576	.5915	.5527	.7006	.2920	.3756
7	.4565	.7095	.2714	.9901	.3676	.1197	.4001	.9827	.8580	.1662
8	.0185	.4289	.2523	.7889	.6315	.0381	.1988	.8066	.3358	.8332
9	.8214	.3046	.8757	.4387	.7176	.4586	.6252	.7036	.6802	.8386
10	.4447	.1897	.7373	.4983	.6927	.8699	.7334	.4850	.0534	.4516
11	.6154	.1934	.1365	.2140	.0841	.9342	.3759	.1146	.3567	.9566
12	.7919	.6822	.0118	.6435	.4544	.2644	.0099	.6649	.4983	.1472
13	.9218	.3028	.8939	.3200	.4418	.1603	.4199	.3654	.4344	.8699
14	.7382	.5417	.1991	.9601	.3533	.8729	.7537	.1400	.5625	.7694
15	.1763	.1509	.2987	.7266	.1536	.2379	.7939	.5668	.6166	.4442
16	.4057	.6979	.6614	.4120	.6756	.6458	.9200	.8230	.1133	.6206
17	.9355	.3784	.2844	.7446	.6992	.9669	.8447	.6739	.8983	.9517
18	.9169	.8600	.4692	.2679	.7275	.6649	.3678	.9994	.7546	.6400
19	.4103	.8537	.0648	.4399	.4784	.8704	.6208	.9616	.7911	.2473
20	.8936	.5936	.9883	.9334	.5548	.0099	.7313	.0589	.8150	.3527
21	.0579	.4966	.5828	.6833	.1210	.1370	.1939	.3603	.6700	.1879
22	.3529	.8998	.4235	.2126	.4508	.8188	.9048	.5485	.2009	.4906
23	.8132	.8216	.5155	.8392	.7159	.4302	.5692	.2618	.2731	.4093
24	.0099	.6449	.3340	.6288	.8928	.8903	.6318	.5973	.6262	.4635
25	.1389	.8180	.4329	.1338	.2731	.7349	.2344	.0493	.5369	.6109
26	.2028	.6602	.2259	.2071	.2548	.6873	.5488	.5711	.0595	.0712
27	.1987	.3420	.5798	.6072	.8656	.3461	.9316	.7009	.0890	.3143
28	.6038	.2897	.7604	.6299	.2324	.1660	.3352	.9623	.2713	.6084
29	.2722	.3412	.5298	.3705	.8049	.1556	.6555	.7505	.4091	.1750
30	.1988	.5341	.6405	.5751	.9084	.1911	.3919	.7400	.4740	.6210
31	.0153	.7271	.2091	.4514	.2319	.4225	.6273	.4319	.9090	.2460
32	.7468	.3093	.3798	.0439	.2393	.8560	.6991	.6343	.5962	.5874
33	.4451	.8385	.7833	.0272	.0498	.4902	.3972	.8030	.3290	.5061
34	.9318	.5681	.6808	.3127	.0784	.8159	.4136	.0839	.4782	.4648
35	.4660	.3704	.4611	.0129	.6408	.4608	.6552	.9455	.5972	.5414
36	.4186	.7027	.5678	.3840	.1909	.4574	.8376	.9159	.1614	.9423
37	.8462	.5466	.7942	.6831	.8439	.4507	.3716	.6020	.8295	.3418
38	.5252	.4449	.0592	.0928	.1739	.4122	.4253	.2536	.9561	.4018
39	.2026	.6946	.6029	.0353	.1708	.9016	.5947	.8735	.5955	.3077
40	.6721	.6213	.0503	.6124	.9943	.0056	.5657	.5134	.0287	.4116

Simulation of random variables

Can a table adequately replace a computer random number generator? There are at least two issues that one should be aware of:

1. Generating a random number seed. A rule of thumb suggests to close your eyes and put your finger “somewhere” on the table. Start reading random numbers from this point, either horizontally, or vertically, or even diagonally, etc.

