Random Numbers for Simulation

Things to talk about

1. Short intro on Simulation
2. Why we need randomness
3. How to generate random numbers
4. How to program RNGs

1. Short intro on Simulation

- What is a simulation exactly? It is a computer program built to model a particular system so that experiments can be carried out on a system *virtually*, and not on the actual system.
  - Global warming
  - Atomic bombs

Other Simulations

- Brain
- Crime
- Housing patterns
- Traffic networks
- Stellar dynamics
- Massive crowds

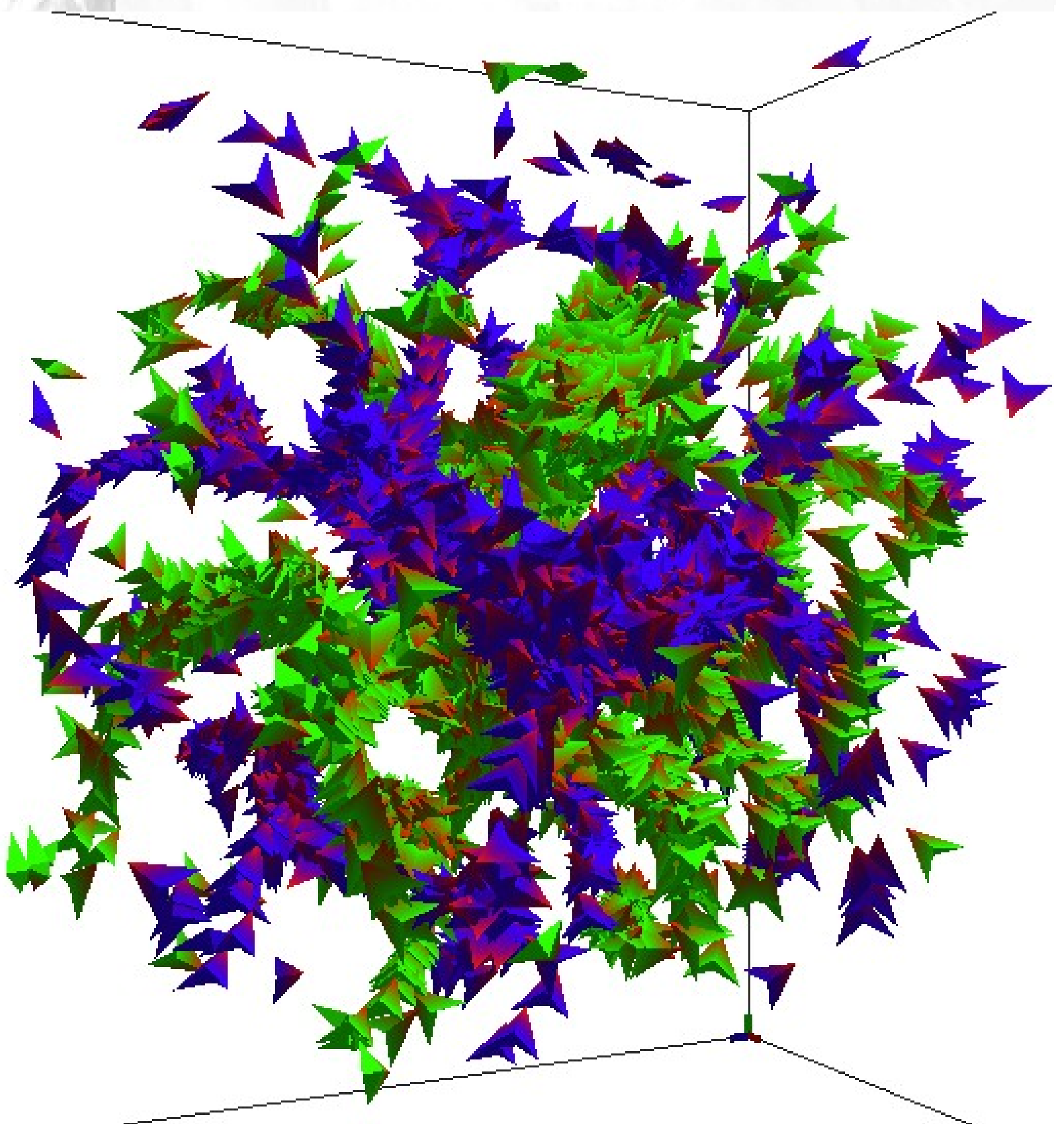
Pros

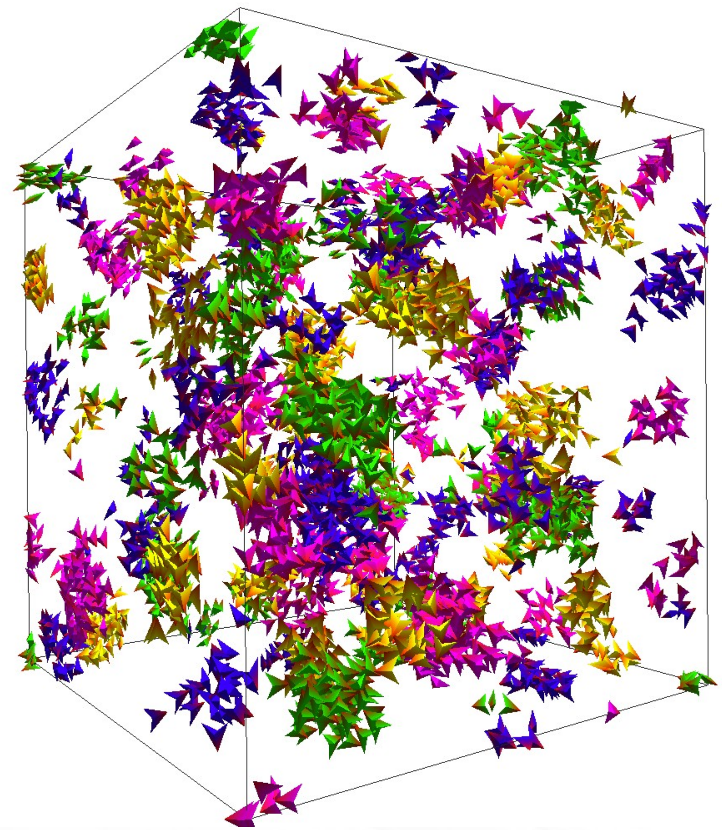
- Computer simulations are useful when the actual experiment is:
  - Physically impossible (greenhouse effect)
  - Expensive, dangerous, time consuming
  - On a system that doesn’t exist
    (design of a plane)
  - We want to carry out “what-if” testing

