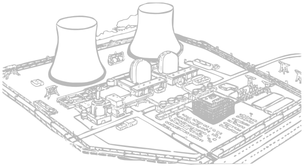
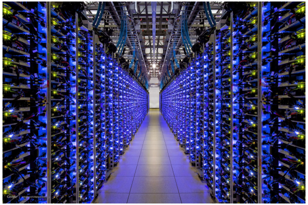


Monte Carlo methods for risk analysis

Eric Marsden

<eric.marsden@risk-engineering.org>

Context



- Some problems cannot be expressed in analytical form
- Some problems are difficult to define in a deterministic manner
- Modern computers are amazingly fast
- Allow you to run "numerical experiments" to see what happens "on average" over a large number of runs
  - also called *stochastic simulation*

Etymological note: stochastic is a synonym for probabilistic: the former comes from the Greek word stokházomai, "aiming at a target, guessing" and the latter from the Latin term probābilis ("probable, credible")

Monte Carlo simulation

- ▷ Monte Carlo method: computational method using repeated random sampling to obtain numerical results
  - named after gambling in casinos
- ▷ Technique invented during the Manhattan project (US nuclear bomb development)
  - their development coincides with invention of electronic computers, which greatly accelerated repetitive numerical computations
- ▷ Widely used in engineering, finance, business, project planning
- ▷ Implementation with computers uses **pseudo-random number generators**

  → `random.org/randomness/`

Photo: Monte Carlo casino

Monte Carlo simulation: steps

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

Aggregate the results

Define the **domain of possible inputs**.

The simulated "universe" should be similar to the universe whose behavior we wish to describe and investigate.

Monte Carlo simulation: steps

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

Aggregate the results

Generate inputs randomly from a probability distribution over the domain

- inputs should be generated so that their characteristics are similar to the real universe we are trying to simulate
- in particular, dependencies between the inputs should be represented

RISK ENGINEERING

Monte Carlo simulation: steps

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

Aggregate the results

The computation should be **deterministic**.

Monte Carlo simulation: steps

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

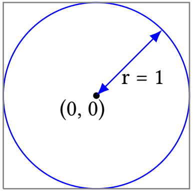
Aggregate the results

Aggregate the results to obtain the output of interest.

Typical outputs:

- histogram
- summary statistics (mean, standard deviation...)
- confidence intervals

Example: estimate the value of pi



- Consider the largest circle which can be fit in the square ranging on $\mathbb{R}^2$ over $[-1, 1]^2$
  - the circle has radius 1 and area π
  - the square has an area of $2^2 = 4$
  - the ratio between their areas is thus $\frac{\pi}{4}$
- We can approximate the value of π using the following Monte Carlo procedure:
  1. draw the square over $[-1, 1]^2$
  2. draw the largest circle that fits inside the square
  3. randomly scatter a large number N of grains of rice over the square
  4. count how many grains fell inside the circle
  5. the count divided by N and multiplied by 4 is an approximation of π

Example: estimate the value of pi

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

Aggregate the results

All points within the $[-1, 1]^2$ unit square, uniformly distributed.

Example: estimate the value of pi

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

Aggregate the results

Generate one point `(x, y)` from the unit square in Python:

```
x = numpy.random.uniform(-1, 1)
y = numpy.random.uniform(-1, 1)
```

RISK ENGINEERING

Example: estimate the value of pi

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

Aggregate the results

Test whether a randomly generated point `(x, y)` is within the circle:

```
if numpy.sqrt(x**2 + y**2) < 1:
    print("The point is inside")
```

RiSK ENGINEERING

Example: estimate the value of pi

Define **possible inputs**

Generate inputs randomly

Computation on the inputs

Aggregate the results

Count the proportion of points that are within the circle:

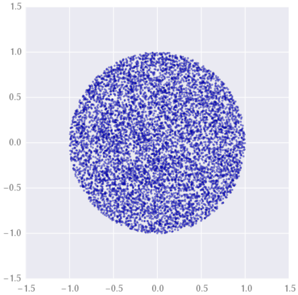
```
N = 10_000
inside = 0
for i in range(N):
    x = numpy.random.uniform(-1, 1)
    y = numpy.random.uniform(-1, 1)
    if numpy.sqrt(x**2 + y**2) < 1:
        inside += 1
p = inside / float(N)
print("Proportion inside: {}".format(p))
```

Example: estimate the value of pi

Putting it all together: here is an implementation in Python with the NumPy library.

```
import numpy

N = 10_000
inside = 0
for i in range(N):
    x = numpy.random.uniform(-1, 1)
    y = numpy.random.uniform(-1, 1)
    if numpy.sqrt(x**2 + y**2) < 1:
        inside += 1
print(4*inside/float(N))
3.142
```



Download the associated Python notebook at `risk-engineering.org`

Exercise: speed of convergence

- Mathematical theory states that the **error** of a Monte Carlo estimation technique should decrease proportionally to the **square root of the number of trials**
- **Exercise**: modify the Monte Carlo procedure for estimation of π
  - within the loop, calculate the current estimation of π
  - calculate the error of this estimation
  - plot the error against the square root of the current number of iterations

RISK ENGINEERING

Why does it work?

- The **law of large numbers** describes what happens when performing the same experiment many times
- After many trials, the average of the results should be close to the expected value
  - increasing the number of trials will increase accuracy
- For Monte Carlo simulation, this means that we can learn properties of a random variable (mean, variance, *etc.*) simply by simulating it over many trials

Example: coin flipping

Flip a coin 10 times. What is the probability of getting more than 3 heads?

Example: coin flipping

Flip a coin 10 times. What is the probability of getting more than 3 heads?

Analytical solution

Let's try to remember how the binomial distribution works. Here we have $n = 10$, $p = 0.5$ and `cdf(3)` is the probability of seeing three or fewer heads.

```
> from scipy.stats import binom
> throws = binom(n=10, p=0.5)
> 1 - throws.cdf(3)
0.828125
```

RISK ENGINEERING

Example: coin flipping

Flip a coin 10 times. What is the probability of getting more than 3 heads?

Analytical solution

Let's try to remember how the binomial distribution works. Here we have $n = 10$, $p = 0.5$ and `cdf(3)` is the probability of seeing three or fewer heads.

```
> from scipy.stats import binom
> throws = binom(n=10, p=0.5)
> 1 - throws.cdf(3)
0.828125
```

Monte Carlo simulation

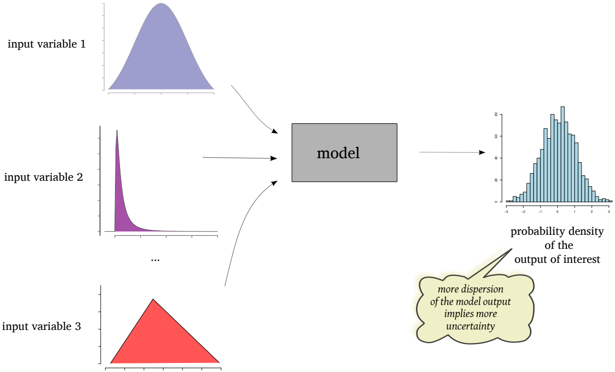
Just simulate the coin flip sequence a million times and count the simulations where we have more than 3 heads.

```
import numpy
def headcount():
    tosses = numpy.random.uniform(0, 1, 10)
    return (tosses > 0.5).sum()

N = 1_000_000
count = 0
for i in range(N):
    if headcount() > 3: count += 1
count / float(N)
0.828117
```

Application to uncertainty analysis

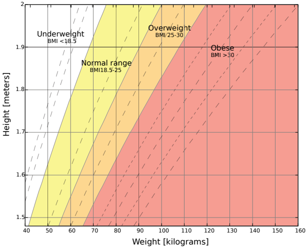
- Uncertainty analysis: propagate uncertainty on input variables through the model to obtain a probability distribution of the output(s) of interest
- Uncertainty on each input variable is characterized by a probability density function
- Run the model a large number of times with input values drawn randomly from their PDF
- Aggregate the output uncertainty as a probability distribution



