University of Southampton

SESA6085 – Advanced Aerospace Engineering Management

Lecture 6

2024-2025

Dr David Toal

Overview

- To date we've considered probability theory and discrete and continuous probability distributions
- During these lectures we've seen how to evaluate reliabilities and other quantities analytically
- This is fine if the distribution is known, has a closed form integral and the system is simple
- What if the system is complex with no closed form solution?
- The system is almost like a black-box
  - Inputs go in, outputs come out but what happens in between is a mystery

Overview

- Monte Carlo simulations can be performed in these cases
- Today we will define:
  - The difference between stochastic and deterministic simulations
  - How random numbers are generated
  - How statistically distributed random numbers are generated
  - What a Monte Carlo simulation is and it’s limitations
  - More efficient variants of this type of simulation

Stochastic Simulations

Deterministic & Stochastic Simulations

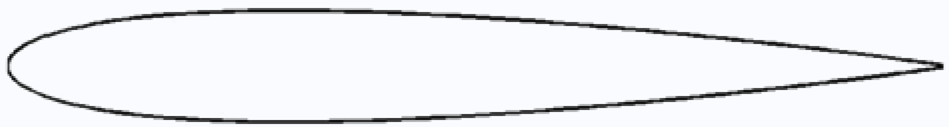
- What do we mean by deterministic or stochastic?
- Deterministic:
  - No randomness in the process
  - Same input will give the same output e.g.

    In Matlab: `2+2 = 4`
- Stochastic:
  - Opposite of deterministic – the process includes a random element based on some probability e.g.

    In Matlab: `2+rand = ?`

An Example

- We want to simulate an aerofoil (NACA 0012)…



- A traditional deterministic simulation of this aerofoil would fix the inputs to the simulation e.g. angle of attack, Reynolds number, Mach number etc.
- From this simulation we would get a series of outputs relating to drag, lift etc.
- Is this realistic??

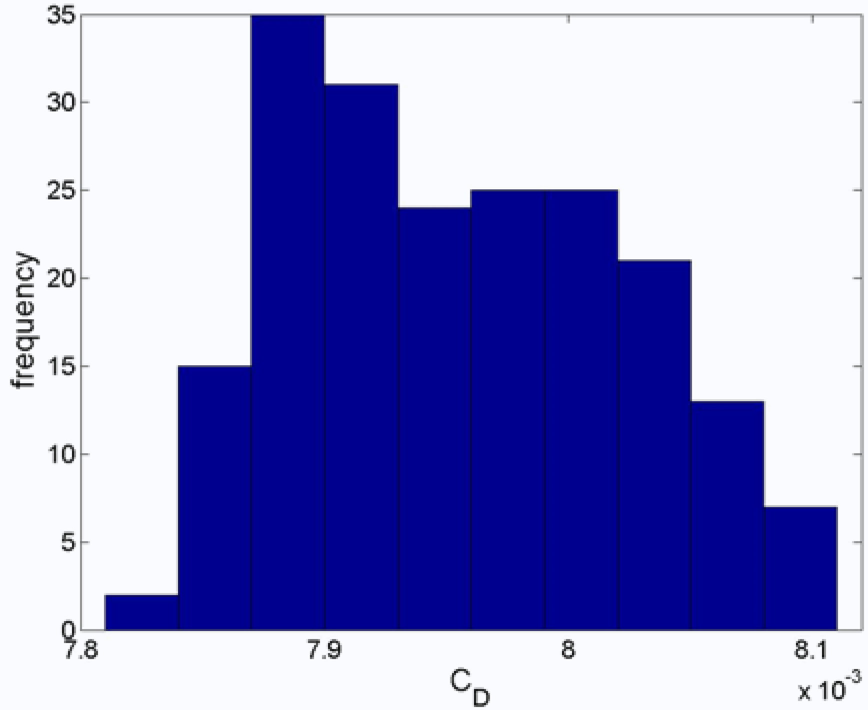
An Example

- Now let's consider a stochastic simulation of the same aerofoil
- In this case the inputs are permitted to vary randomly within a range

$$5.5 \times 10^6 < Re < 6.5 \times 10^6$$

$$1.5° < \alpha < 2.5°$$

$$0.15 < M < 0.25$$



Stochastic Simulations

`2+rand = ?`

- This simple definition of a stochastic simulation highlights a basic requirement of any stochastic process
- We require an ability to generate random numbers
- But not just any random numbers but random numbers generated from a predefined probability distribution
- N.B. For those interested in how random numbers can be generated see the additional slides at the end of this deck