Either way, this method is **not** pattern-free, and it **does not guarantee** “perfect” randomness.

Simulation of random variables

Can a table adequately replace a computer random number generator? There are at least two issues that one should be aware of:

2. Using the same table more than once. Most likely, we cannot find a new table of random numbers for each Monte Carlo study. As soon as we use the same table again, **we may no longer consider our results independent of each other**.

Should we be concerned about it? It depends on the situation. For totally unrelated projects, this should not be a problem.

Simulation of random variables

One way or another, a random number generator or a table of random numbers delivers to us Uniform random variables $\mathcal{U}_1$, $\mathcal{U}_2$, . . . ∈ (0, 1).

The remaining slides are about how to transform them into a random variable (vector) $\mathcal{X}$ with the given desired distribution F($\boldsymbol{x}$).

<u>Notation</u> ‖ $U; U_1, U_2, \ldots$ = generated Uniform(0,1) random variables ‖

Discrete methods

At this point, we have obtained one or several independent Uniform(0,1) random variables by means of a random number generator or a table of random numbers.

Variables from certain simple distributions can be immediately generated from this.

Discrete methods

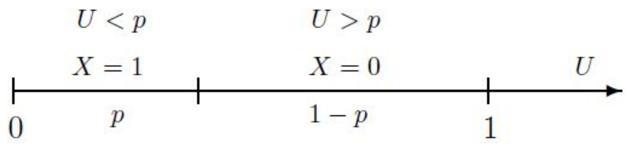
```
— R ————————
U <- runif(1);
X <- 1*(U < p);
```

```
— MATLAB ————
U = rand;      % Standard Uniform variable U
X = (U < p);   % converted into Bernoulli X
```

Example 5.5 (Bernoulli)

Simulate a Bernoulli trial with probability of success p. For a Standard Uniform variable $\mathcal{U}$, $\mathcal{X}$ is 1 if $\mathcal{U} < p$ and 0 if $\mathcal{U} \geq p$.

We call it "a success" if $\mathcal{X}$ is 1 and "a failure" if $\mathcal{X}$ is 0. Using the Uniform distribution of $\mathcal{U}$, we find that P{success} = P{$\mathcal{U}$<p} = p. Thus, we have generated a Bernoulli trial, and $\mathcal{X}$ has Bernoulli distribution with the desired probability p.



Discrete methods

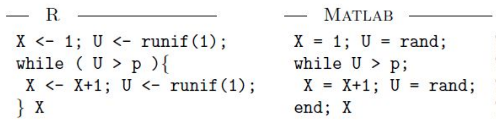
```
n <- 20; p <- 0.68;
U <- runif(n);
X <- sum(U < p);
```

```
n = 20; p = 0.68;
U = rand(n,1);
X = sum(U < p)
```

Example 5.6 (Binomial)

Once we know how to generate Bernoulli variables, we can obtain a Binomial variable as a sum of *n* independent Bernoulli. For this purpose, we generate with *n* Uniform random numbers ($\mathcal{U}_1$, $\mathcal{U}_2$, … $\mathcal{U}_n$), and using those we generate *n* Bernoulli random numbers ($\mathcal{X}_1$, $\mathcal{X}_2$, … $\mathcal{X}_n$) and the sum of $\mathcal{X}$s gives us a Binomial random number.

Discrete methods



```
— R ————————
X <- 1; U <- runif(1);
while ( U > p ){
 X <- X+1; U <- runif(1);
} X
```

```
— MATLAB ————
X = 1; U = rand;
while U > p;
 X = X+1; U = rand;
end; X
```

Example 5.7 (Geometric)

A while-loop of Bernoulli trials will generate a Geometric random variable. We run the loop of trials until the first success occurs. The number of trials gives us a Geometric random number.