Cons

- Simulations might not be best when:
  - It is expensive to make
  - There are many ways of formulating the problem
  - The results are difficult to verify or check
  - We don’t understand why the simulation works or doesn’t work
  - When a simulation is non-deterministic, we must be careful in concluding – obtaining averages, etc

Example: Flocking Simulations



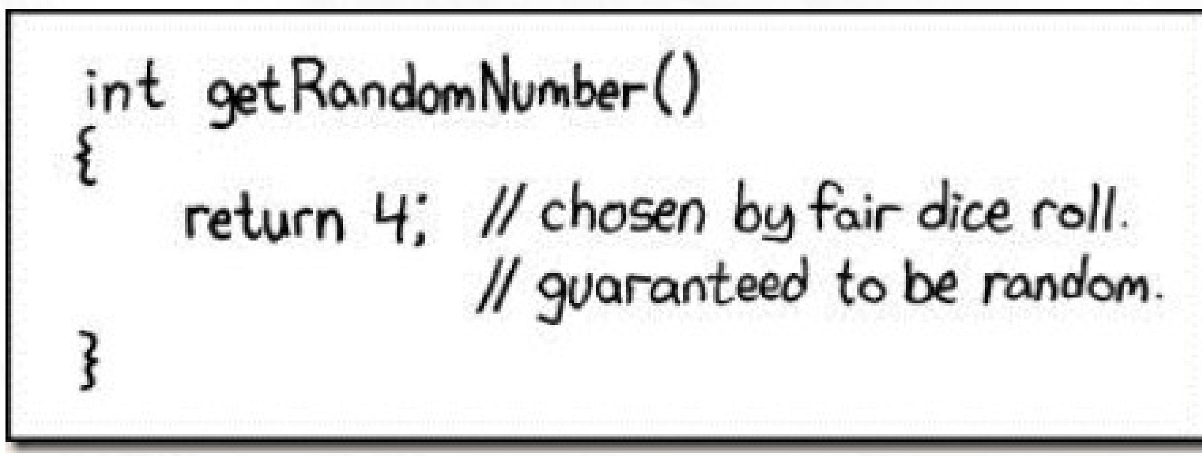


How to program simulations

- First analyse the problem.
  - **This is called the “modelling” phase.**
- **A simulation is the implementation of a model.**

- A model is an abstraction of a problem, an idea of how a system works
- Distill problems into simple “models”
- Unrealistic to simulate 100% realism, not a good goal – this is a lack of direction

2. Why do we need randomness?



Why do we need randomness?