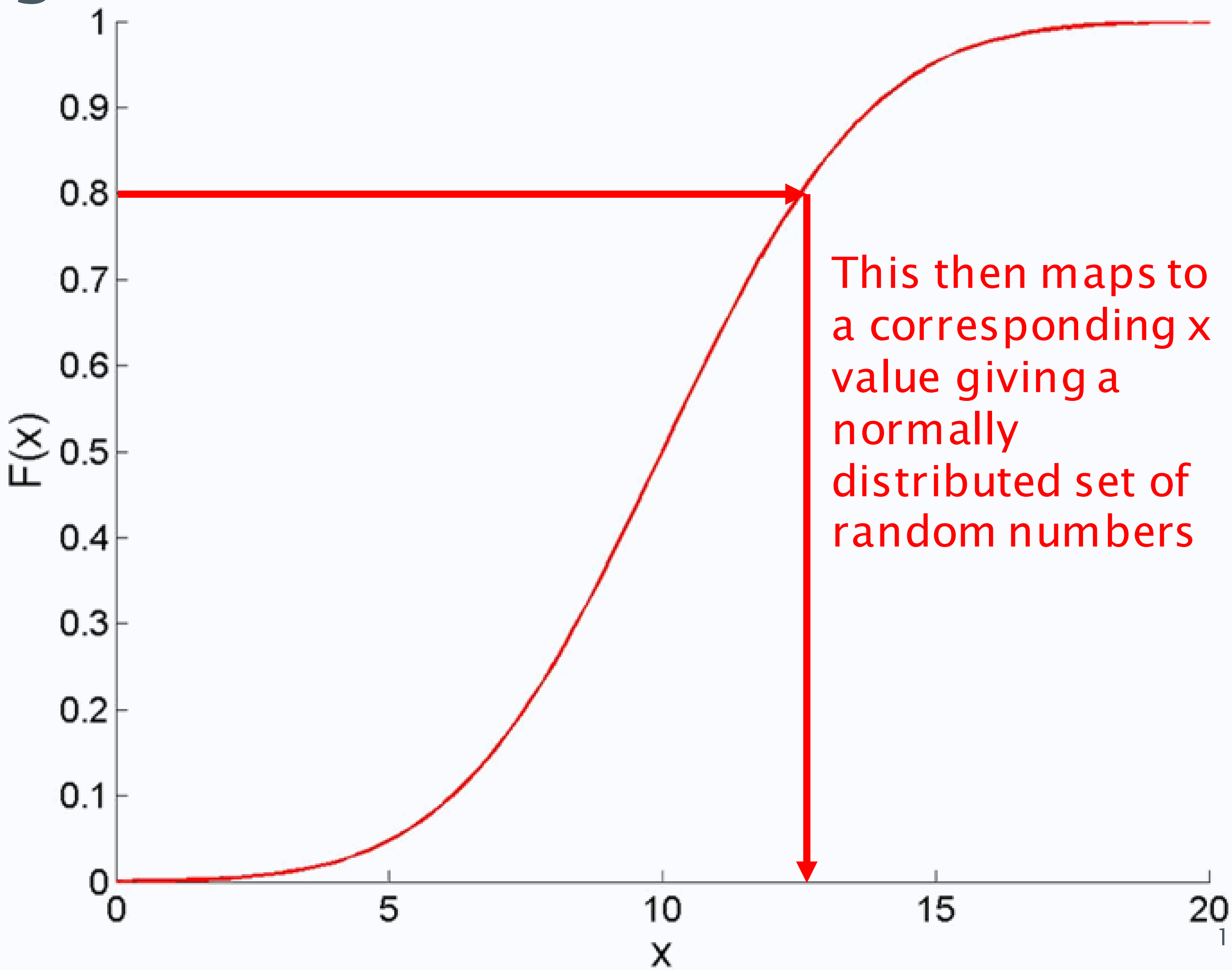
Distributed Random Numbers

Distributed Random Numbers

- How do we generate random numbers based on some form of statistical distribution?
- Let's recall our definition of a CDF:
  - Cumulative distribution function
  - Ranges from 0 to 1
  - Integral of our PDF: $F(t_2) = \int_{-\infty}^{t_2} f(t)dt$
- This gives us two useful things
  1. A link to our original PDF we want our random numbers to be distributed by
  2. Non-repeating outputs from 0 to 1

Generating Distributed Random Numbers

We define F(x) using our uniform random number generator



This then maps to a corresponding x value giving a normally distributed set of random numbers

This is the CDF of a normal distribution with $\mu = 10$ & $\sigma = 3$

Exponential Distribution

- For an exponential distribution we know that:

$$F(t) = e^{-\lambda t} \qquad F(t) \in [0,1]$$

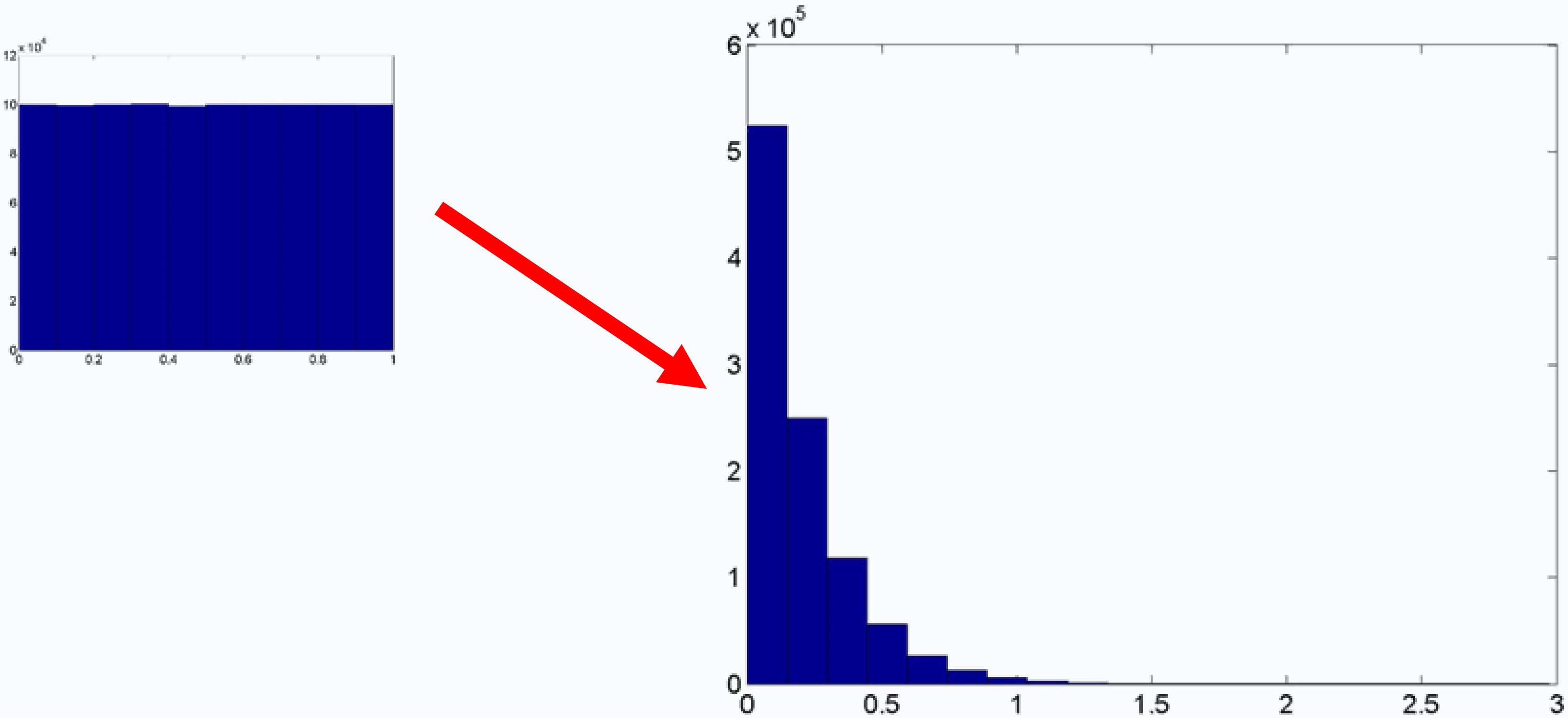
- Which if we rearrange gives us:

$$t = \frac{\ln(F(t))}{-\lambda}$$

- If we then use this inverse function with a given λ and a uniformly distributed random number we will obtain an exponentially distributed random number
- In general, we employ a mapping based on the inverse of the CDF to transform a uniform random number into a distributed one

Exponential Distribution

- Let's consider the case where λ=5
- From a set of uniformly distributed random numbers we get the following exponentially distributed random numbers



N.B. Matlab code to generate these plots is provided in the appendix

Normal Distribution

- The CDF for a normal distribution is:

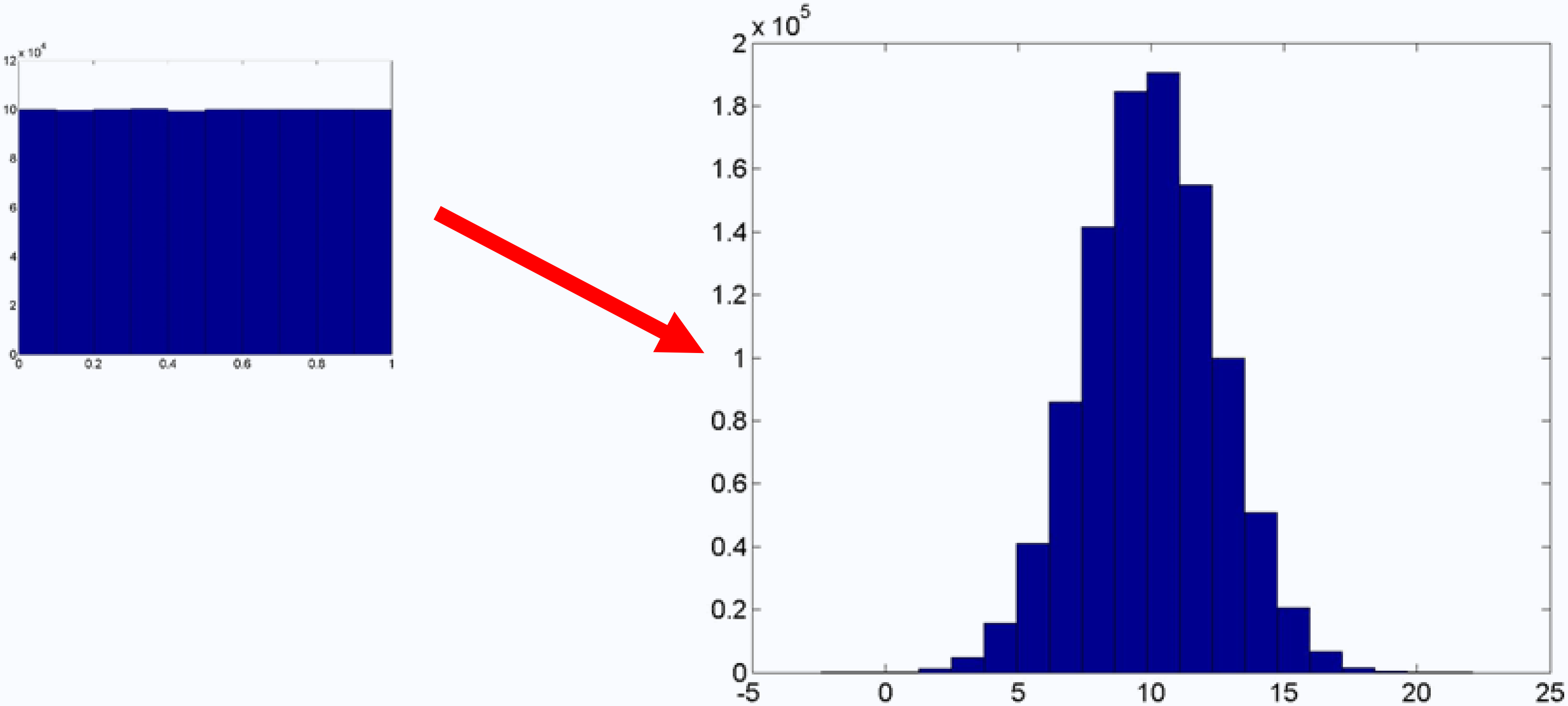
$$F(t) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{t-\mu}{\sigma\sqrt{2}}\right)\right]$$

- erf() denotes the error function which must be computed or approximated
- Taking its inverse we obtain:

$$t = \mu + \sigma\sqrt{2}\,\mathrm{erf}^{-1}(2F(t) - 1)$$

Normal Distribution

- Assuming the case where μ=10 and σ=2.5
- Again, from an initial set of uniformly distributed random numbers we obtain



N.B. Matlab code to generate these plots is provided in the appendix

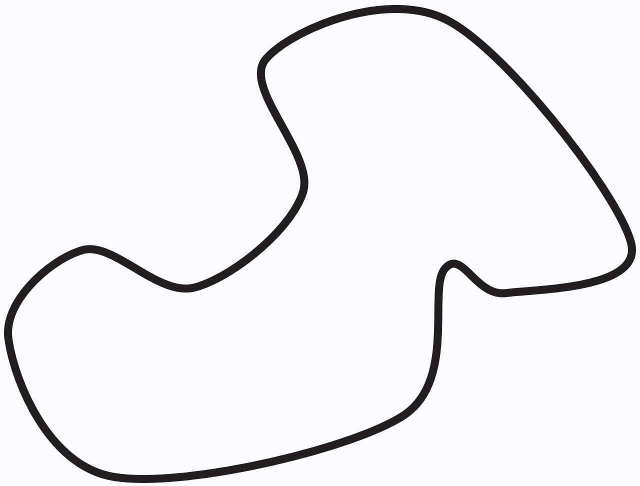
Monte Carlo Simulation

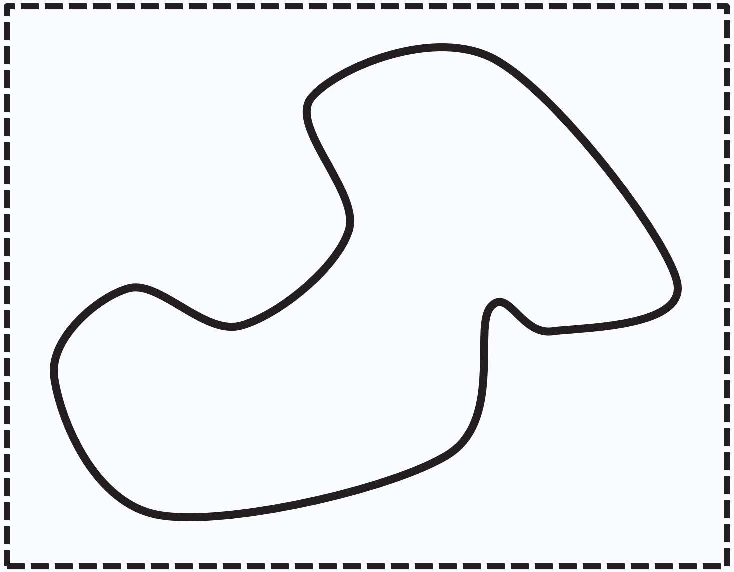
Monte Carlo – An Introduction

- Now that we know how to calculate distributed random numbers let's move on to Monte Carlo simulations
- The Monte Carlo method is a method of random sampling
- Named after the casinos of Monte Carlo. Why?
  - Roulette wheels are a good way of generating random numbers

Monte Carlo – A Simple Example

- Suppose we have an irregular shape like the one below, how can we calculate its area, S?