Discrete methods

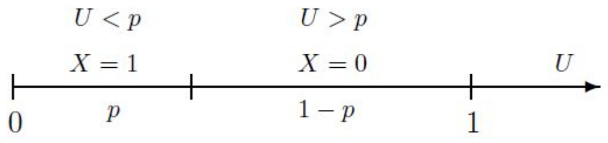
Last 3 examples show how to generate Bernoulli, Binomial, and Geometric variables using a random number generator that produces Standard Uniform variables.

What should we do if our distribution is not one of these?

Discrete methods

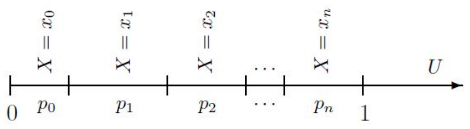
Arbitrary discrete distribution

- For the Bernoulli example, we divided the interval [0, 1] into two parts, p and $(1 - p)$ in length. Then we determined the value of X according to the part where the generated value of U fell.



- Apparently, this can be extended to any arbitrary discrete distribution.

Discrete methods



Consider an arbitrary discrete random variable X that takes values $x_0, x_1, \dots$ with probabilities $p_0, p_1, \dots,$

$$p_i = \boldsymbol{P}\{X = x_i\}, \qquad \sum_i p_i = 1.$$

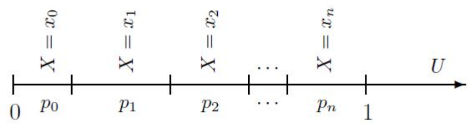
Algorithm 5.1 (Generating discrete variables)

1. Divide the interval [0, 1] into the following subintervals:

$A_0 = [0, p_0)$, $A_1 = [p_0, p_0 + p_1)$, $A_2 = [p_0 + p_1, p_0 + p_1 + p_2)$, etc.

Subinterval A_i will have length p_i; there may be a finite or infinite number of them, according to possible values of X.

Discrete methods



Algorithm 5.1 (Generating discrete variables)

1. Divide the interval [0, 1] into the following subintervals:

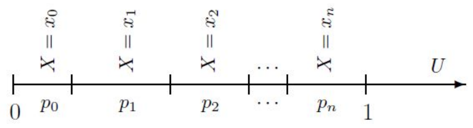
$A_0 = [0, p_0)$, $A_1 = [p_0, p_0 + p_1)$, $A_2 = [p_0 + p_1, p_0 + p_1 + p_2)$, etc.

Subinterval A_i will have length p_i; there may be a finite or infinite number of them, according to possible values of $\mathcal{X}$.

2. Obtain a Standard Uniform random variable from a random number generator or a table of random numbers.

3. If $\mathcal{U}$ belongs to A_i, let $\mathcal{X} = \boldsymbol{x}_i$.

Discrete methods



Algorithm 5.1 (Generating discrete variables)

From the Uniform distribution, it follows again that

$$P\{\mathcal{X} = \boldsymbol{x}_i\} = P\{\mathcal{U} \in A_i\} = p_i.$$

Hence, the generated variable $\mathcal{X}$ has the desired distribution.

Notice that contrary to Binomial or Geometric examples we have seen before, this algorithm is economic as it requires only one Uniform random number for each generated variable.

Values $\boldsymbol{x}_i$ can be written in any order, but they have to correspond to their probabilities p_i.

Discrete methods

Example 5.9 (Poisson)

Generate a Poisson variable with parameter λ = 5.

Recall from Section 3.4.5 that a Poisson variable takes values $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, … with probabilities

$$p_i = \boldsymbol{P}\{X = x_i\} = e^{-\lambda}\frac{\lambda^i}{i!} \text{ for } i = 0, 1, 2, \ldots$$

Following the algorithm, we generate a Uniform random number $\mathcal{U}$ and find the set A_i containing $\mathcal{U}$, so that

$$p_0 + \ldots + p_{i-1} \leq \mathcal{U} < p_0 + \ldots + p_{i-1} + p_i,$$

or, in terms of the cdf: $F(i-1) \leq \mathcal{U} < F(i)$.

