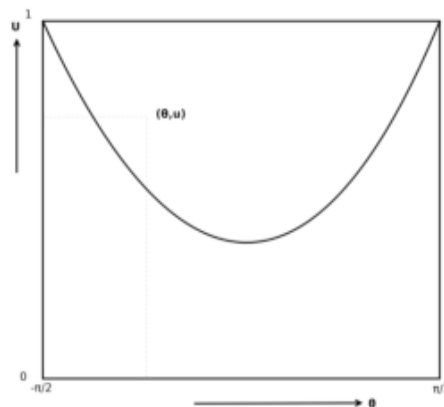


```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```

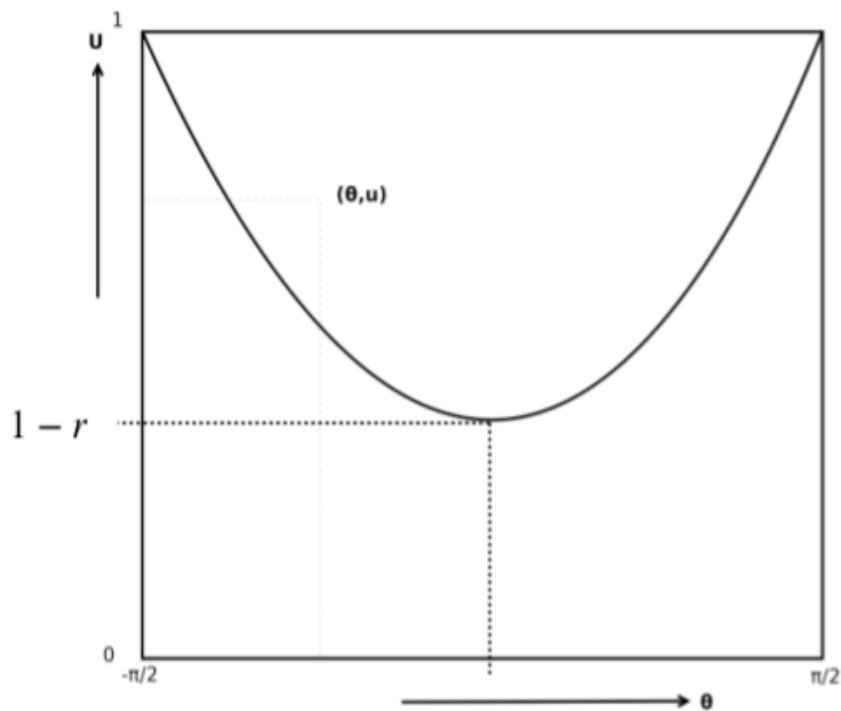
<https://xkcd.com/221/>

# Montecarlo simulation



- Randomly sample:  $(\theta_1, u_1), (\theta_2, u_2), \dots, (\theta_n, u_n)$
- If some  $(\theta_i, u_i)$  falls above the curve (i.e.,  $u_i > 1 - r \cos \theta$ ) call it a **hit**, otherwise a **miss**.
- Probability of the needle crossing can be approximated as number of **hits** over number of samples ( $n$ )

```
def buffon(number_of_samples, r):  
    hits = 0  
    for _ in range(number_of_samples):  
        tetha = random.uniform(-math.pi/2, math.pi/2)  
        u = random.rand()  
        if u > (1-r*np.cos(tetha)):  
            hits+=1  
    return hits/number_of_samples
```



$$\text{Probability of not crossing} = \frac{\text{Area under curve}}{\text{Total area}}$$

$$\text{Probability of crossing} = \frac{\text{Area above curve}}{\text{Total area}} = \frac{\text{Total area} - \text{Area under curve}}{\text{Total area}} = \frac{2r}{\pi}$$

$$\text{Total area} = \pi$$

$$\text{Area under curve} = \int_{-\pi/2}^{\pi/2} (1 - r \cos \theta) d\theta$$

$$\text{Area under curve} = \left[ \theta - r \sin \theta \right]_{-\pi/2}^{\pi/2}$$

$$\text{Area under curve} = \pi - 2r$$

# Workshop 14

## Sampling

### **FIT 3139** Computational Modelling and Simulation




How do we generate  
random numbers?

<https://www.random.org/>

# Random Number Generator (**RNG**)

Historically...

- Physical generators
- Table look-up generators
- Software 
  - Randomness
  - Efficiency
  - Reproducibility

## **Pseudo-random numbers:**

Deterministically generated sequence of independent numbers that appears to be ***uniformly distributed in the interval (0, 1)***.

# Multiplicative congruential method

$$\begin{aligned} x_0 &\longrightarrow \text{seed} \\ x_n &= ax_{n-1} \bmod m \\ \frac{x_n}{m} &\longrightarrow \text{approximates the value of a } (0, 1) \text{ uniform random variable} \end{aligned}$$

- This sequence is **periodic**, and will repeat itself at some point.
- It is desirable to choose  $a$  and  $m$ , in a way that leads to a **large period** for *any*  $x_0$ .
- In general,  $m$  should be a large prime.

For a 32-bit word machine, the choice  $m = 2^{31} - 1$ ,  $a = 7^5$  exhibits desirable statistical properties. **A number of statistical tests are available to evaluate RNG.**



# Linear congruential method


$$\begin{aligned} x_0 &\longrightarrow \text{seed} \\ x_n &= (ax_{n-1} + c) \bmod m \\ \frac{x_n}{m} &\longrightarrow \text{approximates the value of a } (0, 1) \text{ uniform random variable} \end{aligned}$$

- Where  $0 < a < m$ , is called the **multiplier**.