→ *slides on uncertainty in risk analysis at risk-engineering.org*

RISK ENGINEERING

A simple application in uncertainty propagation



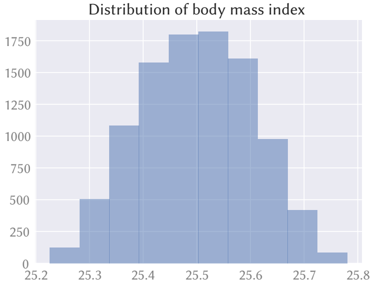
- The **body mass index** (BMI) is the ratio $\frac{\text{body mass } (kg)}{\text{body height } (m)^2}$
  - often used as an (imperfect) indicator of obesity or malnutrition
- **Task**: calculate your BMI and the associated uncertainty interval, assuming:
  - your weight scale tells you that you weigh 84 kg (precision shown to the nearest kilogram)
  - a tape measure says you are between 181 and 182 cm tall (most likely value is 181.5 cm)
- We will run a Monte Carlo simulation on the model BMI $= \frac{m}{h^2}$ with
  - m drawn from a $U(83.5, 84.5)$ uniform distribution
  - h drawn from a $T(1.81, 1.815, 1.82)$ triangular distribution

A simple application in uncertainty propagation

```
import numpy
from numpy.random import *
import matplotlib.pyplot as plt

N = 10_000
def BMI():
    m = uniform(83.5, 84.5)
    h = triangular(1.81, 1.815, 1.82)
    return m / h**2

sim = numpy.zeros(N)
for i in range(N):
    sim[i] = BMI()
plt.hist(sim)
```



A simple application in uncertainty propagation

- Note: analytical estimation of the output uncertainty would be difficult even on this trivial example
- With more than two input probability distributions, becomes *very* difficult
- Quantile measures, often needed for risk analysis, are often difficult to calculate analytically
  - *"what is the 95th percentile of the high water level?"*
- Monte Carlo techniques are a **simple and convenient** way to obtain these numbers
  - express the problem in a direct way and **let the computer do the hard work**!

RISK ENGINEERING

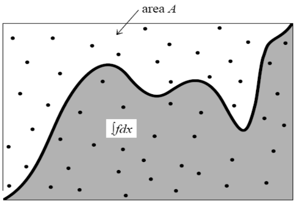
Second example for uncertainty propagation

- X and Y are both uniformly distributed over $[0, 100]$
- We are interested in the distribution of $Z = X \times Y$
- **Q**: What is the 95$^{\text{th}}$ percentile of Z?

```
N = 10_000
zs = numpy.zeros(N)
for i in range(N):
    x = numpy.random.uniform(0, 100)
    y = numpy.random.uniform(0, 100)
    zs[i] = x * y
numpy.percentile(zs, 95)
```

RISK ENGINEERING

Application in resolving numerical integrals



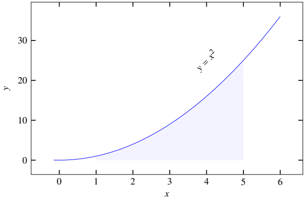
- Assume we want to evaluate an integral $\int_I f(x)\mathrm{d}x$
- **Principle**: the integral to compute is related to the expectation of a random variable

$$\mathbb{E}(f(X)) = \int_I f(x)\mathrm{d}x$$

- **Method**:
  - Sample points within I
  - Calculate the mean of the random variable within I
  - Integral = sampled area × mean
- **Advantages**: the method works even without knowing the analytical form of f, and also if f is not continuous

RISK ENGINEERING

Trivial integration example



Task: find the shaded area, $\int_1^5 x^2 \, \mathrm{d}x$

Analytical solution

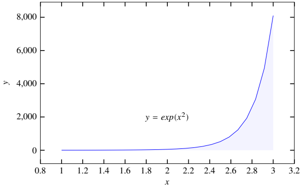
```
import sympy
x = sympy.Symbol("x")
i = sympy.integrate(x**2)
i.subs(x, 5) - i.subs(x, 1)
124/3
float(i.subs(x, 5) - i.subs(x, 1))
41.333333333333336
```

