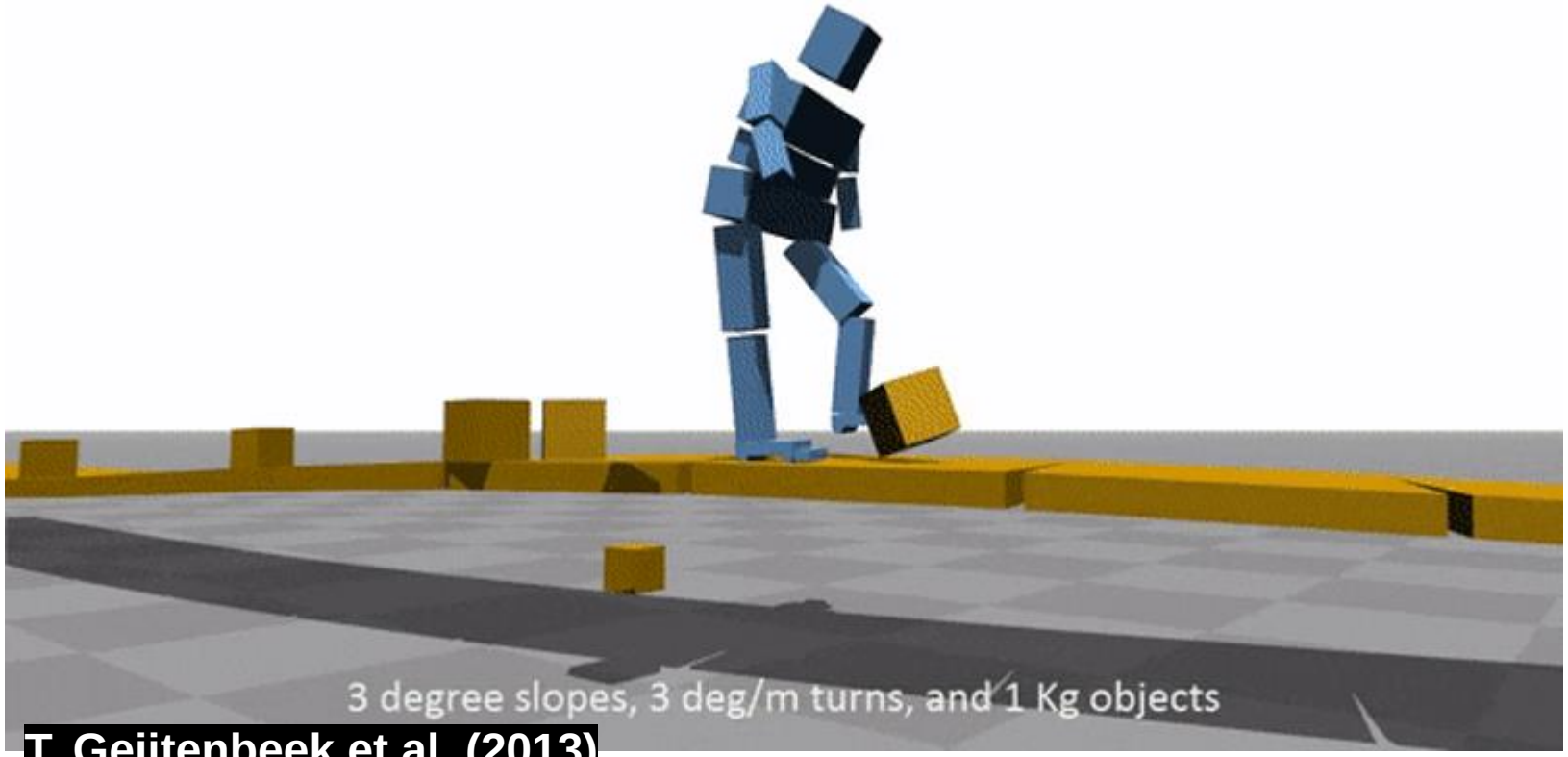
University of St Andrews

Things every computational physicist should be aware of…



T. Geijtenbeek et al. (2013)

Lewis McMillan
lm959@st-andrews.ac.uk
github.com/lewisfish

Introduction

- I’m not an expert on any of these topics
- Not an exhaustive list by any means

Talk Contents

- Binary
- Floating-Point numbers
- Random Numbers
- Various Algorithms

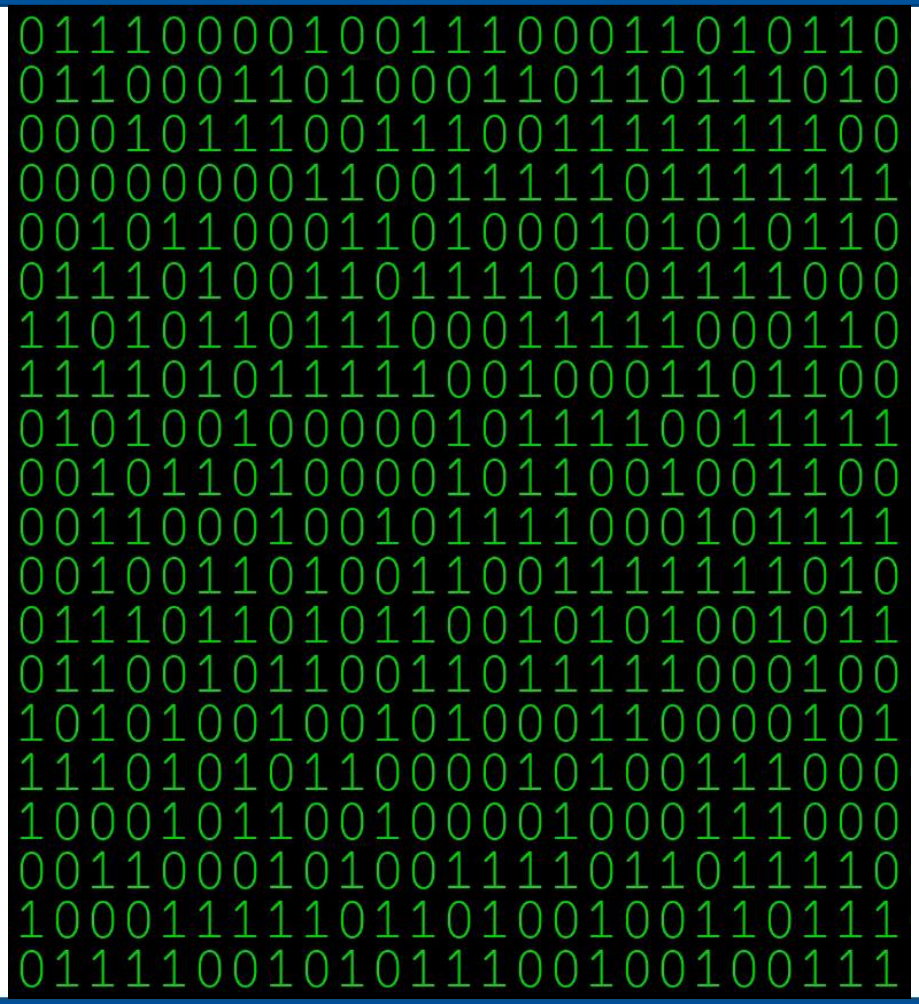
Binary

- Representation of numbers in base 2
- Used in almost every computer due to “on/off”

- Represent the number 30 in base 2

100 10 1
0 3 0 = 10 + 10 +10 = 30

32 16 8 4 2 1
0 1 1 1 1 0 = 2 + 4 + 8 + 16 = 30



Binary

- Represent the number 30 in base 2

  100 10 1
  0 3 0 = 10 + 10 +10 = 30

  32 16 8 4 2 1
  0 1 1 1 1 0 = 2 + 4 + 8 + 16 = 30

  011110b = 30

- 1 bit is either 0 or 1
- 1 byte is 8 bits

Floating-point numbers

- Floating-point numbers should we be worried?
- Maybe...
- 0.1 + 0.2 = 0.30000000000000004
- Really depends on what you are doing