- And  $0 \leq c < m$ , is called the **increment**.
- *Hull–Dobell Theorem* provides sufficient conditions on these constants to guarantee **largest possible period**.
- Very popular: simple theory and amenable to efficient implementations.

$(m, a, c)$

Source	modulus $m$	multiplier $a$	increment $c$
<i>Numerical Recipes</i>	$2^{32}$	1664525	1013904223
Borland C/C++	$2^{32}$	22695477	1
glibc (used by GCC) <sup>[9]</sup>	$2^{31}$	1103515245	12345
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++ <sup>[10]</sup> C99, C11: Suggestion in the ISO/IEC 9899 <sup>[11]</sup>	$2^{31}$	1103515245	12345
Borland Delphi, Virtual Pascal	$2^{32}$	134775813	1
Turbo Pascal	$2^{32}$	134775813 (0x8088405 <sub>16</sub> )	1
Microsoft Visual/Quick C/C++	$2^{32}$	214013 (343FD <sub>16</sub> )	2531011 (269EC3 <sub>16</sub> )
Microsoft Visual Basic (6 and earlier) <sup>[12]</sup>	$2^{24}$	1140671485 (43FD43FD <sub>16</sub> )	12820163 (C39EC3 <sub>16</sub> )
RtlUniform from Native API <sup>[13]</sup>	$2^{31} - 1$	2147483629 (7FFFFFFD <sub>16</sub> )	2147483587 (7FFFFFFC3 <sub>16</sub> )
Apple CarbonLib, C++11's <code>minstd_rand0</code> <sup>[14]</sup>	$2^{31} - 1$	16807	0
C++11's <code>minstd_rand</code> <sup>[14]</sup>	$2^{31} - 1$	48271	0
MMIX by Donald Knuth	$2^{64}$	6364136223846793005	1442695040888963407
Newlib, Musl	$2^{64}$	6364136223846793005	1
VMS's <code>MTH\$RANDOM</code> , <sup>[15]</sup> old versions of glibc	$2^{32}$	69069 (10DCD <sub>16</sub> )	1
Java's <code>java.util.Random</code> , POSIX <code>[ln]rand48</code> , glibc <code>[ln]rand48[_r]</code>	$2^{48}$	25214903917 (5DEECE66D <sub>16</sub> )	11
<code>random0</code> <sup>[16][17][18][19][20]</sup> If $X_{i-1}$ is even then $X_i$ will be odd, and vice versa—the lowest bit oscillates at each step.	$134456 = 2^{27}5$	8121	28411
POSIX <sup>[21]</sup> <code>[jm]rand48</code> , glibc <code>[mj]rand48[_r]</code>	$2^{48}$	25214903917 (5DEECE66D <sub>16</sub> )	11
POSIX <code>[de]rand48</code> , glibc <code>[de]rand48[_r]</code>	$2^{48}$	25214903917 (5DEECE66D <sub>16</sub> )	11
cc65 <sup>[22]</sup>	$2^{23}$	65793 (10101 <sub>16</sub> )	4282663 (415927 <sub>16</sub> )
cc65	$2^{32}$	16843009 (1010101 <sub>16</sub> )	826366247 (31415927 <sub>16</sub> )

[https://en.wikipedia.org/wiki/Linear\\_congruential\\_generator](https://en.wikipedia.org/wiki/Linear_congruential_generator)



Almost all module functions depend on the basic function `random()`, which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of  $2^{19937}-1$ . The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the `random.Random` class. You can instantiate your own instances of `Random` to get generators that don't share state.

Class `Random` can also be subclassed if devising: in that case, override the `__init__` method. Optionally, a new generator can supply produce selections over an arbitrarily large range.

The `random` module also provides the `os.urandom()` to generate random numbers.

**Warning:** The pseudo-random generator is not suitable for cryptographic uses. For security or cryptographic uses, see the `secrets` module.

**See also:** M. Matsumoto and T. Nishimura, "A highly efficient equidistributed uniform pseudorandom number generator," *Computer Simulation* Vol. 8, No. 1, January 1989.

**Complementary-Multiply-with-Carry random number generator** with a long period and comparatively simple implementation.

## Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator

Makoto Matsumoto<sup>1</sup> and Takuji Nishimura<sup>2</sup>

<sup>1</sup>Keio University/Max-Planck-Institut für Mathematik

<sup>2</sup>Keio University

In this paper, a new algorithm named *Mersenne Twister* (MT) for generating uniform pseudorandom numbers is proposed. For a particular choice of parameters, the algorithm provides a super astronomical period of  $2^{19937} - 1$  and 623-dimensional equidistribution up to 32 bits accuracy, while consuming a working area of only 624 words. This is a new variant of the previously proposed generators TGFSR, modified so as to admit a Mersenne-prime period. The characteristic polynomial has many terms. The distribution up to  $v$  bits accuracy for  $1 \leq v \leq 32$  is also shown to be good.

Also, an algorithm to check the primitivity of the characteristic polynomial of MT with computational complexity  $O(p^2)$  is given, where  $p$  is the degree of the polynomial.

We implemented this generator as a portable C-code. It passed several stringent statistical tests, including diehard. The speed is comparable to other modern generators.

These merits are a consequence of the efficient algorithms unique to polynomial calculations over the two-element field.