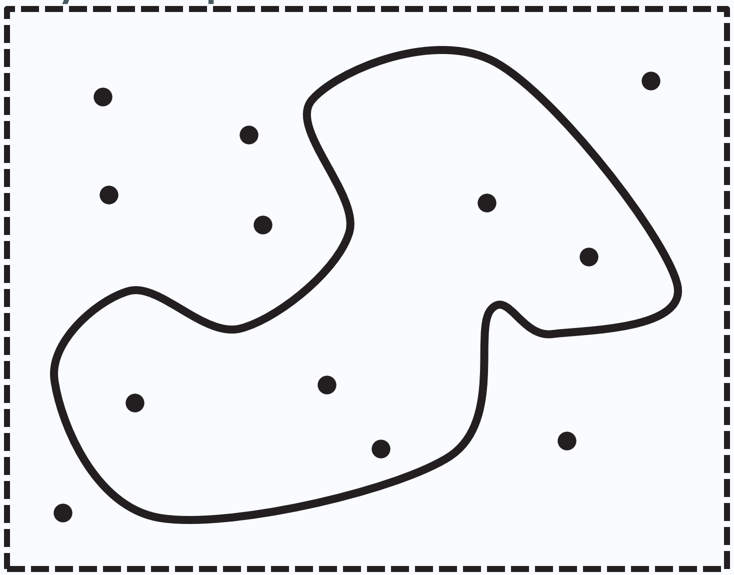




- Let's enclose the shape in a box of known area A

Monte Carlo – A Simple Example

- Now lets randomly sample inside this box

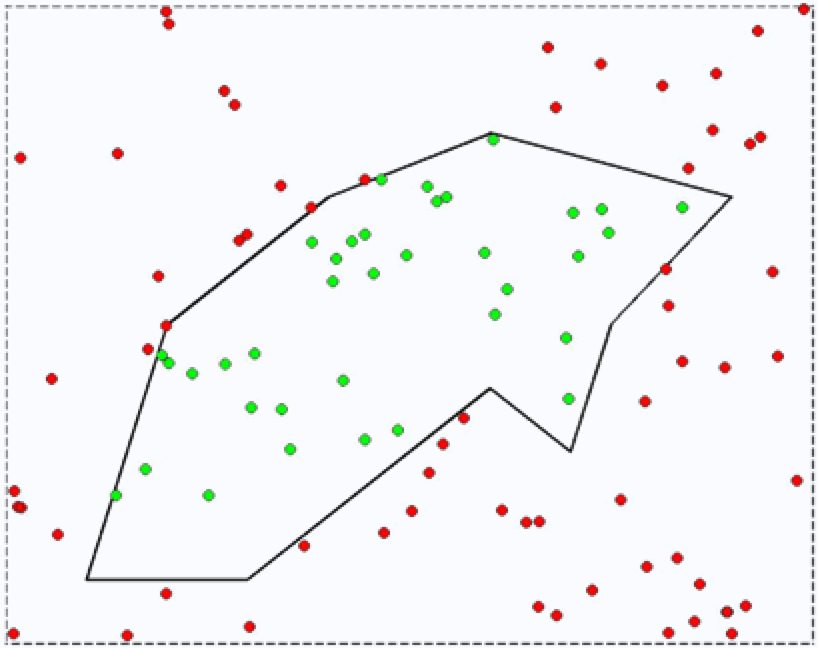


- If we sample with N points and N` are inside the irregular shape then the area of the shape will be

$$S = A\frac{N'}{N}$$

Monte Carlo – A Simple Example

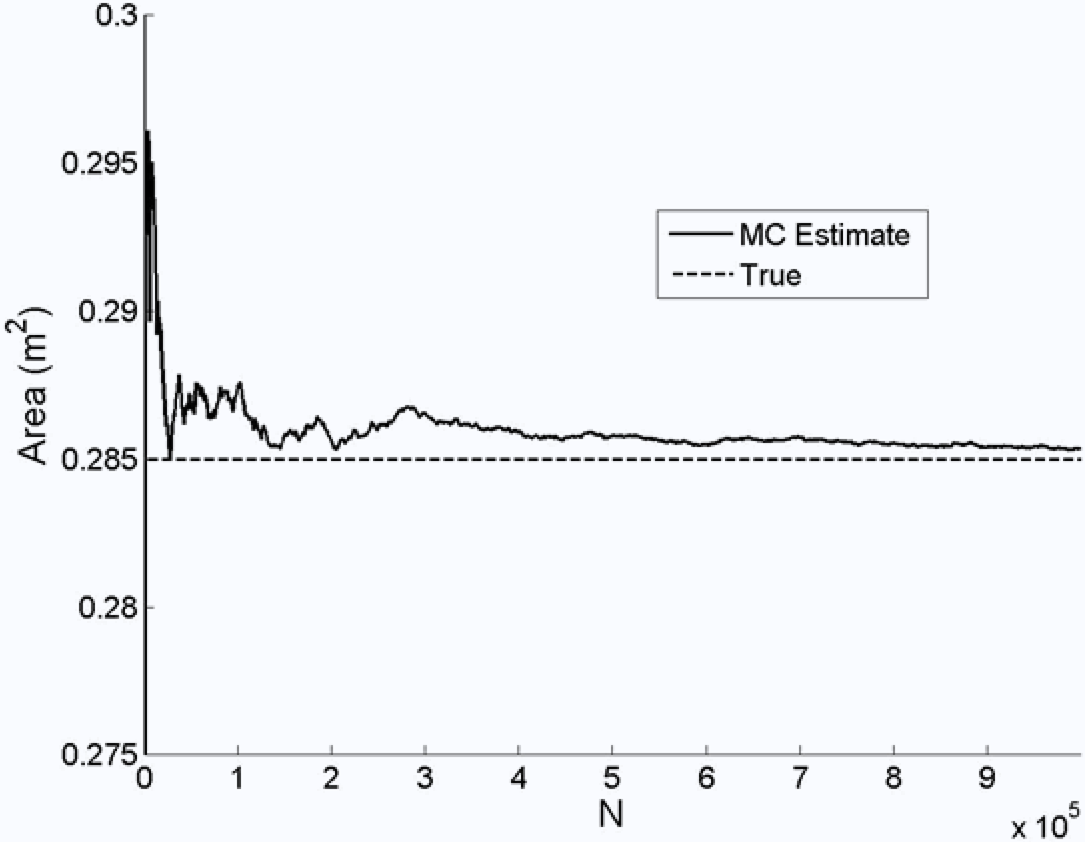
- Let's take a case where we have a polygon inside a 1.0×1.0m square



- Out of 100 points 36 are in the square giving an estimated area of 0.36m^2, clearly there's a bit of an error

Monte Carlo – A Simple Example

- However, if we continue to increase the number of sample points we get closer to the true area



Aerofoil Example Revisited

- Lets revisit our aerofoil example
- Previously we had assumed a uniform random distribution of inputs to our simulation between a fixed set of bounds
- Was this realistic?
- We will rarely encounter fixed limits on things like Reynolds number, angle of attack or Mach number, there is more likely to be a distribution based on some prior measurements
  - Typical usage of this aircraft type etc.