Inverse transform method

We now turn attention to the generation of continuous random variables. The method will be based on the following theorem:

Theorem 2 Let $\mathcal{X}$ be a continuous random variable with cdf $F_{\mathcal{X}}(x)$. Define a random variable $\mathcal{U} = \mathcal{F}_{\mathcal{X}}(\mathcal{X})$. The distribution of $\mathcal{U}$ is Uniform(0,1).

Regardless of the initial distribution of $\mathcal{X}$, it becomes Uniform(0,1), once $\mathcal{X}$ is substituted into its own cumulative distribution function.

Inverse transform method

Arbitrary continuous distribution

In order to generate variable $\mathcal{X}$ with the given continuous cdf F, let us revert the formula $\mathcal{U} = F(\mathcal{X})$. Then $\mathcal{X}$ can be obtained from a generated Standard Uniform variable $\mathcal{U}$ as $\mathcal{X} = F^{-1}(\mathcal{U})$.

Algorithm 5.2 (Generating continuous variables)

1. Obtain a Standard Uniform random variable from a random number generator.

2. Compute $\mathcal{X} = F^{-1}(\mathcal{U})$. In other words, solve the equation $F(\mathcal{X}) = \mathcal{U}$ for $\mathcal{X}$.

Inverse transform method

Example 5.10 (Exponential)

How shall we generate an Exponential variable with parameter λ? According to Algorithm 5.2, we start by generating a Uniform random variable $\mathcal{U}$. Then we recall that the Exponential cdf is $F(x) = 1 - e^{-\lambda x}$ and solve the equation

$$1 - e^{-\lambda \mathcal{X}} = \mathcal{U}.$$

The result is: $\mathcal{X} = -1/\lambda \ln(1-\mathcal{U})$

$(1-\mathcal{U})$ has the same distribution as $\mathcal{U}$. Therefore, we can replace $\mathcal{U}$ by $(1 - \mathcal{U})$, and variable although different from $\mathcal{X}$, will also have the desired Exp(λ) distribution: $\mathcal{X} = -1/\lambda \ln(\mathcal{U})$

Inverse transform method

Discrete distributions revisited

Algorithm 5.2 is not directly applicable to discrete distributions because the inverse function F^{-1} does not exist in the discrete case. In other words, the key equation $F(\mathcal{X}) = \mathcal{U}$ may have either infinitely many roots or no roots at all.

Moreover, a discrete variable $\mathcal{X}$ has a finite or countable range of possible values, and so does $F(\mathcal{X})$. The probability that $\mathcal{U}$ coincidentally equals one of these values is 0. We conclude that the equation $F(\mathcal{X}) = \mathcal{U}$ has no roots with probability 1.

Inverse transform method

Discrete distributions revisited

Instead of solving $F(\boldsymbol{x}) = \mathcal{U}$ exactly, which is impossible, we solve it **approximately** by finding x, the smallest possible value of $\mathcal{X}$ such that $F(\boldsymbol{x}) > \mathcal{U}$.

Algorithm 5.3 (Generating discrete variables, revisited)

1. Obtain a Standard Uniform random variable from a random number generator or a table of random numbers.

2. Compute $\mathcal{X} = \min\{\boldsymbol{x} \in S$ such that $F(\boldsymbol{x}) > \mathcal{U}\}$, where S is a set of possible values of $\mathcal{X}$.

Inverse transform method

Discrete distributions revisited

Instead of solving $F(x) = U$ exactly, which is impossible, we solve it **approximately** by finding x, the smallest possible value of X such that $F(x) > U$.

Algorithm 5.3 (Generating discrete variables, revisited)

1. Obtain a Standard Uniform random variable from a random number generator or a table of random numbers.

2. Compute $X = \min\{x \in S$ such that $F(x) > U\}$, where S is a set of possible values of X.

Algorithms 5.1 and 5.3 are equivalent if values x_i are arranged in their increasing order in Algorithm 5.1.