1989

<https://docs.python.org/3/library/random.html>

**See also:** <http://www.pcg-random.org/>

# rand()

- Every call of this function is a *realisation* or a *sample* from a random variable  $X$ .
- $X$  is a continuous random variable.
- $X$  is uniformly distributed in  $(0, 1)$ ; denoted  $X \sim U(0,1)$
- How to sample continuous random variables, from **arbitrary** distributions?

- How to sample continuous random variables, from **arbitrary** distributions?

We say a random variable is **continuous**, if the set of possible values is an interval.

$$P\{X \in C\} = \int_C f(x) dx$$

$f(x)$   $\longrightarrow$  Probability **density** function  
[for a continuous RV]

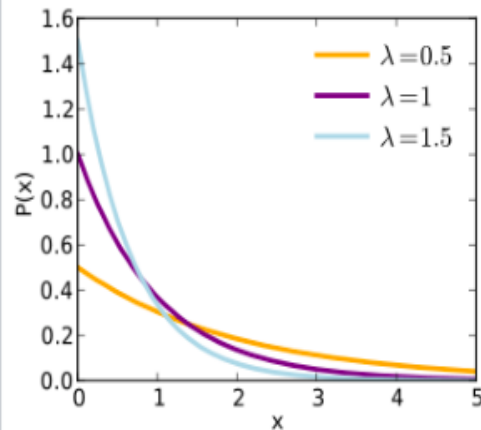
$$F(\cdot) \longrightarrow \text{Cumulative } \mathbf{distribution}$$

$$F(a) = P\{X \in (-\infty, a]\} = \int_{-\infty}^a f(x) dx$$

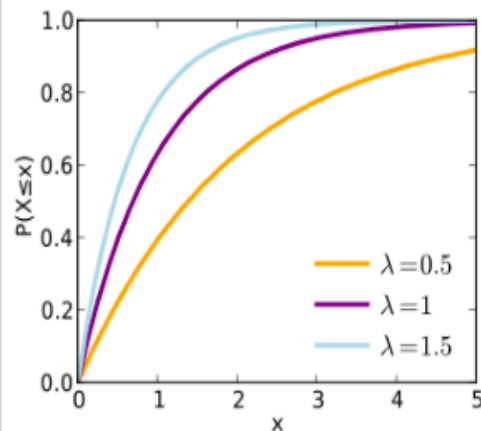
(the density is the derivative of the cumulative distribution)

## Exponential

Probability density function



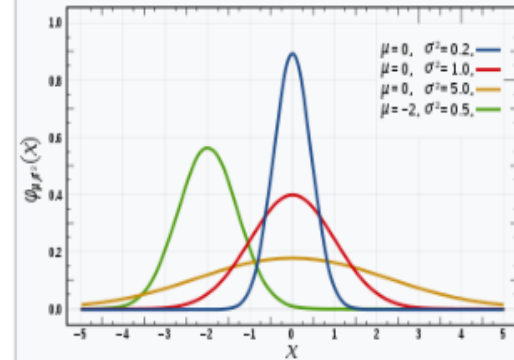
Cumulative distribution function



<b>Parameters</b>	$\lambda > 0$ rate, or inverse <b>scale</b>
<b>Support</b>	$x \in [0, \infty)$
<b>PDF</b>	$\lambda e^{-\lambda x}$
<b>CDF</b>	$1 - e^{-\lambda x}$

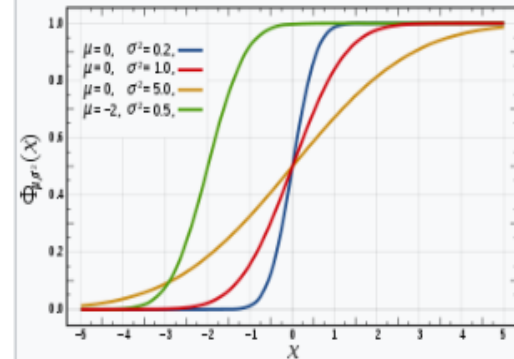
## Normal Distribution

Probability density function



The red curve is the standard normal distribution

Cumulative distribution function



<b>Notation</b>	$\mathcal{N}(\mu, \sigma^2)$
<b>Parameters</b>	$\mu \in \mathbb{R}$ = mean ( <b>location</b> ) $\sigma^2 > 0$ = variance (squared <b>scale</b> )
<b>Support</b>	$x \in \mathbb{R}$
<b>PDF</b>	$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
<b>CDF</b>	$\frac{1}{2} \left[ 1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right) \right]$

## Proposition

Let  $U$  be an uniform  $(0,1)$  random variable . For any continuous distribution  $F$ , the random variable  $X$  defined by:

$$X = F^{-1}(U)$$

has distribution  $F$

Let  $F_x$  be the distribution of  $X$

$$\begin{aligned} F_x(x) &= P\{X \leq x\} \\ &= P\{F^{-1}(U) \leq x\} \quad (\text{from the proposition}) \end{aligned}$$

$F$  must be monotonously increasing because it is a distribution.