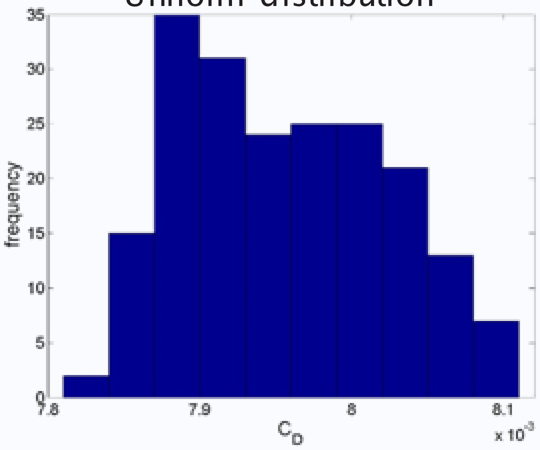
Aerofoil Example Revisited

- Let's perform a Monte Carlo simulation of the aerofoil
  - Re varies according to a normal distribution with $\mu = 6.0 \times 10^6$ & $\sigma = 0.25 \times 10^6$
  - α varies according to a normal distribution with $\mu = 2°$ & $\sigma = 0.25°$
  - M varies according to a normal distribution with $\mu = 0.2$ & $\sigma = 0.025$

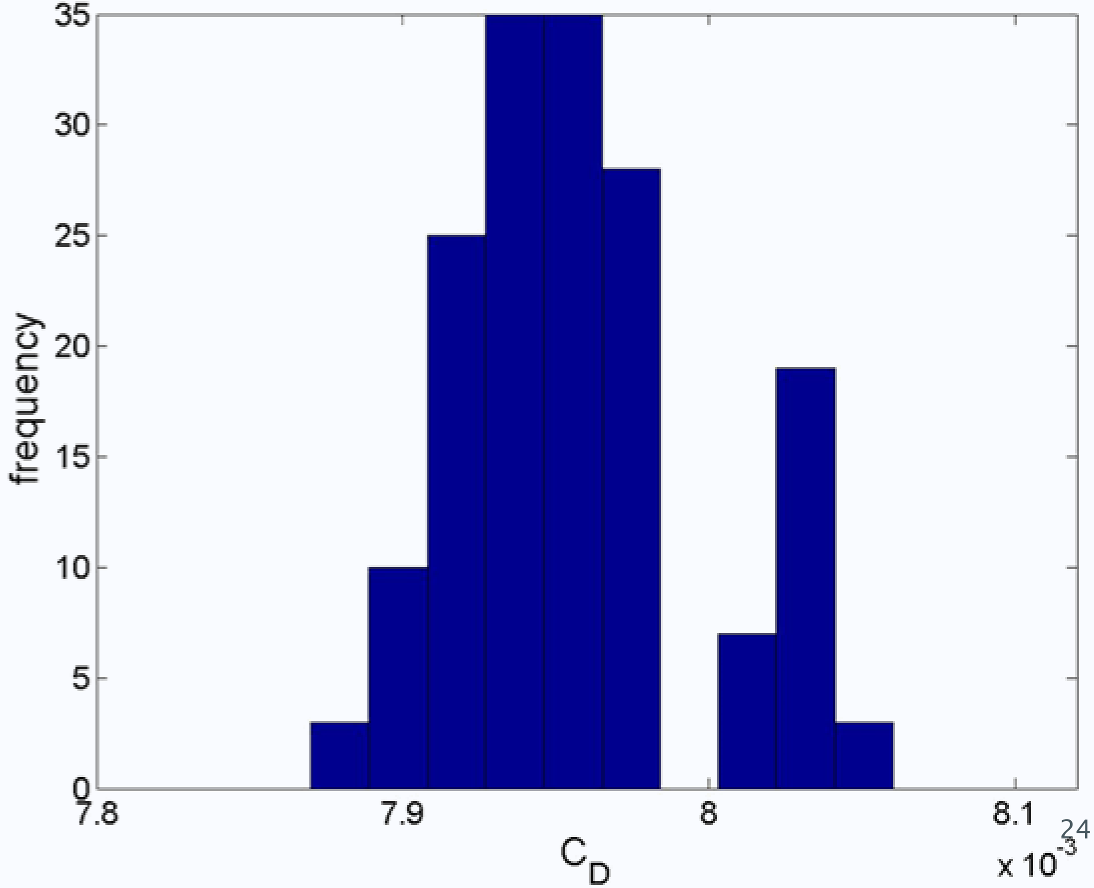
Aerofoil Example Revisited

- We now get a quite different result:

Uniform distribution



Non-Uniform distribution



Monte Carlo Convergence

- There isn't a simple way of estimating the number of runs required for a MC simulation
- Generally, the number of trials depends on:
  - The complexity of the underlying simulation
  - The required accuracy
  - The variance in the input
- High input & output variance requires more simulations

Monte Carlo Convergence

- As it's a statistical measure the error in the distribution mean can be found:

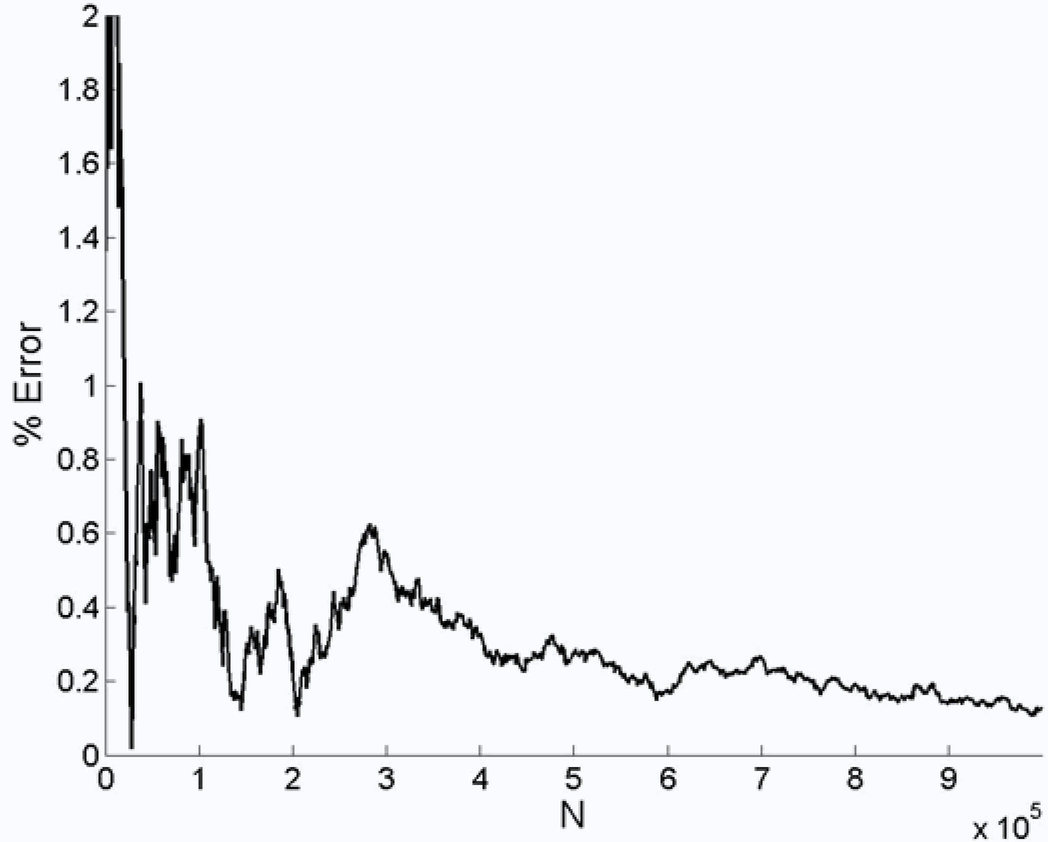
$$Er(\mu) = \frac{Z_{\alpha/2}\sigma}{\sqrt{N}}$$