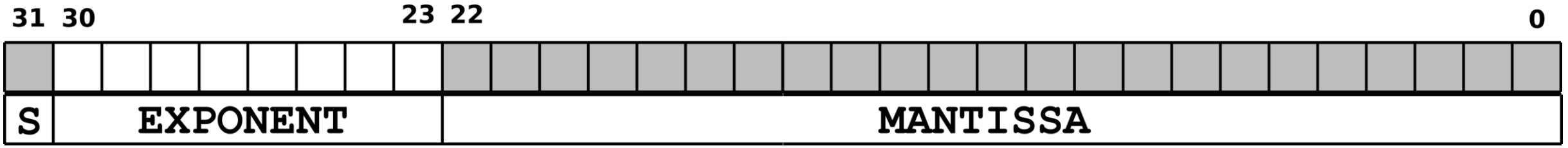
www.gao.gov/assets/220/215614.pdf

(Photo by Anthony Sweeney/U.S. Army Europe)

Floating-point numbers

- What are they?
- S = sign
- M = mantissa
- E = exponent
- 32 bits used to represent numbers in IEEE 754 standard

$$(-1)^{S} * 1.M * 2^{(E-127)}$$



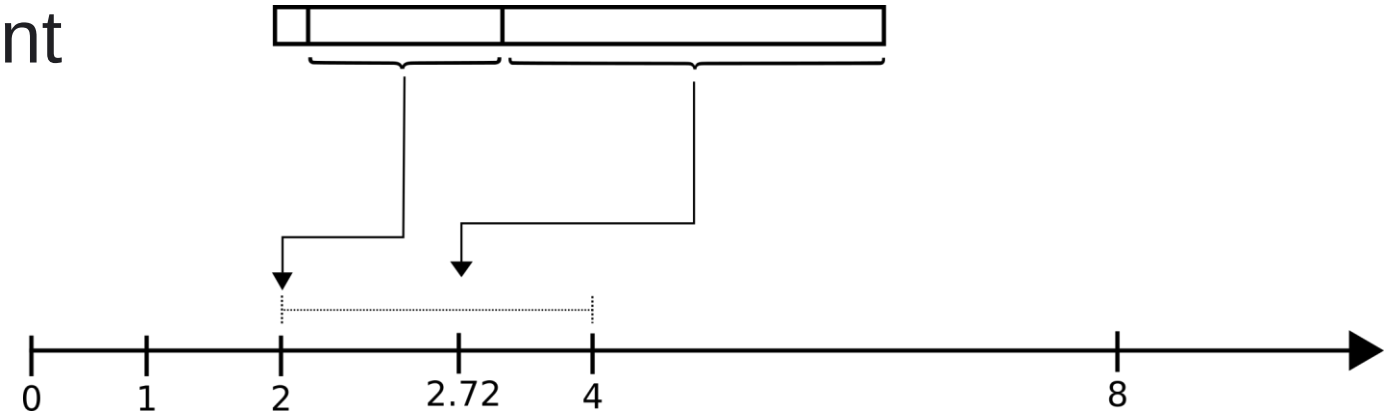
F. Sanglard (2017)

Floating-point numbers

- S = sign
- M = mantissa
- E = exponent

$$(-1)^S * 1.M * 2^{(E-127)}$$

- Represent *e*=2.72 in floating-point
- S = 0
- E = 128
- M = $\frac{2.72-2}{4-2}$ =0.36