- Otherwise there would be no Lotto
- No games except for puzzles
- Every solitaire game would be the same

More technically,

- Simulations aim to analyse complex real-world problems, which are not easily reproducible
  - Instantaneous conditions may differ, and hence different results might appear
- Simpler reason: in solitaire, how does the computer shuffle cards?
- Random goes beyond simulation, we need it everywhere – cryptography in particular

3. How to generate random numbers

- Random numbers (also known as random deviates) are computed by random number generators (**RNGs**)
- We can’t make random numbers on a computer
- We can only make “pseudo-random” deviates
- A very common (and poor) pseudo-RNG (PRNG) is the Linear Congruential Generator (LCG)

Linear Congruential Generator

Formula

$$X_{n+1} = (aX_n + c) \mod m$$

X is a random number, *a* is an integer multiplier, *c* is an additive constant, *m* is the modulus

How to pick the parameters?

- More of an art. glibc (ie. GCC) uses:

$$m = 2^{31}$$
$$a = 1103515245$$
$$c = 12345$$

Calculating a random number

- Start with a value for $X_0 = 123$ (“seed”!!)
- Multiply with *a*, then add *c*, then *mod m*!

$$X_{n+1} = (1103515245(123) + 12345) \mod 2^{31}$$

```
unsigned long long seed = 123,
m = (unsigned long long)2 << 31,
a = 1103515245;

next = ((a*seed) + 12345) % m;

printf(“%llu\n”, next);
```

Properties and problems of RNGs

- The **period** of an RNG is the number of deviates it can produce, before wrapping back around.
- The LCG wraps around very quickly
- Also, you'd expect all the bits in a deviate to be equally likely to be 0 or 1
  - In the LCG, the low-order bits are less random!