Numerical solution

```
N = 100_000
accum = 0
for i in range(N):
    x = numpy.random.uniform(1, 5)
    accum += x**2
area = 4
integral = area * accum / float(N)
41.278
```

The SymPy library for symbolic mathematics in Python is available from sympy.org

Simple integration example



Task: find the shaded area, $\int_1^3 e^{x^2}\,\mathrm{d}x$

Analytical solution

```
import sympy
x = sympy.Symbol("x")
i = sympy.integrate(sumpy.exp(x**2))
i.subs(x, 3) - i.subs(x, 1)
-sqrt(pi)*erfi(1)/2 + sqrt(pi)*erfi(3)/2
float(i.subs(x, 3) - i.subs(x, 1))
1443.082471146807
```

Numerical solution

```
N = 100_000
accum = 0
for i in range(N):
    x = numpy.random.uniform(1, 3)
    accum += numpy.exp(x**2):
        count += 1
area = 3 - 1
integral = area * accum / float(N)
1451.3281492713274
```

The SymPy library for symbolic mathematics in Python is available from sympy.org

2D integration example

Task: resolve the double integral

$$\int_0^1 \int_4^6 cos(x^4) + 3y^2 \, dx \, dy$$

Analytical solution

```
import sympy
x = sympy.Symbol("x")
y = sympy.Symbol("y")
d1 = sympy.integrate(sympy.cos(x**4) + 3 * y**2, x)
d2 = sympy.integrate(d1.subs(x, 6) - d1.subs(x, 4), y)
sol = d2.subs(y, 1) - d2.subs(y, 0)
float(sol)
2.005055086749674
```

Numerical solution

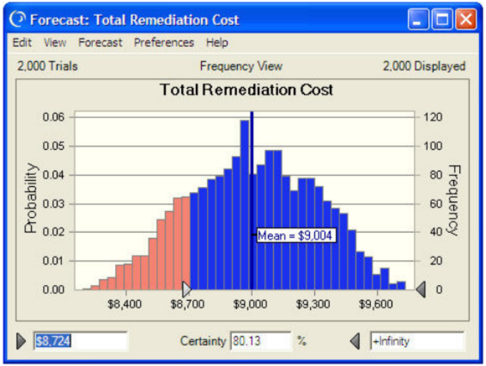
```
N = 100_000
accum = 0
for i in range(N):
    x = numpy.random.uniform(4, 6)
    y = numpy.random.uniform(0, 1)
    accum += numpy.cos(x**4) + 3 * y * y
volume = 2 * 1
integral = volume * accum/float(N)
2.0100840446967103
```

RiSK ENGINEERING

Relevant tools

(if you can't use Python...)

Relevant commercial tools

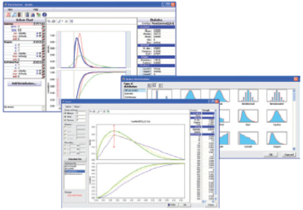


Example tools with Excel integration:

- Palisade TopRank®
- Oracle Crystal Ball®

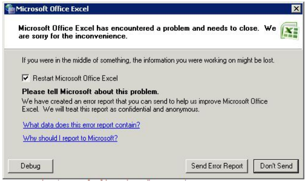
Typically quite expensive…

Free plugins for Microsoft Excel



- ▷ A free Microsoft Excel plugin from Vose Software
  - `vosesoftware.com/products/modelrisk/`
  - "standard" version is free (requires registration)
- ▷ Simtools, a free add-in for Microsoft Excel by R. Myerson, professor at the University of Chicago
  - `home.uchicago.edu/ rmyerson/addins.htm`
- ▷ MonteCarlito, a free add-in for Microsoft Excel
  - `montecarlito.com`
  - distributed under the terms of the GNU General Public Licence

Beware the risks of Excel!