$$a \leq b \rightarrow F(a) \leq F(b)$$

$$\begin{aligned} F_x(x) &= P\{F(F^{-1}(U)) \leq F(x)\} && (\text{since } F \text{ is mon. inc}) \\ &= P\{U \leq F(x)\} && (\text{since } F^{-1}F(U) = U) \\ &= F(x) && (\text{since } U \text{ is uniform } (0,1)) \end{aligned}$$

# Inverse transform sampling

To sample from a r.v with **invertible** distribution  $F$

1. Sample a uniform random number  $u \in [0,1)$
2. Compute  $x = F^{-1}(u)$
3. The value  $x \sim F(\cdot)$



# Example: simulating arrival times...

The **Poisson distribution** describes number of events occurring in an interval of time. Rate:  $\lambda$

The **exponential distribution** describes time between independent events that occur at a constant rate.

Density of exponential distribution:

$$f(x|\lambda) = \lambda e^{-\lambda x} \quad x \in [0, \infty]$$

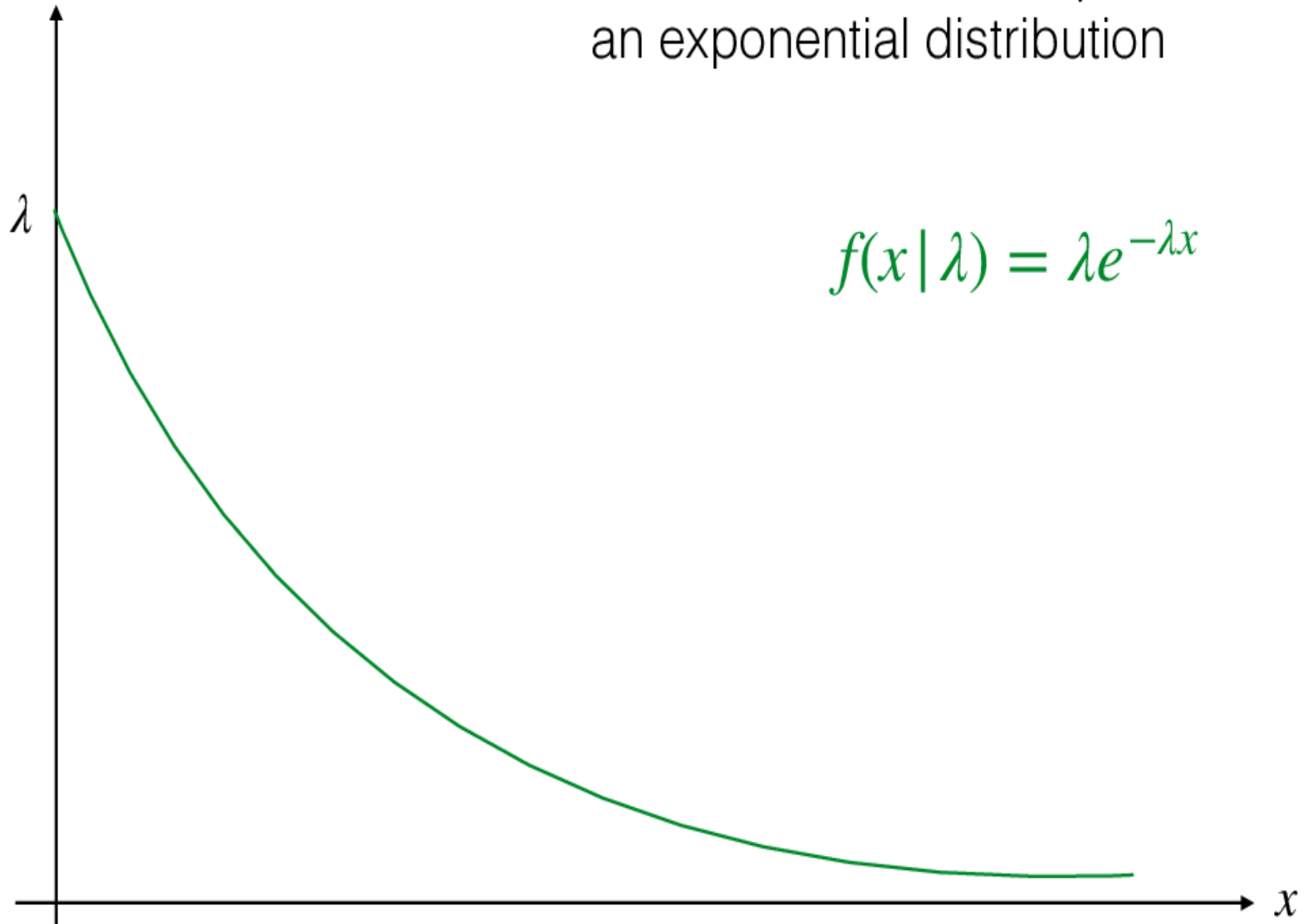
$$F(x|\lambda) = \int_0^x f(t|\lambda) dt = 1 - e^{-\lambda x}$$

Using inverse transform sampling...