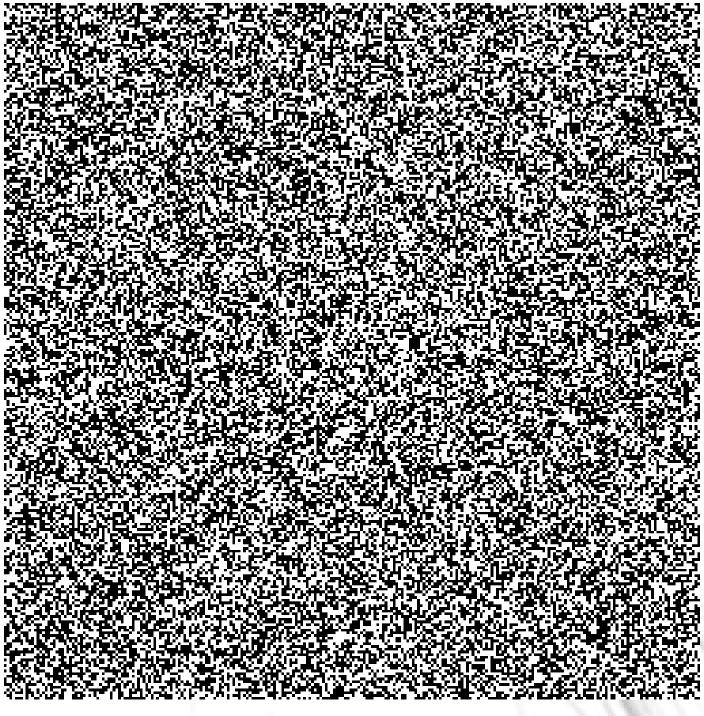
Which RNG to choose

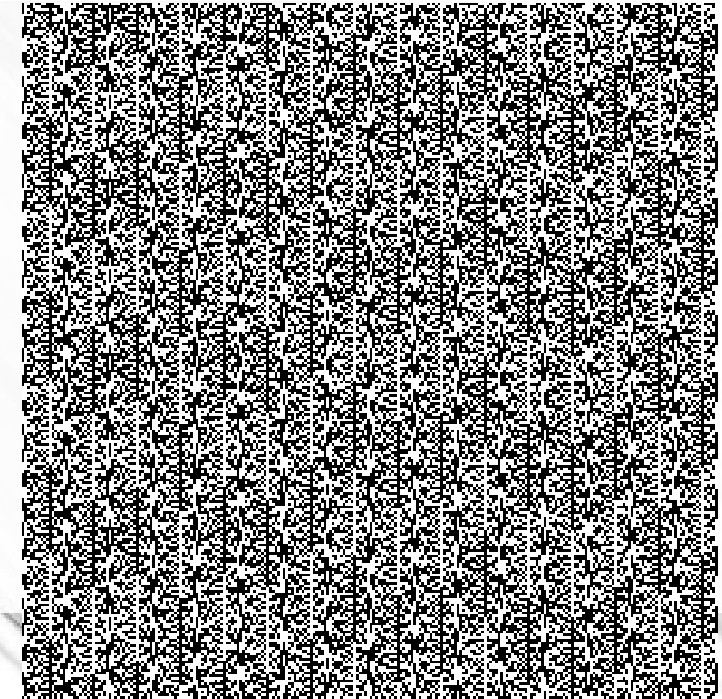
- We have : LCG, Mersenne Twister, Park-Miller, Marsaglia...
- Reason for so many types: RNG quality
- There are ways of getting true random on computers... Quantis RNG hardware (~$4000)
  - Slow, expensive...

There is no random number...

- Only random sequences of numbers.
- A random sequence of numbers have properties:
  - Uniform (numbers equally likely)
  - Independent values (not possible to predict)
  - Summation (sum of 2 numbers should be equally likely to be even or odd)
  - Duplication (some values can be repeated)

How random is a RNG?





How random is a RNG?

- “Diehard tests”
  - Lots of odd tests: “monkey tests”, “count the 1s”, “squeeze test”, “parking lot test”, etc
- NIST (Nat. Inst. Standards & Tech) tests
  - Statistical tests – Frequency test, Random binary matrix rank test, spectral test, etc

Practicality: Ranges

- We can have a random number between

  0 and m = 2147483648

  Does this include 0?

  Does this include 2147483648?
- No. For the LCG, it includes all except 2147483648
- Write this down as [0,2147483648)

Other ranges

- Between 0 and 1 including 0 and 1

  [0,1] 0 <= x <= 1 (a closed interval)

- Between 23 and 39, including 39, excluding 23

  (23,39] 23 < x <= 39 (a half-closed interval)

- Between 10 and 20 excluding 10 and 20

  (10,20) 10 < x < 20 (an open interval)

Other ranges

- Assume a random number **r = 174** generated in [0,12345] and RAND_MAX = 12345
- Want a float in [0,1)

  (r / (RAND_MAX+1.0f))

- Want an int in [5, 10]

  (r % (10-5) + 5)

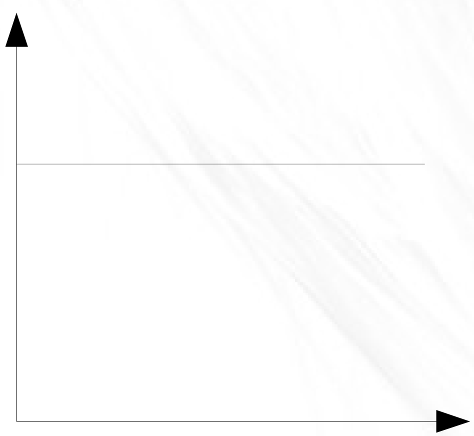
- Sometimes the safest is to get a good random float, and apply that to whatever range you need.