- This clearly illustrates that for one order of magnitude improvement in accuracy we need a two order of magnitude increase in the number of runs
- Note that we won't know σ (the standard deviation in the output) until we start the simulation

Quasi-Monte Carlo Simulation

Quasi-Monte Carlo Methods

- As demonstrated by our simple area calculation it can take a great number of simulations for a MC to converge



Quasi-Monte Carlo Methods

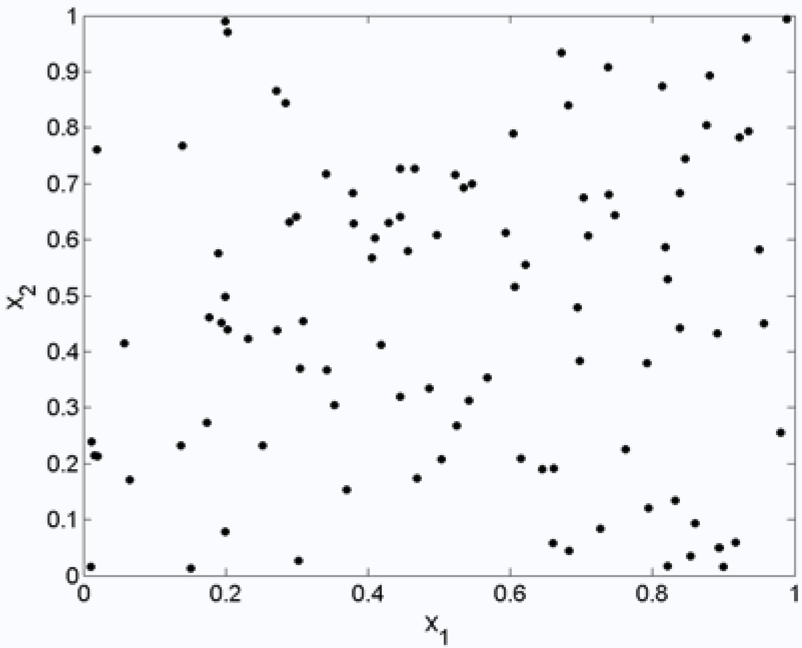
- Quasi-Monte Carlo methods aim to accelerate the rate of convergence by replacing the pseudorandom number sequence with a quasi-random number sequence
- Other than this change the two approaches are almost identical
- Using a quasi-random number sequence alters the convergence rate from $O\left({}^{1}/_{\sqrt{N}}\right)$ to $O({}^{1}/_{N})$

Quasi-Random Numbers

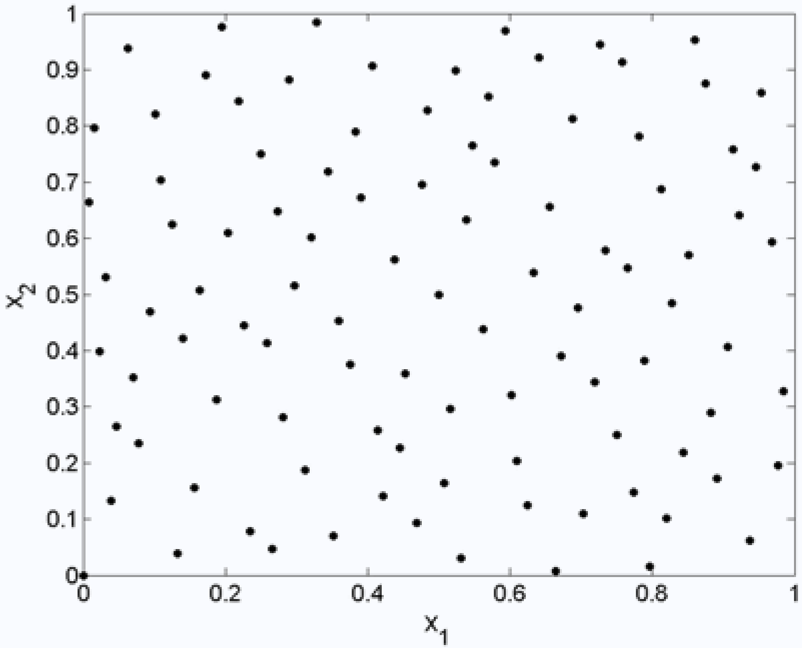
- These are a deterministic alternative to pseudorandom numbers
- Where pseudorandom numbers attempt to mimic randomness, quasi-random numbers attempt to attain better uniformity
- Standard MC accuracy is affected by the clumping of points that occurs and spaces with no points in them
- This clumping occurs due to the independence of points – new points know nothing about previous ones so there’s a chance they will fall near each other

Quasi-Random Numbers

- Quasi-random numbers use correlations between points to eliminate clumping



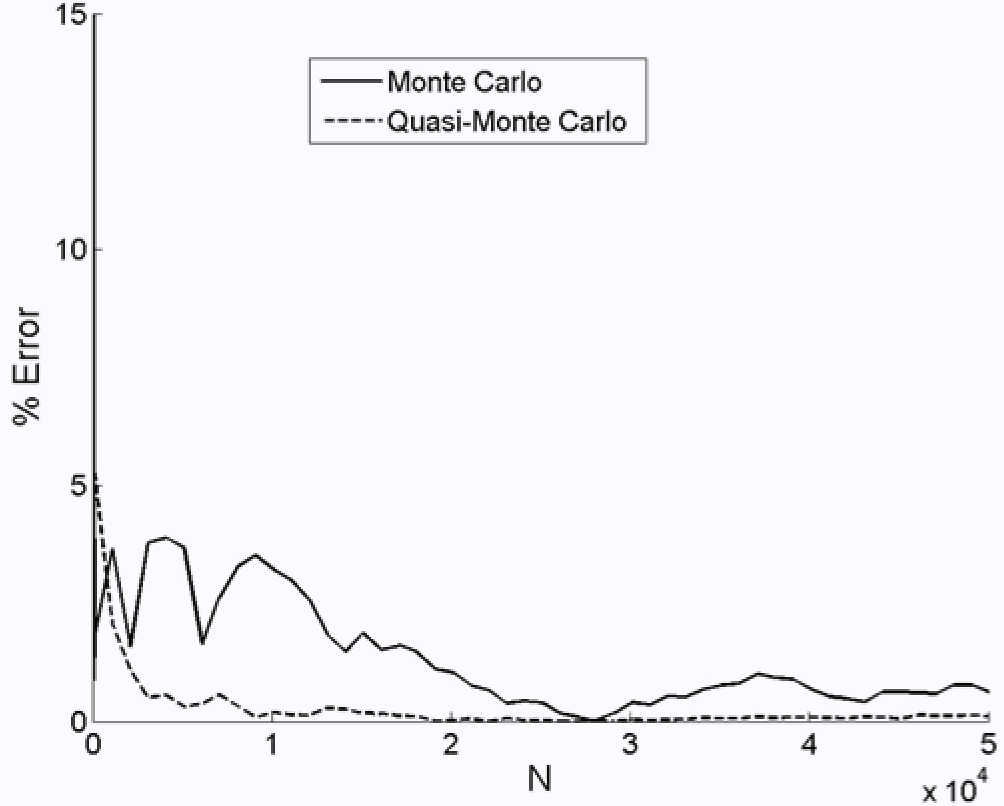
100 pseudorandom numbers



100 quasi-random numbers (Sobol sequence or LPτ)