$$x = -\frac{1}{\lambda} \log(1 - u)$$

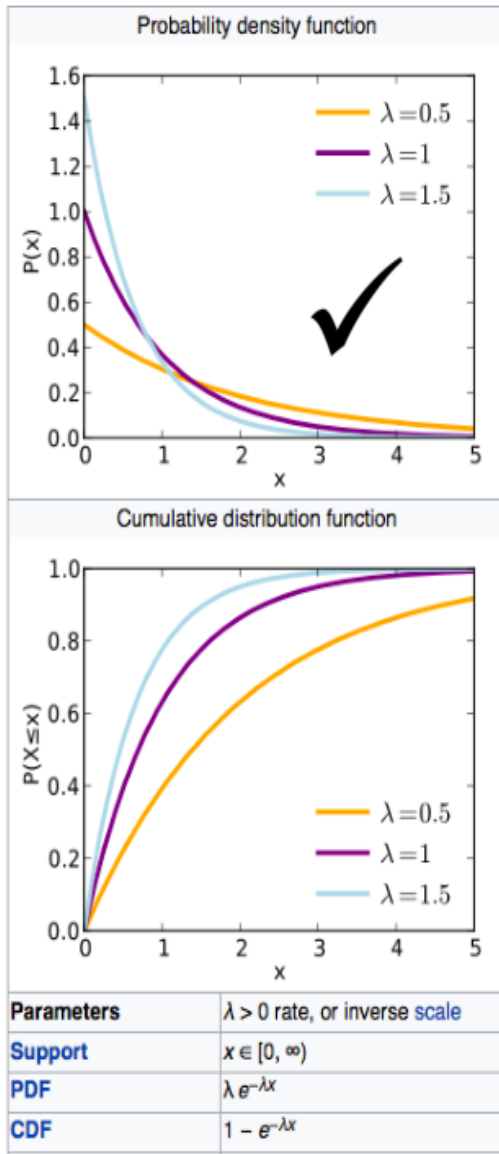
```
def sample(lambda_):  
    return -1*np.log(rand())/lambda_
```

We know how to sample  
an exponential distribution



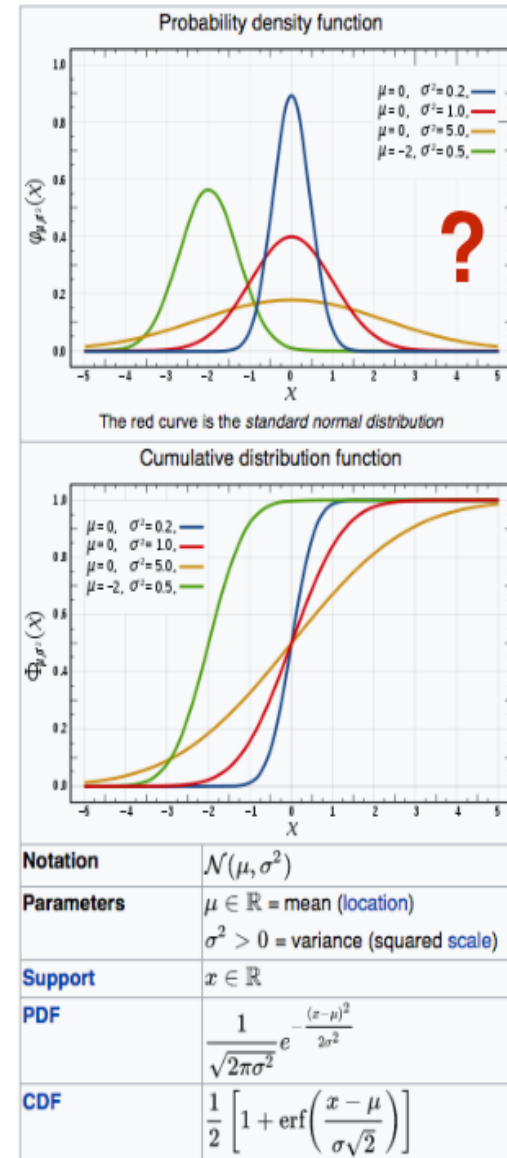
What if the **distribution**  
cannot be inverted?

## Exponential

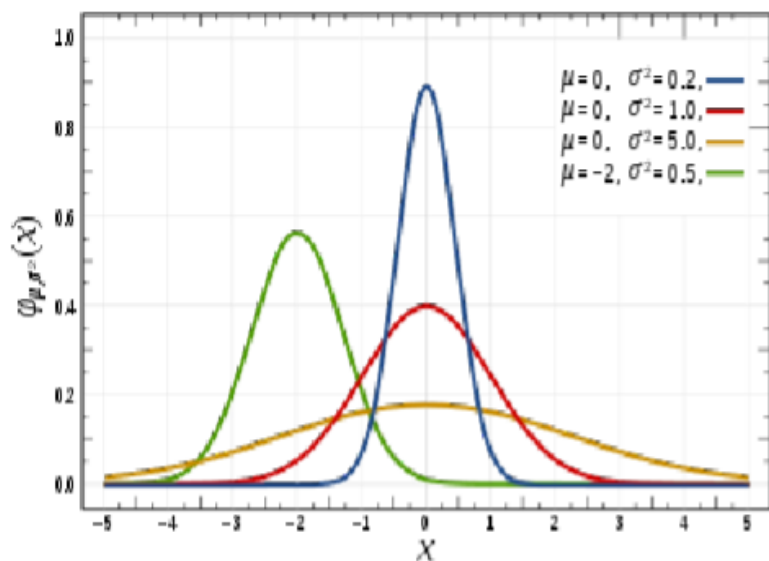


[https://en.wikipedia.org/wiki/Exponential\\_distribution](https://en.wikipedia.org/wiki/Exponential_distribution)

## Normal Distribution



[https://en.wikipedia.org/wiki/Normal\\_distribution](https://en.wikipedia.org/wiki/Normal_distribution)



$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

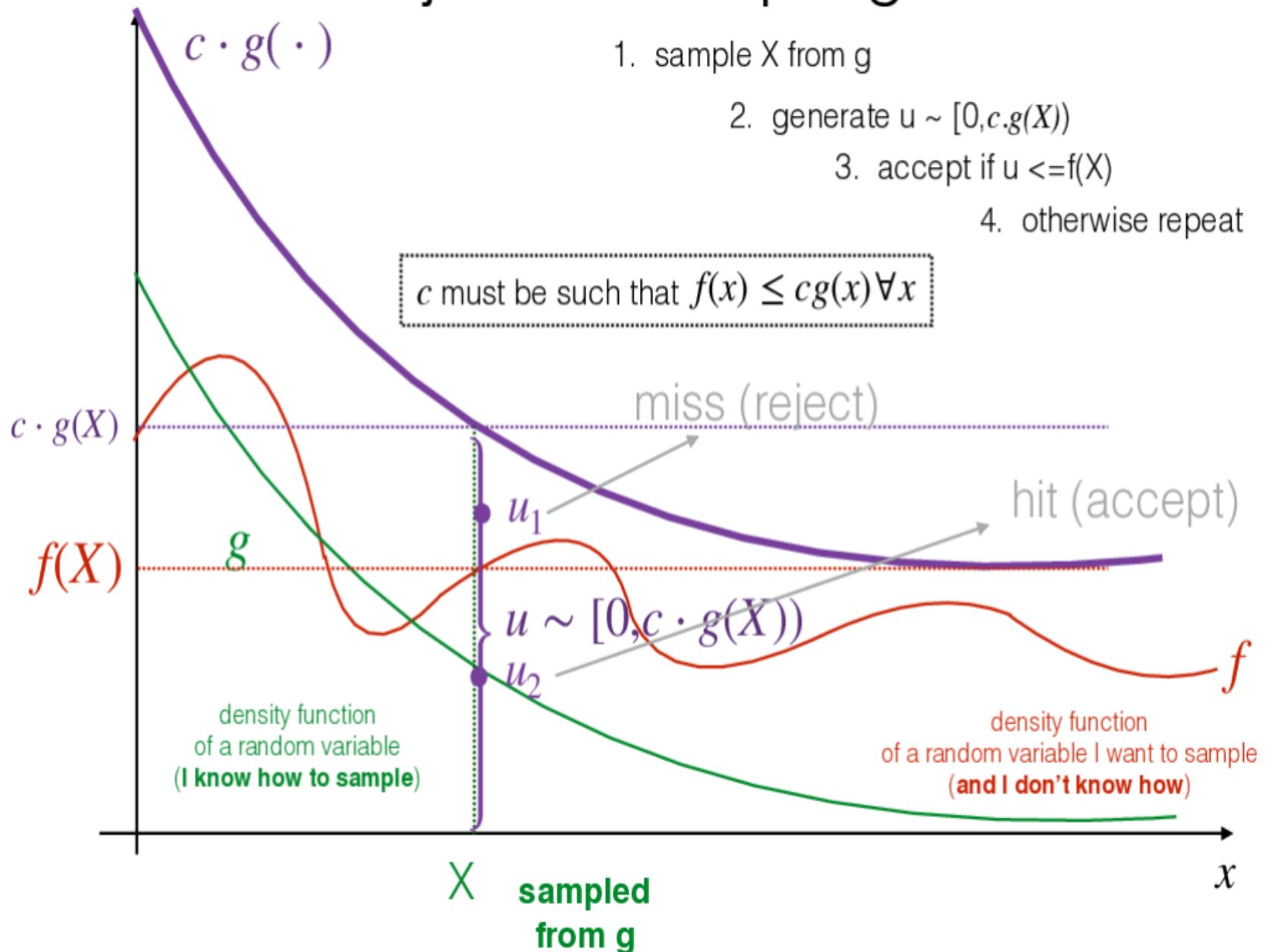
$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$$