S	WINDOW	OFFSET
S	EXPONENT	MANTISSA

F. Sanglard (2017)

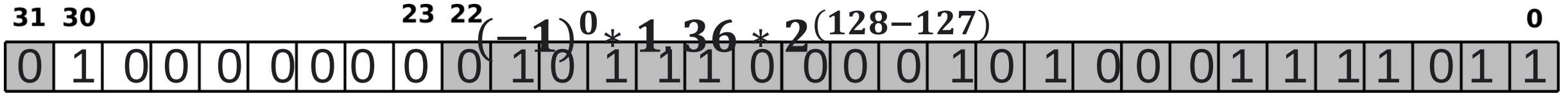
Floating-point numbers

- S = sign
- M = mantissa
- E = exponent

$$(-1)^S * 1.M * 2^{(E-127)}$$

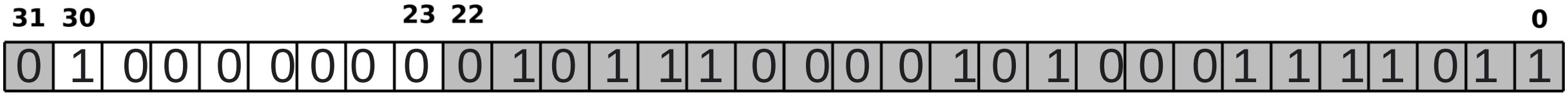
- Represent *e*=2.72 in floating-point
- S = 0
- E = 128
- M = $\frac{2.72-2}{4-2} = 0.36 * 2^{23}$

$$(-1)^0 * 1.36 * 2^{(128-127)}$$



Floating-point numbers

- Main problem in floating-point representation
- Can’t exactly represent all real numbers!



2.72 != 2.720000028610229492187 5

Floating-point numbers

- Muller's recurrence

$$f(y,z) = 108 - \frac{815 - {}^{1500}/_{z}}{y}$$

$$x_i = f(x_{i-1}, x_{i-2}) \quad x_0 = 4 \quad x_1 = 4.25$$

FORTRAN

i	4bytes
0	4.00000000
1	4.25000000
2	4.47058868
3	4.64474487
4	4.77070618
5	4.85921478
6	4.98312378
7	6.39543152
8	27.6326294
9	86.9937592
10	99.2555084
11	99.9625854
12	99.9981308
13	99.9999084
14	100.000000
15	100.000000
16	100.000000
17	100.000000
18	100.000000
19	100.000000
20	100.000000
21	100.000000
22	100.000000
23	100.000000
24	100.000000

Floating-point numbers

- Can’t exactly represent all real numbers!
- Errors can accumulate

- Multiplication and division are “safe”
- Addition and subtraction can be dangerous

- Loss of digits can be ok or catastrophic
- The more calculations are done, i.e. iterative algorithms, the more danger there usually is