Quasi-Monte Carlo

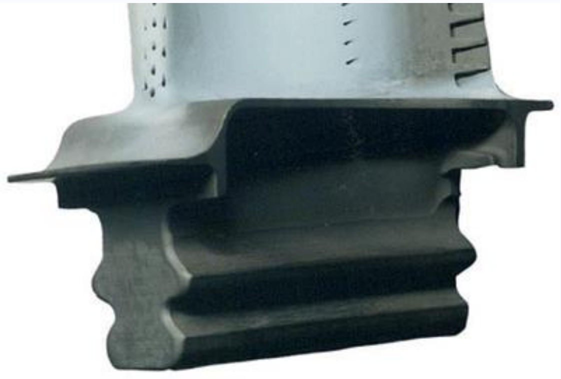
- Lets apply this approach to our area calculation



- Of course, the space-filling nature of the Sobol sequence offers it a massive advantage in this simple case

Blade Firtree Example

- Let's perform a MC analysis of a turbine blade firtree
- The stresses in a firtree are found to vary according to a normal distribution with μ = 400MPa & σ = 50MPa
- What is the probability of a blade exceeding 600MPa?



Blade Firtree Example

- Analytically:

$$P(>600) = 1 - \int_0^{600} f(s)ds = 3.16 \times 10^{-5}$$

- Monte Carlo with 100,000 points:

$$P(>600) = 5.0 \times 10^{-5}$$

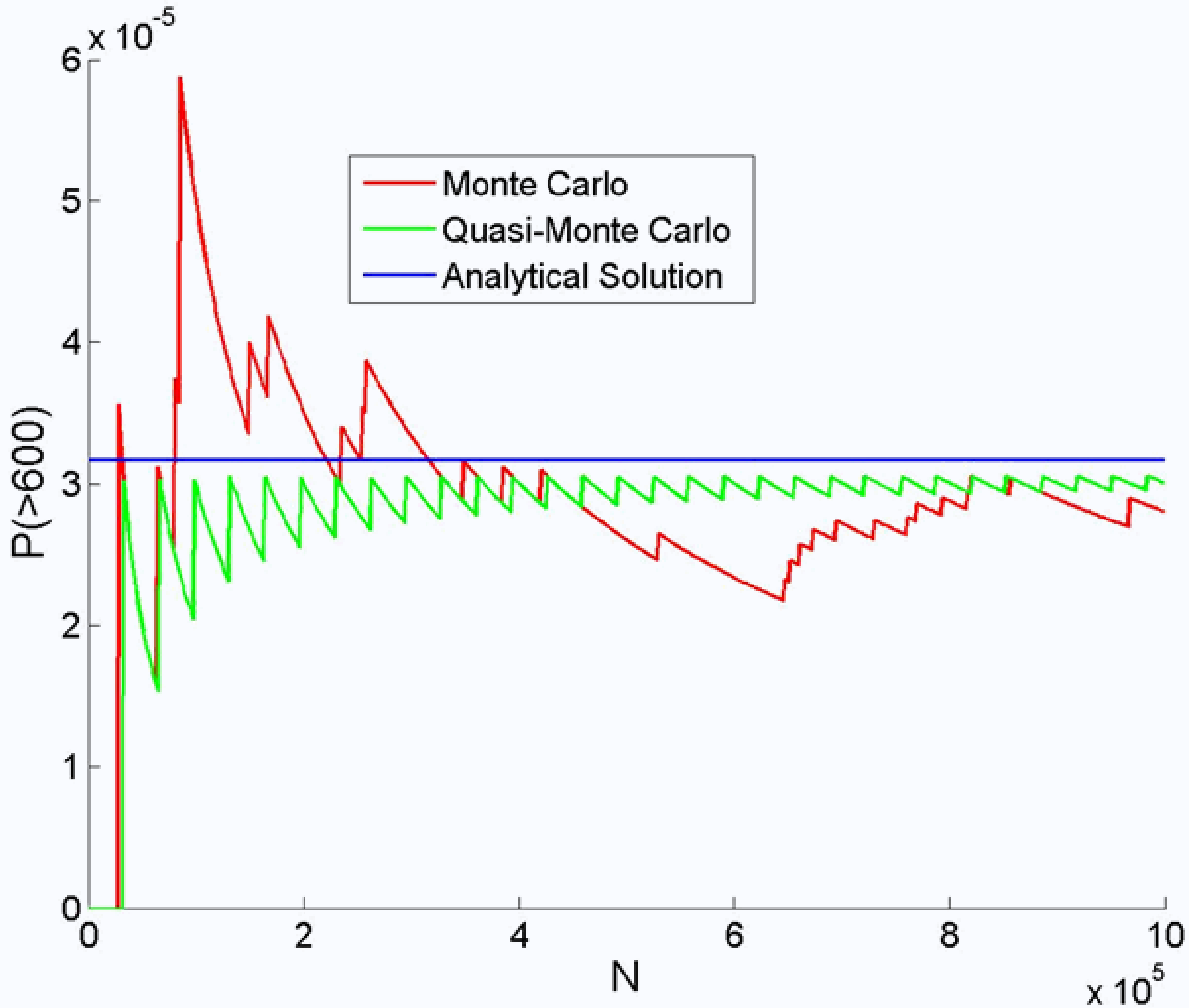
- Quasi-Monte Carlo with 100,000 points:

$$P(>600) = 3.0 \times 10^{-5}$$

- Monte Carlo with 10 million points:

$$P(>600) = 3.32 \times 10^{-5}$$

Blade Firtree Example



Monte Carlo

- This case illustrates one of the problems with Monte Carlo techniques based on pseudorandom numbers
- It can be very costly to accurately predict very small probabilities as the chances of placing a point in that region are very small
- The space filling nature of a quasi-Monte Carlo technique helps it to evaluate small probabilities much more accurately

University of Southampton

Random Number Generation

(Not Examinable)

Generating Random Numbers

- How do we create a simple random number?
  - We could use a random number table…

13962	70992	65172	28053	02190	83634	66012	70305	66761	88344
43905	46941	72300	11641	43548	30455	07686	31840	03261	89139
00504	48658	38051	59408	16508	82979	92002	63606	41078	86326
61274	57238	47267	35303	29066	02140	60867	39847	50968	96719
43753	21159	16239	50595	62509	61207	86816	29902	23395	72640
83503	51662	21636	68192	84294	38754	84755	34053	94582	29215
36807	71420	35804	44862	23577	79551	42003	58684	09271	68396
19110	55680	18792	41487	16614	83053	00812	16749	45347	88199
82615	86984	93290	87971	60022	35415	20852	02909	99476	45568
05621	26584	36493	63013	68181	57702	49510	75304	38724	15712
06936	37293	55875	71213	83025	46063	74665	12178	10741	58362
84981	60458	16194	92403	80951	80068	47076	23310	74899	87929
66354	88441	96191	04794	14714	64749	43097	83976	83281	72038
49602	94109	36460	62353	00721	66980	82554	90270	12312	56299
78430	72391	96973	70437	97803	78683	04670	70667	58912	21883

  - But what's the issue with this method?