Rejection method

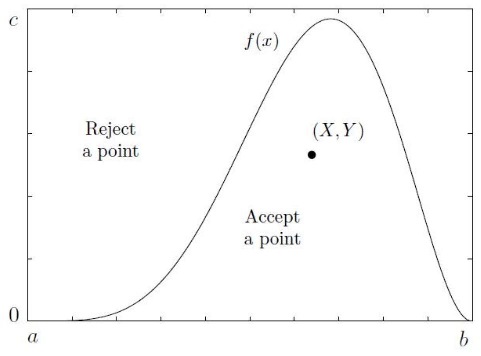
Besides a good computer, one thing is needed to generate continuous variables using the inverse transform method. It is a reasonably simple form of the cdf $F(x)$ that allows direct computation of $X = F^{-1}(U)$.

When $F(x)$ has a complicated form but a density $f(x)$ is available, random variables with this density can be generated by **rejection method**.

Rejection method

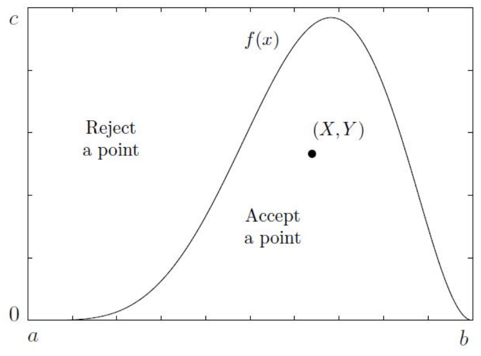
Consider a point (X, Y) chosen at random from under the graph of density f(x), as shown in the figure below.

What is the distribution of X?



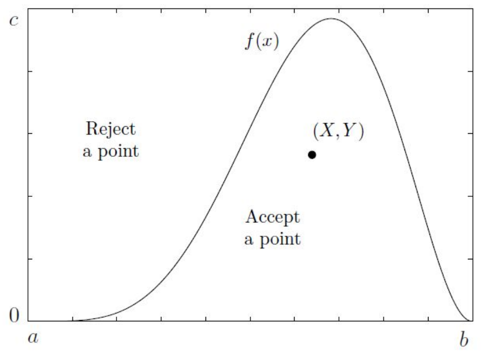
Rejection method

Theorem 3 Let a pair (X, Y) have Uniform distribution over the region A = {(x, y) | 0 ≤ y ≤ f(x)} for some density function f. Then f is the density of X.



Rejection method

It remains to generate Uniform points in the region A. For this purpose, we select a **bounding box** around the graph of f(x), generate random points in this rectangle, and reject all the points not belonging to A. The remaining points are Uniformly distributed in A.



Rejection method

Algorithm 5.4 (Rejection method)

1. Find such numbers a, b, and c that $0 \le f(x) \le c$ for $a \le x \le b$. The bounding box stretches along the x-axis from a to b and along the y-axis from 0 to c.

2. Obtain Standard Uniform random variables U and V from a random number generator or a table of random numbers.

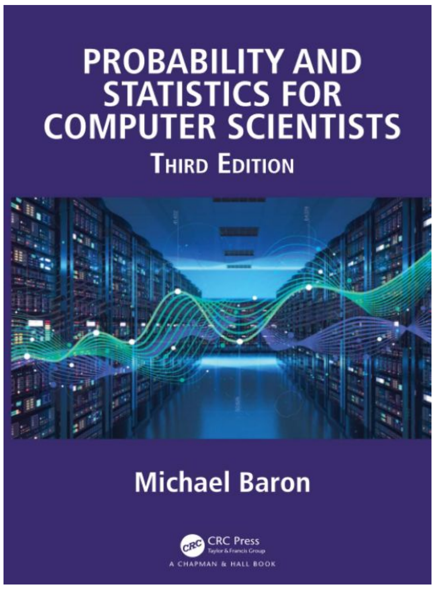
3. Define $X = a + (b - a)U$ and $Y = cV$. Then X has Uniform(a, b) distribution, Y is Uniform(0, c), and the point (X, Y) is Uniformly distributed in the bounding box.