- ▷ Student finds serious errors in austerity research undertaken by Reinhart and Rogoff (cells left out of calculations of averages…)
- ▷ JP Morgan underestimates value at risk due to a spreadsheet error
- ▷ London 2012 Olympics: organization committee oversells synchronized swimming events by 10 000 tickets
- ▷ Cement factory receives 350 000 USD fine for a spreadsheet error (2011, Arizona)

Source: European Spreadsheet Risks Interest Group, `eusprig.org/horror-stories.htm`

Sampling methods

Latin Hypercube Sampling

- ▷ With standard random sampling, you may end up with samples unevenly spread out over the input space
- ▷ Latin Hypercube Sampling (LHS):
  - split up each input variable into a number of equiprobable intervals
  - sample separately from each interval
- ▷ Also called *stratified sampling without replacement*
- ▷ Typically leads to faster convergence than Monte Carlo procedures using standard random sampling

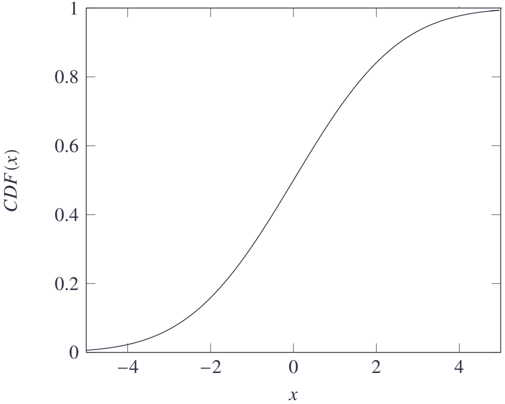
RISK ENGINEERING

Sampling methods illustrated

Standard sampling (one dimensional):

1. generate a random number from a uniform distribution between 0 and 1
2. use the inverse CDF of the target distribution (the percentile function) to calculate the corresponding output
3. repeat

Illustrated to the right with the normal distribution.

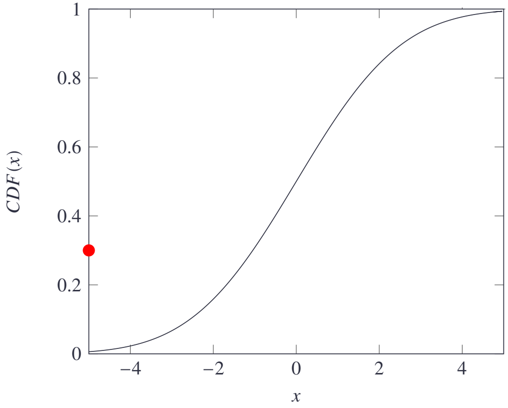


Sampling methods illustrated

Standard sampling (one dimensional):

1. generate a random number from a uniform distribution between 0 and 1
2. use the inverse CDF of the target distribution (the percentile function) to calculate the corresponding output
3. repeat

Illustrated to the right with the normal distribution.

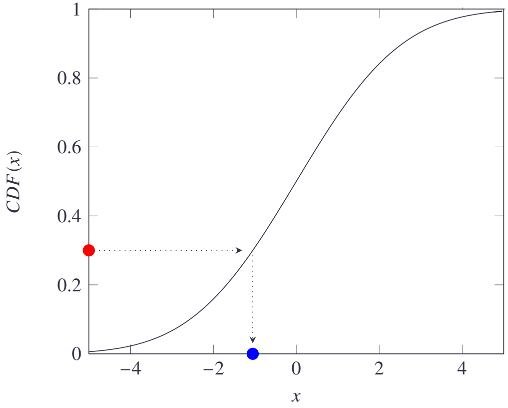


Sampling methods illustrated

Standard sampling (one dimensional):

1. generate a random number from a uniform distribution between 0 and 1
2. use the inverse CDF of the target distribution (the percentile function) to calculate the corresponding output
3. repeat

Illustrated to the right with the normal distribution.