Floating-point numbers

If you really want to know more…

https://floating-point-gui.de

https://randomascii.wordpress.com/category/floating-point

https://doi.org/10.1145/103162.103163

What Every Computer Scientist Should Know About Floating-Point Arithmetic

DAVID GOLDBERG

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304

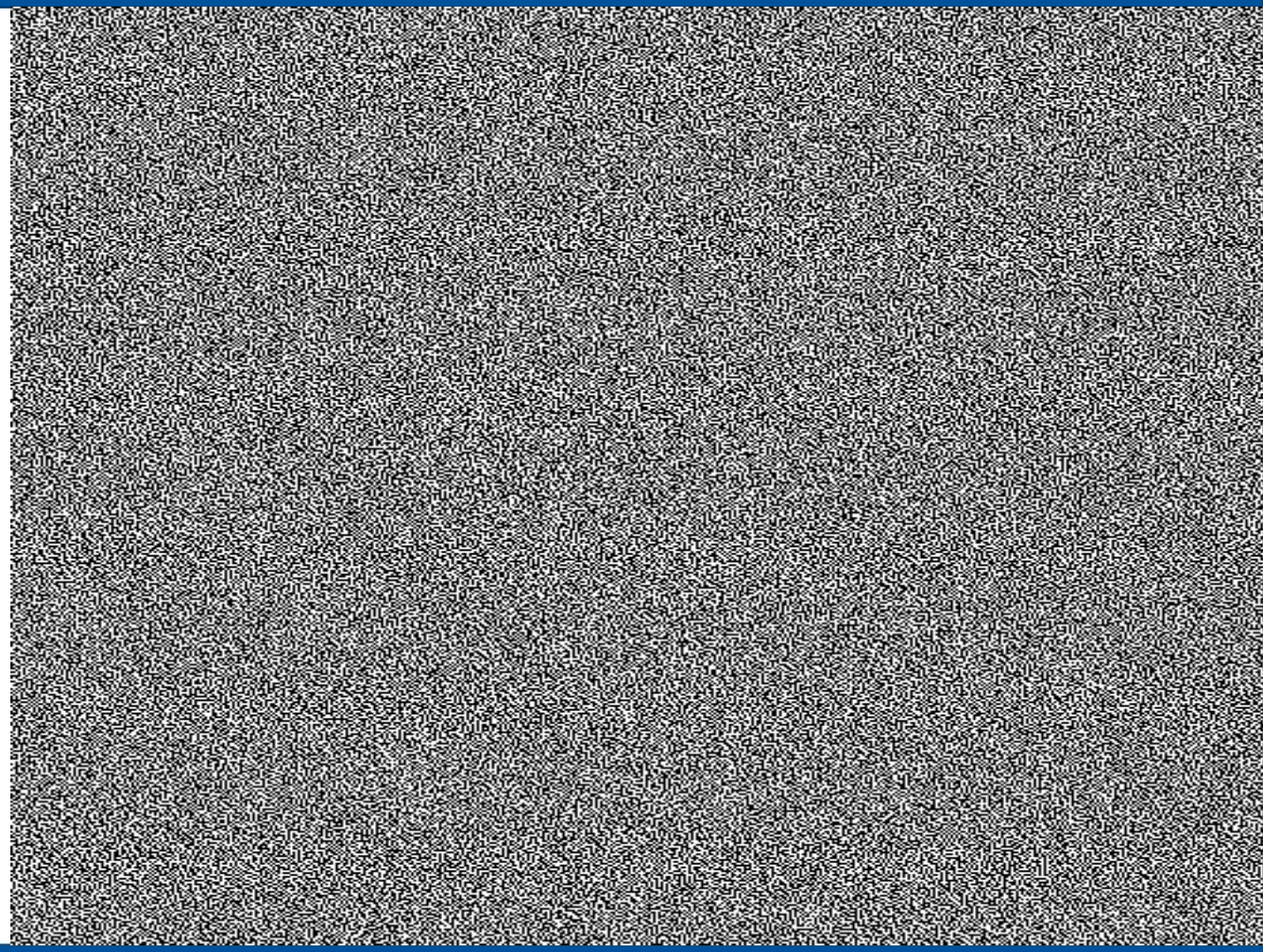
Floating-point arithmetic is considered an esoteric subject by many people. This is rather surprising, because floating-point is ubiquitous in computer systems: Almost every language has a floating-point datatype; computers from PCs to supercomputers have floating-point accelerators; most compilers will be called upon to compile floating-point algorithms from time to time; and virtually every operating system must respond to floating-point exceptions such as overflow This paper presents a tutorial on the aspects of floating-point that have a direct impact on designers of computer systems. It begins with background on floating-point representation and rounding error, continues with a discussion of the IEEE floating-point standard, and concludes with examples of how computer system builders can better support floating point.

Categories and Subject Descriptors: (Primary) C.0 [**Computer Systems Organization**]:

Random numbers

- Is the random number generator I use “random” enough??