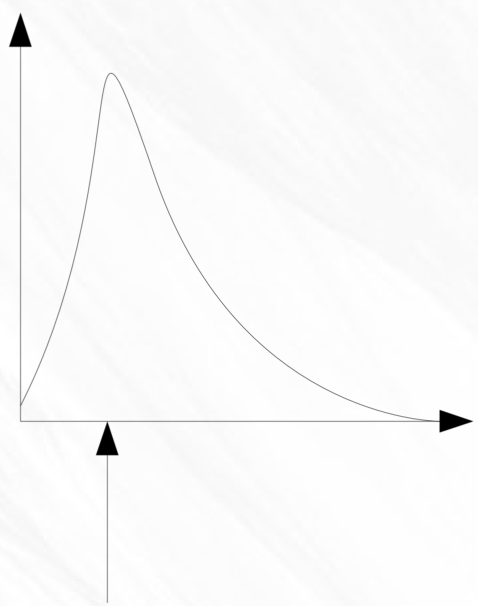
Distributions

- Discrete distributions
  - 1, 9, 3, 1, 4
- Continuous distributions
  - 0.2, 0.49, 0.11, 23.245

Other distributions

- We can have discrete and continuous
- But we can also have uniform and non-uniform





These are far more likely

Exponential Distribution

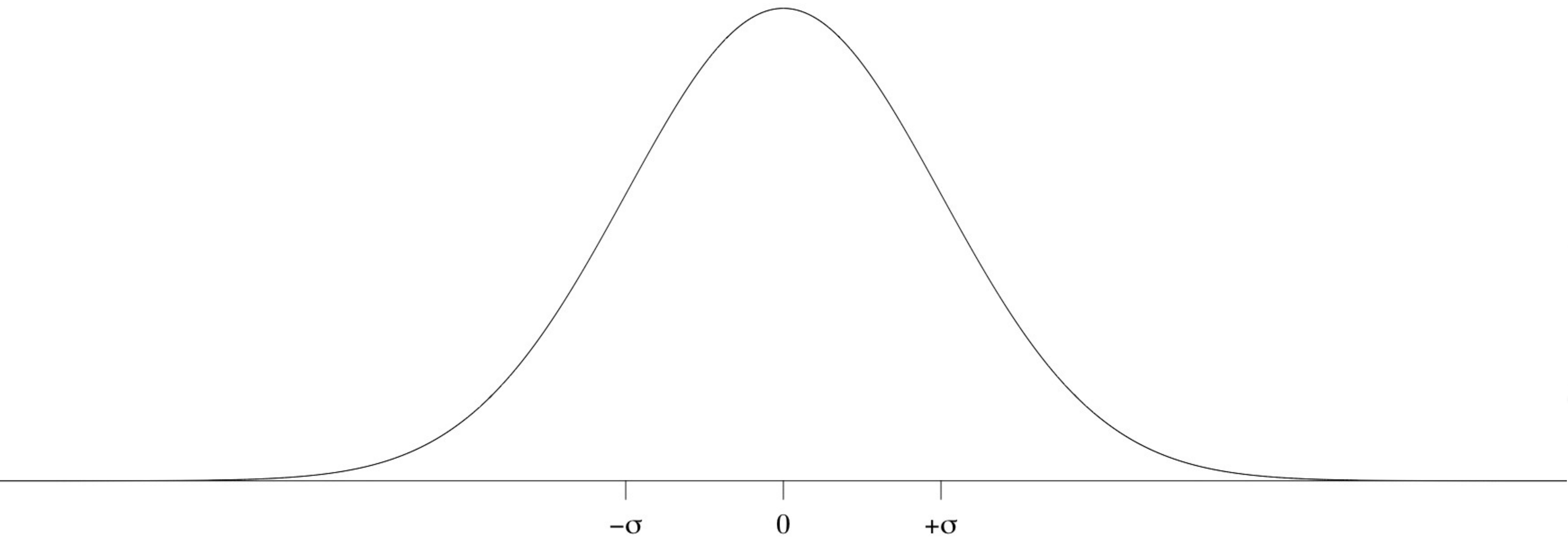
Generating Exponential Deviates

- Easy

$$n = \ln(a)/r$$

- n is our exponential random deviate
- a is a uniform random number
- r is the decay rate