I cannot express the **distribution** in terms of elementary functions, let alone find the inverse...

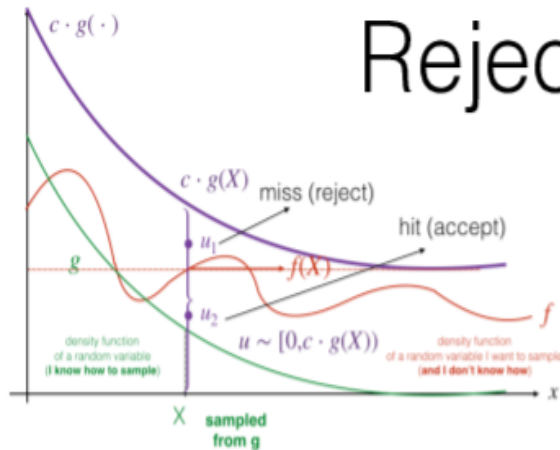
# Rejection sampling

1. sample  $X$  from  $g$
2. generate  $u \sim [0, c \cdot g(X)]$
3. accept if  $u \leq f(X)$
4. otherwise repeat

$c$  must be such that  $f(x) \leq cg(x) \forall x$

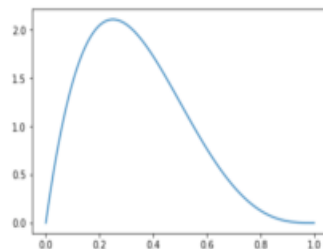


# Rejection sampling



To sample from a r.v with **density**  $f$

1. Choose a **density**  $g$  that you know how to sample.
2. Find  $c$ , such that  $\frac{f(y)}{g(y)} \leq c \forall y$
3. Generate  $X$ , having density  $g$
4. Generate a random number  $u \in [0,1)$
5. If  $u \leq \frac{f(X)}{cg(X)}$ , then  $X$  is your sample.
6. Otherwise, repeat from 3.



Example.