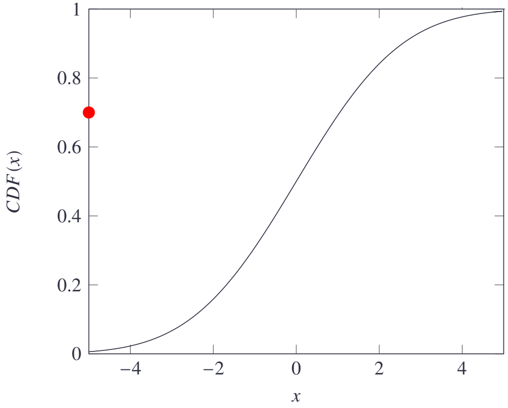


Sampling methods illustrated

Standard sampling (one dimensional):

1. generate a random number from a uniform distribution between 0 and 1
2. use the inverse CDF of the target distribution (the percentile function) to calculate the corresponding output
3. repeat

Illustrated to the right with the normal distribution.

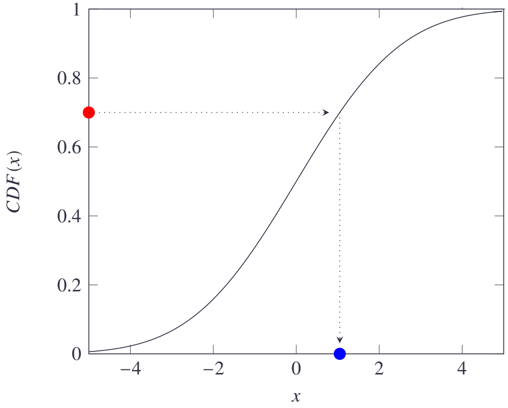


Sampling methods illustrated

Standard sampling (one dimensional):

1. generate a random number from a uniform distribution between 0 and 1
2. use the inverse CDF of the target distribution (the percentile function) to calculate the corresponding output
3. repeat

Illustrated to the right with the normal distribution.

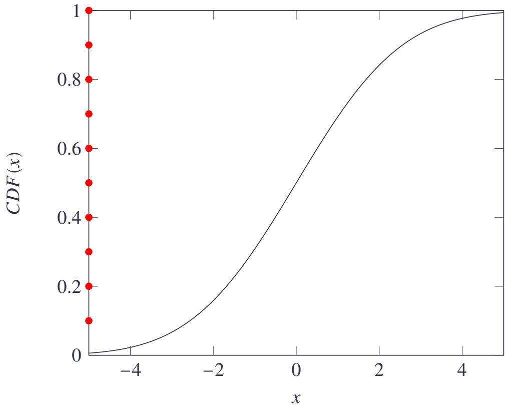


Latin Hypercube Sampling: illustration

Latin hypercube sampling (one dimensional):

1. split the [0,1] interval into 10 equiprobable intervals
2. propagate via the inverse CDF to the output distribution
3. take $N/10$ standard samples from each interval of the output distribution

Illustrated to the right with the normal distribution.

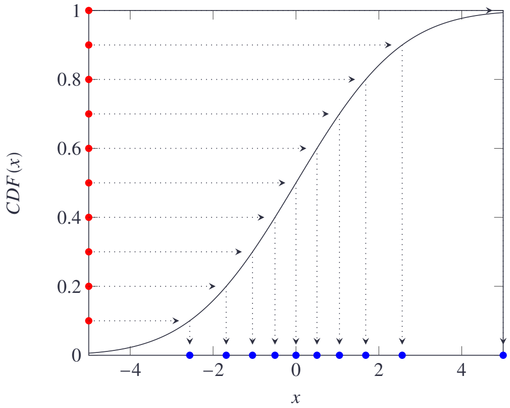


Latin Hypercube Sampling: illustration

Latin hypercube sampling (one dimensional):

1. split the [0,1] interval into 10 equiprobable intervals
2. propagate via the inverse CDF to the output distribution
3. take $N/10$ standard samples from each interval of the output distribution

Illustrated to the right with the normal distribution.