Normal Distribution



Normal Distribution

Generating Normal Deviates

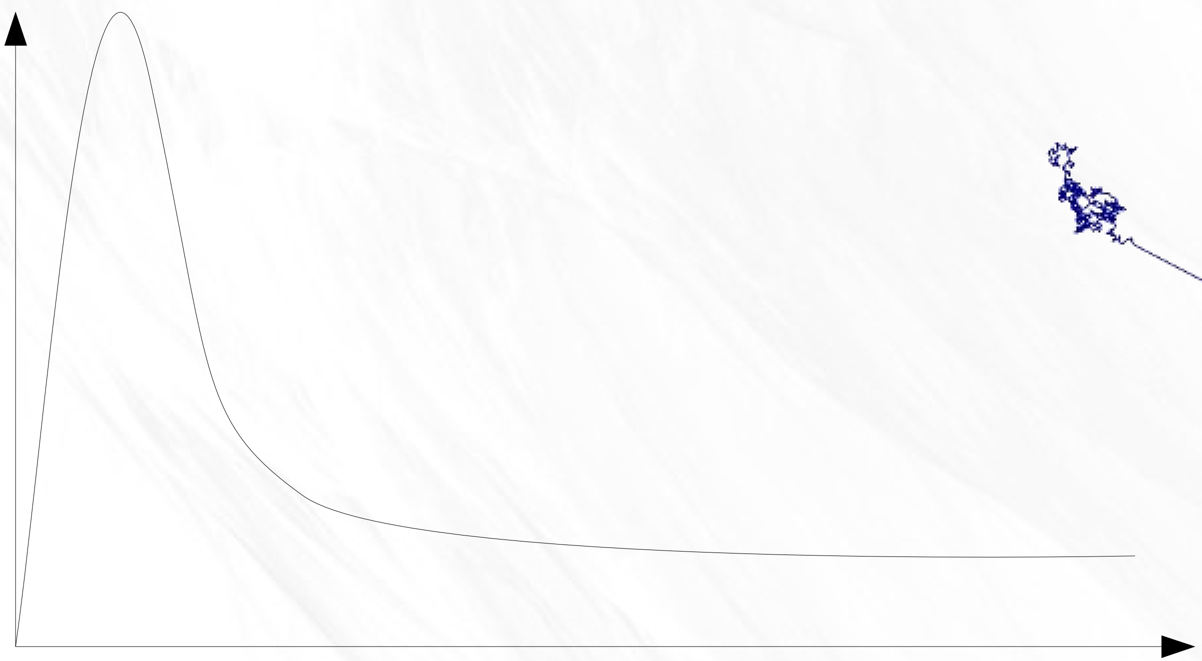
- Use the Box-Muller transform

$$n_1 = (\sigma\sqrt{-2\ln(x)})\sin(y) + \mu$$

$$n_2 = (\sigma\sqrt{-2\ln(x)})\cos(y) + \mu$$

- x = [0,2pi) and y = [0,1) are uniform randoms
- You choose μ and σ (mean and std. dev)

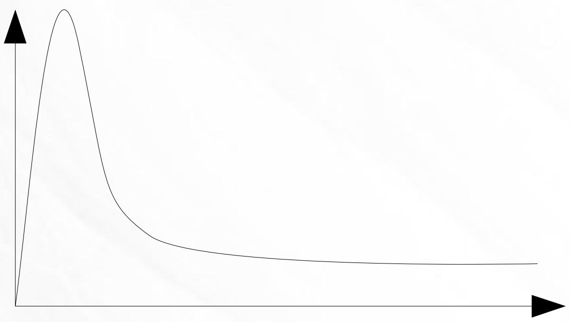
Stable random deviates



Golden Eagles

Generating Levy Flights

- Random deviates that follow this distribution



- The algorithm for generating stable random deviates such as Levy is very complex
- An approximation – Exponential!

Generating Levy Flights

- Take a continuous random number **r in [0,1)**
- And compute -ln(r) / 0.3

Programming an RNG

```
unsigned int mw, mz;  // must be global
int main() {
  float f;
  // load mw and mz - these two numbers make up the seed
  mw = 35;
  mz = 478;
  f = (float) GetUniform();
  printf("%1.2f ", f);
}

unsigned int GetUint() {
  mz = 36969 * (mz & 65535) + (mz >> 16);
  mw = 18000 * (mw & 65535) + (mw >> 16);
  return (mz << 16) + mw;
}



double GetUniform() {
// returns a double in the open interval (0, 1)
  unsigned int u;
  u = GetUint();
  return (u + 1.0) * 2.328306435454494e-10;
}
```