$$\text{In}[1] := \int_0^1 20 x (1-x)^3 dx$$

$$\text{Out}[1] = 1$$

$$f(x) = 20x(1-x)^3 \longrightarrow \text{Target density} \quad 0 < x < 1$$

1. Choose a **density**  $g$  that you know how to sample.

$$u \sim [0,1) \quad 0 < x < 1 \quad g(x) = 1$$

2. Find  $c$ , such that  $\frac{f(y)}{g(y)} \leq c \forall y$

$$c = \max \frac{f(x)}{g(x)} \quad f'(x) = 20[(1-x)^3 - 3x(1-x^2)] = 0 \quad x = \frac{1}{4}$$

$$c = 20 \left( \frac{1}{4} \right) \left( \frac{3}{4} \right)^3 = \frac{135}{64} \quad \frac{f(x)}{cg(x)} = \frac{256}{27} x(1-x)^3$$



1. Choose a **density**  $g$  that you know how to sample.

$$0 < x < 1 \quad g(x) = 1 \quad u \sim [0,1)$$

2. Find  $c$ , such that  $\frac{f(y)}{g(y)} \leq c \forall y$

$$\frac{f(x)}{cg(x)} = \frac{256}{27}x(1-x)^3$$

3. Generate  $X$ , having density  $g$

$$U_1 \sim U[0,1)$$

4. Generate a random number  $u \in [0,1)$

$$U_2 \sim U[0,1)$$

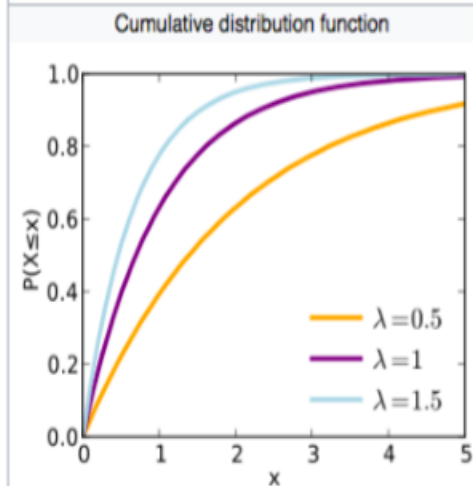
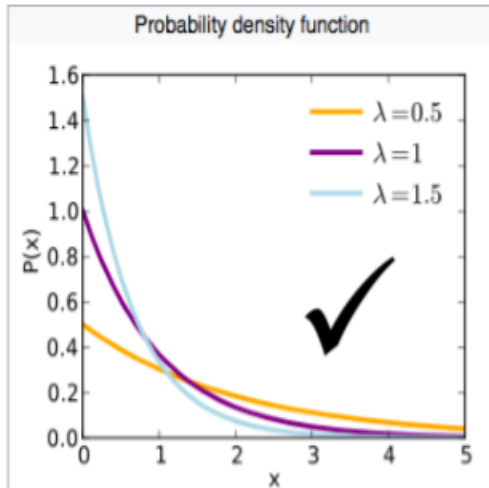
5. If  $u \leq \frac{f(X)}{cg(X)}$ , then  $X$  is your sample.

$$U_2 \leq \frac{256}{27}U_1(1-U_1)^3 \quad \text{yes, return } U_1$$

**no**, repeat from 3

6. Otherwise, repeat from 3.

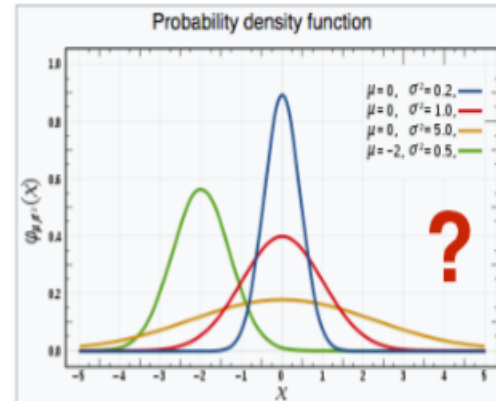
## Exponential



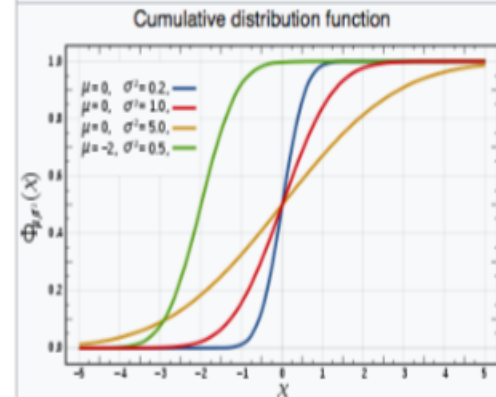
<b>Parameters</b>	$\lambda > 0$ rate, or inverse <b>scale</b>
<b>Support</b>	$x \in [0, \infty)$
<b>PDF</b>	$\lambda e^{-\lambda x}$
<b>CDF</b>	$1 - e^{-\lambda x}$

[https://en.wikipedia.org/wiki/Exponential\\_distribution](https://en.wikipedia.org/wiki/Exponential_distribution)

## Normal Distribution



The red curve is the standard normal distribution



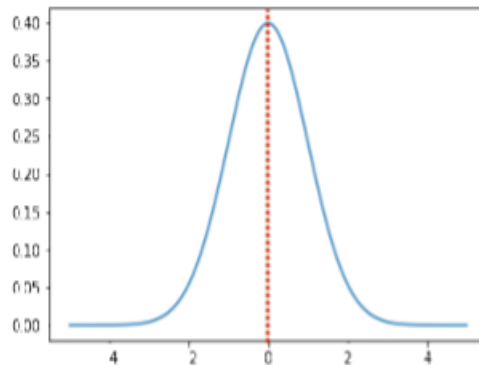
<b>Notation</b>	$\mathcal{N}(\mu, \sigma^2)$
<b>Parameters</b>	$\mu \in \mathbb{R}$ = mean ( <b>location</b> ) $\sigma^2 > 0$ = variance (squared <b>scale</b> )
<b>Support</b>	$x \in \mathbb{R}$
<b>PDF</b>	$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
<b>CDF</b>	$\frac{1}{2} \left[ 1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right) \right]$

[https://en.wikipedia.org/wiki/Normal\\_distribution](https://en.wikipedia.org/wiki/Normal_distribution)

# Generate a standard normal distribution $Z$

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

$$-\infty < x < \infty$$

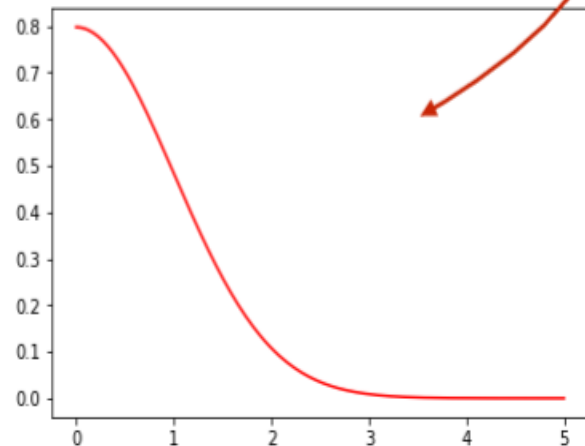


Let **g** be an exponential distribution (I know how to sample it, right?)

$$0 < x < \infty \quad g(x) = e^{-x}$$

$$f(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

This is the  
distribution of  
the **absolute value of Z**



Target:  $f(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

1. Choose a **density**  $g$  that you know how to sample.

$$g(x) = e^{-x}$$