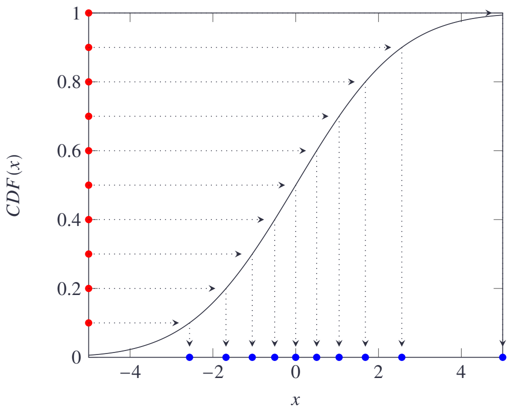
Note: spacing of red points is regular; more space between blue points near the tails of the distribution (where probability density is lower)

Latin Hypercube Sampling: illustration

Latin hypercube sampling (one dimensional):

1. split the [0,1] interval into 10 equiprobable intervals
2. propagate via the inverse CDF to the output distribution
3. take $N/10$ standard samples from each interval of the output distribution

Illustrated to the right with the normal distribution.

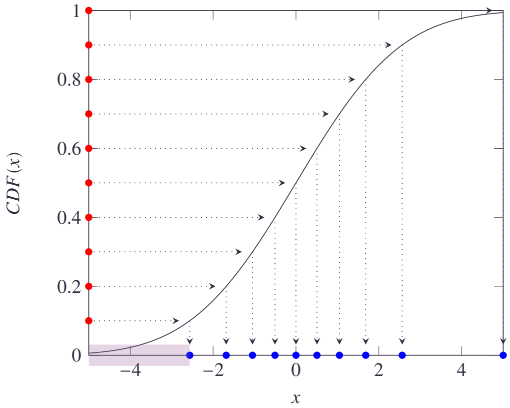


Latin Hypercube Sampling: illustration

Latin hypercube sampling (one dimensional):

1. split the [0,1] interval into 10 equiprobable intervals
2. propagate via the inverse CDF to the output distribution
3. take $N/10$ standard samples from each interval of the output distribution

Illustrated to the right with the normal distribution.

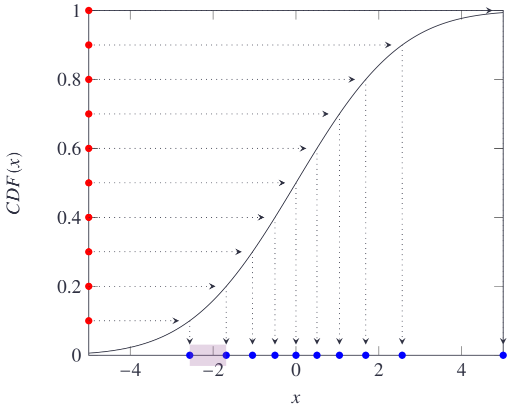


Latin Hypercube Sampling: illustration

Latin hypercube sampling (one dimensional):

1. split the [0,1] interval into 10 equiprobable intervals
2. propagate via the inverse CDF to the output distribution
3. take $N/10$ standard samples from each interval of the output distribution

Illustrated to the right with the normal distribution.

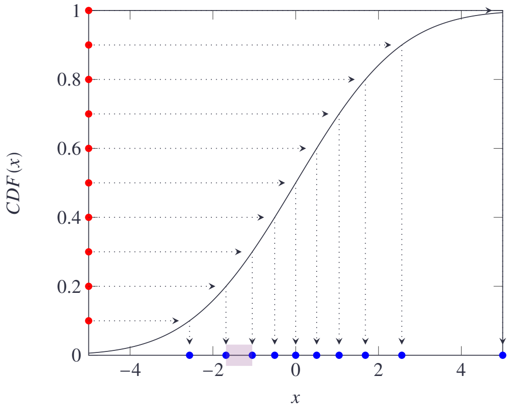


Latin Hypercube Sampling: illustration

Latin hypercube sampling (one dimensional):

1. split the [0,1] interval into 10 equiprobable intervals
2. propagate via the inverse CDF to the output distribution
3. take $N/10$ standard samples from each interval of the output distribution

Illustrated to the right with the normal distribution.