Generating Random Numbers

- Clearly using a random number table is not really appropriate if we wanted to automate a stochastic simulation. At some point we will run out of numbers
  - The largest published table has 1 million numbers in it!
- We therefore need a method which can be used to create random numbers on demand
  - Pseudorandom numbers

Pseudorandom Numbers

- These are numbers which simulate the values of a random variable but are calculated using some formula
- E.g. John von Neumann's "mid-square method"
  - Take a four digit number, square it to create an eight digit number and take the middle four digits

$$r_1 = 0.9876 \qquad r_1^2 = 0.97535376 \qquad r_2 = 0.5353$$

$$r_2 = 0.5353 \qquad r_2^2 = 0.28654609 \qquad r_3 = 0.6546$$

Pseudorandom Numbers

- Generally such pseudorandom numbers are generated via a formula of the form:

$$r_i = g(r_{i-1})$$

- If the same initial starting number (or seed) is used then the same sequence of random numbers is generated
- This has advantages allowing us to repeat a simulation if necessary
- However, when defining $g()$ we need be aware of the period of the generator i.e. how quickly will it produce a number already defined and start to repeat itself

Congruential Generator

- The congruential generator is one of the most popular methods of generating random numbers
- Generally these are of the form:

$$m_i = (am_{i-1}) \pmod{M}$$

$$r_i = \frac{m_i}{M}$$

- Where we define an initial m_0, calculate m_i and divide by M to obtain a new random number
- mod() – modulus after division of am_{i-1} by M
  - e.g. mod(5,4) = 1 mod(4,4)=0

Examples

- One example of such a congruential generator is where:

$$m_0 = 1, \qquad M = 2^{40}, \qquad a = 5^{17}$$

- This generator has a period of 2.7×10^{11}
- Another example, used by IBM uses:

$$m_0 = 1, \qquad M = 2^{29}, \qquad a = 2^{16} + 3$$

- Of course these generators aren't of much use to machines with less than 40bits

Pseudorandom Numbers on a PC

- For PCs with less than 40bits we need an alternative approach
- Wichman & Hill's parallel approach is one such method
- In this approach three separate m values are calculated and combined to generate the random number

$$m_i' = (171 m_{i-1}') \pmod{30269}$$

$$m_i'' = (172 m_{i-1}'') \pmod{30307}$$

$$m_i''' = (170 m_{i-1}''') \pmod{30323}$$

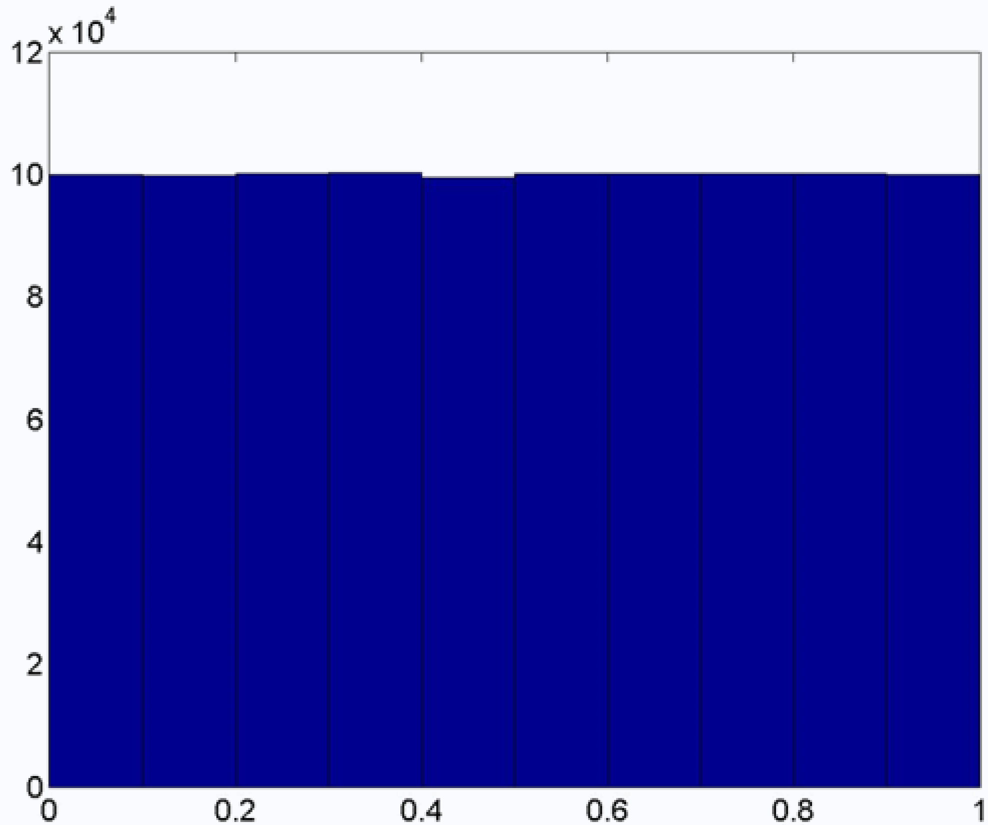
$$r_i = \left\{ \frac{m_i'}{30269} + \frac{m_i''}{30307} + \frac{m_i'''}{30323} \right\} \pmod{1}$$

Wichman-Hill's Generator

```
% script to create Wichman & Hill random numbers

% define the no. of numbers
NO_RAND = 1e6;

% initialise the storage
R = nan(NO_RAND,1);

% define the parameters
m1 = 1;
m2 = 2;
m3 = 3;

for i=1:NO_RAND;
    m1 = mod(171*m1,30269);
    m2 = mod(172*m2,30307);
    m3 = mod(170*m3,30323);
    R(i) = mod(m1/30269 + m2/30307 + m3/30323,1);
end

hist(R)
```

Random Numbers

- The methods discussed for creating pseudorandom numbers all result in numbers between 0 and 1 which are uniformly distributed



Histogram for 1e6 random numbers generated using Wichman-Hill

Exponential Distribution

```
% convert these random numbers into an exponential
% distribution
lambda = 5;

Re = log(R)./-lambda;

figure;
hist(Re)
```

Normal Distribution

```
% convert these random numbers into a normal
% distribution
mu = 10; sigma = 2.5;

Rn = mu+sigma.*sqrt(2).*erfinv(2.*R-1);

figure;
hist(Rn,20)
```