2. Find  $c$ , such that  $\frac{f(y)}{g(y)} \leq c \forall y$

$$\frac{f(x)}{g(x)} = \frac{\frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}}{e^{-x}} = \sqrt{\frac{2}{\pi}} e^{\left(x - \frac{x^2}{2}\right)} \quad \begin{array}{l} \text{maximised} \\ \text{for } x=1 \end{array} \quad c = \sqrt{\frac{2}{\pi}} e$$

$$\frac{f(x)}{cg(x)} = \exp\left(x - \frac{x^2}{2} - \frac{1}{2}\right) = \exp\left(\frac{-(x-1)^2}{2}\right)$$

3. Generate  $X$ , having density  $g$

$$X = -\log(\text{rand}())$$

2. Find  $c$ , such that  $\frac{f(y)}{g(y)} \leq c \forall y$

$$\frac{f(x)}{cg(x)} = \exp\left(x - \frac{x^2}{2} - \frac{1}{2}\right) = \exp\left(\frac{-(x-1)^2}{2}\right)$$

3. Generate  $X$ , having density  $g$

$$X = -\log(\text{rand}())$$

4. Generate a random number  $u \in [0,1)$

`rand()`

5. If  $u \leq \frac{f(X)}{cg(X)}$ , then  $X$  is your sample.

6. Otherwise, repeat from 3.

Remember: this samples from the **absolute value**, to recover standard normal distribution multiply by -1 with probability 0.5



What about arbitrary normal  
distributions?

**Not examinable**

## Box-Mueller-Gauss method

Given the parameters of a normal distribution,  $\sigma$  and  $\mu$ .

1. Choose a uniform random number  $\theta \in [0, 2\pi)$
2. Choose a uniform random number  $u \in [0, 1)$
3. Compute  $r = \sigma \sqrt{-2 \ln(1 - u)}$

The values  $r \cos(\theta) + \mu$  and  $r \sin(\theta) + \mu$   
can be proven to be normally distributed with mean  $\mu$   
and standard distribution  $\sigma$

(get in touch if you want more info about the proof)

---

# Some Comments on C. S. Wallace's Random Number Generators

RICHARD P. BRENT

*Oxford University Computing Laboratory,  
Wolfson Building, Parks Road,  
Oxford OX1 3QD, UK  
Email: rpb213@rpbrent.co.uk*

*Dedicated to Chris Wallace on the occasion of his 70th birthday.*

---

**We outline some of Chris Wallace's contributions to pseudo-random number generation. In particular, we consider his recent idea for generating normally distributed variates without relying on a source of uniform random numbers, and compare it with more conventional methods for generating normal random numbers. Implementations of Wallace's idea can be very fast (approximately as fast as good uniform generators). We discuss the statistical quality of the output, and mention how certain pitfalls can be avoided.**

*Keywords: Gaussian distribution, maximum-entropy distributions, normal distribution, orthogonal transformations, random number generation, Wallace algorithm*

*Received 22 March 2003; revised 14 June 2003; accepted 12 May 2003*



**Next up:**

Stochastic Dynamical Systems (Gillespie Algorithm)  
Stochastic Processes (Markov Chains)