4. If $Y > f(X)$, reject the point and return to step 2. If $Y \le f(X)$, then X is the desired random variable having the density f(x).

References

WIKIPEDIA

Google images


PROBABILITY AND STATISTICS FOR COMPUTER SCIENTISTS

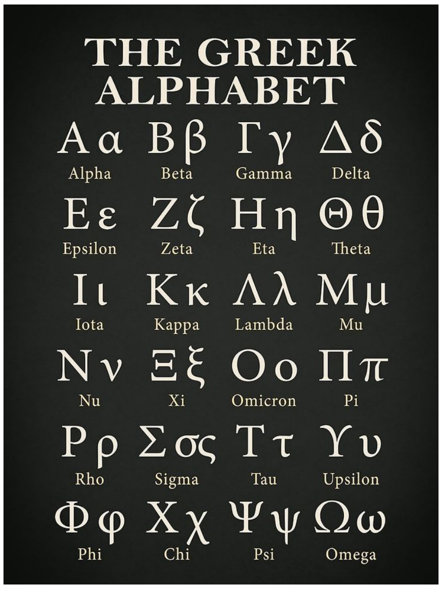
THIRD EDITION

Michael Baron

CRC Press, Taylor & Francis Group

A CHAPMAN & HALL BOOK

Appendix


THE GREEK ALPHABET

Α α	Β β	Γ γ	Δ δ
Alpha	Beta	Gamma	Delta
Ε ε	Ζ ζ	Η η	Θ θ
Epsilon	Zeta	Eta	Theta
Ι ι	Κ κ	Λ λ	Μ μ
Iota	Kappa	Lambda	Mu
Ν ν	Ξ ξ	Ο ο	Π π
Nu	Xi	Omicron	Pi
Ρ ρ	Σ σ ς	Τ τ	Υ υ
Rho	Sigma	Tau	Upsilon
Φ φ	Χ χ	Ψ ψ	Ω ω
Phi	Chi	Psi	Omega