Note: this method assumes we know how to calculate the inverse CDF

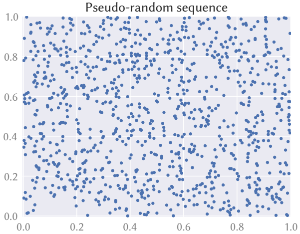
Other sampling techniques

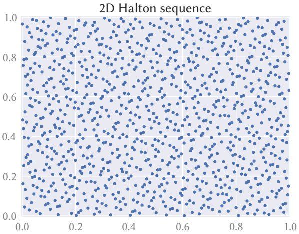
Another random sampling technique you may see in the literature: use of **low-discrepancy sequences** to implement **quasi-Monte Carlo sampling**

- low-discrepancy (or "quasi-random") sequences are constructed deterministically using formulæ
- they fill the input space more quickly than pseudorandom sequences, so lead to faster convergence
- intuition behind these types of sequences: each time you draw a new point it is placed as far away as possible from all points you already have

Low discrepancy sequences

A low discrepancy sequence is a deterministic mathematical sequence that doesn't show clusters and tends to fill space more uniformly than pseudo-random points. (Pseudo-random means as random as you can get when working with a computer.)





Commonly used low discrepancy sequences for Monte Carlo modelling include the Halton sequence and the Sobol' sequence.

More information: see the Jupyter/Python notebook at `risk-engineering.org`

The Saint Petersberg game

- ▷ You flip a coin repeatedly until a tail first appears
  - the pot starts at 1€ and doubles every time a head appears
  - you win whatever is in the pot the first time you throw tails and the game ends
- ▷ For example:
  - T (tail on the first toss): win 1€
  - H T (tail on the second toss): win 2€
  - H H T: win 4€
  - H H H T: win 8€
- ▷ Reminder (see associated slides on *Economic viewpoint on risk transfer*): the expected value of this game is infinite

→ let's estimate the expected value using a Monte Carlo simulation

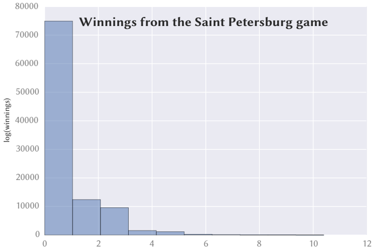
The Saint Petersburg game and limits of Monte Carlo methods

```
import numpy, matplotlib.pyplot as plt

def petersburg():
    payoff = 1
    while numpy.random.uniform() > 0.5:
        payoff *= 2
    return payoff

N = 1_000_000
games = numpy.zeros(N)
for i in range(N):
    games[i] = petersburg()

plt.hist(numpy.log(games), alpha=0.5)
print(games.mean())
12.42241
```



This game illustrates a situation where very unlikely events have an extremely high impact on the mean outcome. Monte Carlo simulation will not allow us to obtain a good estimation of the true (theoretical) expected value.

Image credits

- Monte Carlo casino on slide 3: Wikimedia Commons, CC BY licence
- Body mass index chart on slide 10: InvictaHOG from Wikimedia Commons, public domain
- Cat on slide 12: Marina del Castell via `flic.kr/p/otQtCc`, CC BY licence

For more information

- Harvard course on Monte Carlo methods, `harvard.edu/courses/am207/`
- MIT OpenCourseWare notes from the *Numerical computation for mechanical engineers* course
- Article *Principles of Good Practice for Monte Carlo Techniques*, Risk Analysis, 1994, DOI: 10.1111/j.1539-6924.1994.tb00265.x
- Book *The Monte Carlo Simulation Method for System Reliability and Risk Analysis*, Enrico Zio, ISBN: 978-1447145882

For more free content on risk engineering, visit `risk-engineering.org`

Feedback welcome!

Was some of the content unclear? Which parts were most useful to you? Your comments to `feedback@risk-engineering.org` (email) or `@LearnRiskEng` (Twitter) will help us to improve these materials. Thanks!

`@LearnRiskEng`

`fb.me/RiskEngineering`

For more free content on risk engineering, visit `risk-engineering.org`