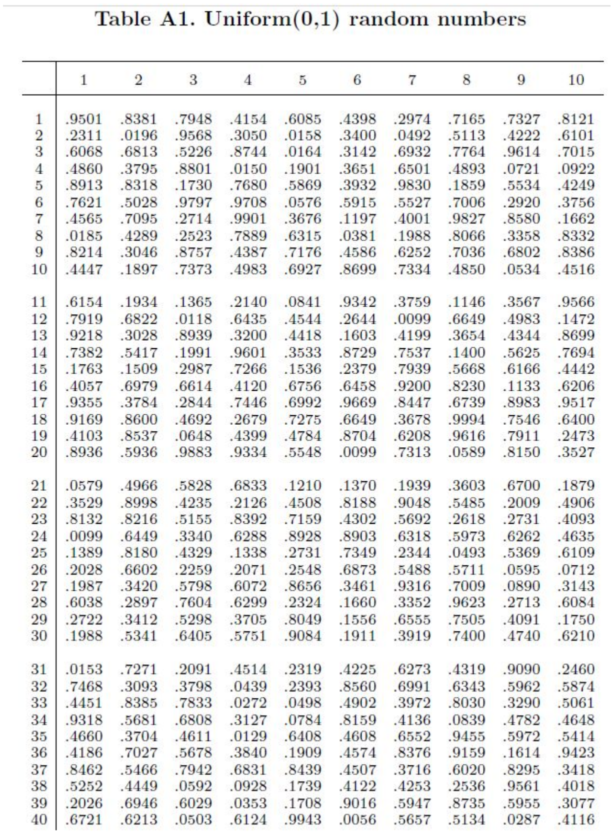


Table A1. Uniform(0,1) random numbers

	1	2	3	4	5	6	7	8	9	10
1	.9501	.8381	.7948	.4154	.6085	.4398	.2974	.7165	.7327	.8121
2	.2311	.0196	.9568	.3050	.0158	.3400	.0492	.5113	.4222	.6101
3	.6068	.6813	.5226	.8744	.0164	.3142	.6932	.7764	.9614	.7015
4	.4860	.3795	.8801	.0150	.1901	.3651	.6501	.4893	.0721	.0922
5	.8913	.8318	.1730	.7680	.5869	.3932	.9830	.1859	.5534	.4249
6	.7621	.5028	.9797	.9708	.0576	.5915	.5527	.7006	.2920	.3756
7	.4565	.7095	.2714	.9901	.3676	.1197	.4001	.9827	.8580	.1662
8	.0185	.4289	.2523	.7889	.6315	.0381	.1988	.8066	.3358	.8332
9	.8214	.3046	.8757	.4387	.7176	.4586	.6252	.7036	.6802	.8386
10	.4447	.1897	.7373	.4983	.6927	.8699	.7334	.4850	.0534	.4516
11	.6154	.1934	.1365	.2140	.0841	.9342	.3759	.1146	.3567	.9566
12	.7919	.6822	.0118	.6435	.4544	.2644	.0099	.6649	.4983	.1472
13	.9218	.3028	.8939	.3200	.4418	.1603	.4199	.3654	.4344	.8699
14	.7382	.5417	.1991	.9601	.3533	.8729	.7537	.1400	.5625	.7694
15	.1763	.1509	.2987	.7266	.1536	.2379	.7939	.5668	.6166	.4442
16	.4057	.6979	.6614	.4120	.6756	.6458	.9200	.8230	.1133	.6206
17	.9355	.3784	.2844	.7446	.6992	.9669	.8447	.6739	.8983	.9517
18	.9169	.8600	.4692	.2679	.7275	.6649	.3678	.9994	.7546	.6400
19	.4103	.8537	.0648	.4399	.4784	.8704	.6208	.9616	.7911	.2473
20	.8936	.5936	.9883	.9334	.5548	.0099	.7313	.0589	.8150	.3527
21	.0579	.4966	.5828	.6833	.1210	.1370	.1939	.3603	.6700	.1879
22	.3529	.8998	.4235	.2126	.4508	.8188	.9048	.5485	.2009	.4906
23	.8132	.8216	.5155	.8392	.7159	.4302	.5692	.2618	.2731	.4093
24	.0099	.6449	.3340	.6288	.8928	.8903	.6318	.5973	.6262	.4635
25	.1389	.8180	.4329	.1338	.2731	.7349	.2344	.0493	.5369	.6109
26	.2028	.6602	.2259	.2071	.2548	.6873	.5488	.5711	.0595	.0712
27	.1987	.3420	.5798	.6072	.8656	.3461	.9316	.7009	.0890	.3143
28	.6038	.2897	.7604	.6299	.2324	.1660	.3352	.9623	.2713	.6084
29	.2722	.3412	.5298	.3705	.8049	.1556	.6555	.7505	.4091	.1750
30	.1988	.5341	.6405	.5751	.9084	.1911	.3919	.7400	.4740	.6210
31	.0153	.7271	.2091	.4514	.2319	.4225	.6273	.4319	.9090	.2460
32	.7468	.3093	.3798	.0439	.2393	.8560	.6991	.6343	.5962	.5874
33	.4451	.8385	.7833	.0272	.0498	.4902	.3972	.8030	.3290	.5061
34	.9318	.5681	.6808	.3127	.0784	.8159	.4136	.0839	.4782	.4648
35	.4660	.3704	.4611	.0129	.6408	.4608	.6552	.9455	.5972	.5414
36	.4186	.7027	.5678	.3840	.1909	.4574	.8376	.9159	.1614	.9423
37	.8462	.5466	.7942	.6831	.8439	.4507	.3716	.6020	.8295	.3418
38	.5252	.4449	.0592	.0928	.1739	.4122	.4253	.2536	.9561	.4018
39	.2026	.6946	.6029	.0353	.1708	.9016	.5947	.8735	.5955	.3077
40	.6721	.6213	.0503	.6124	.9943	.0056	.5657	.5134	.0287	.4116