- Maybe
- So long as you use a generator that passes “basic” tests *should* be ok
- Depends on application



Random numbers

- A good RNG is fast, has a long period and passes various tests
  - **Linear Congruent Generator (LCG)**

$$y_{i+1} = mod(ay_i + c, M)$$

  - Choose a, c and M
  - Can make “good” and “bad” choices

Random numbers

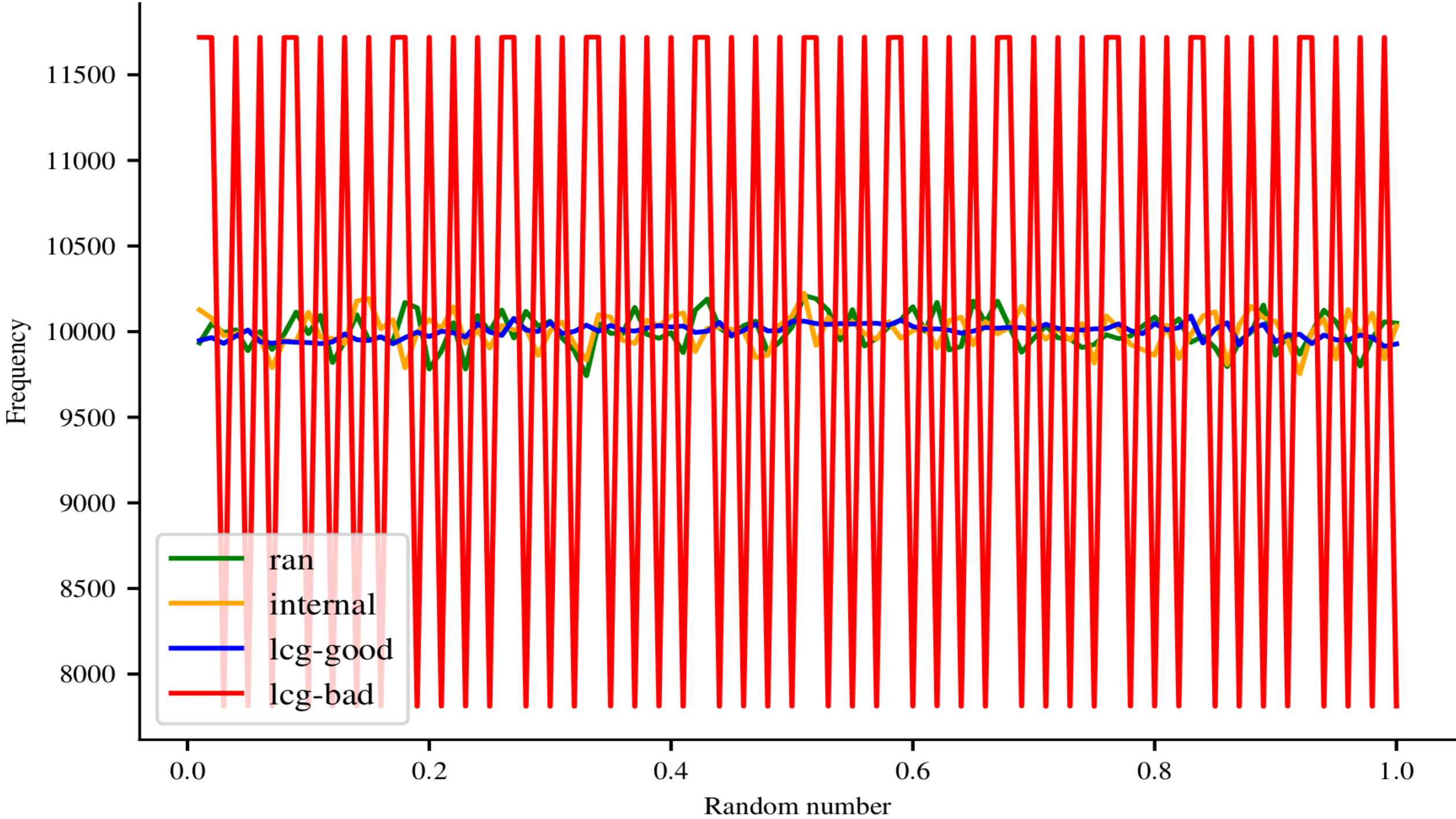
- A good RNG is fast, has a long period and passes various tests

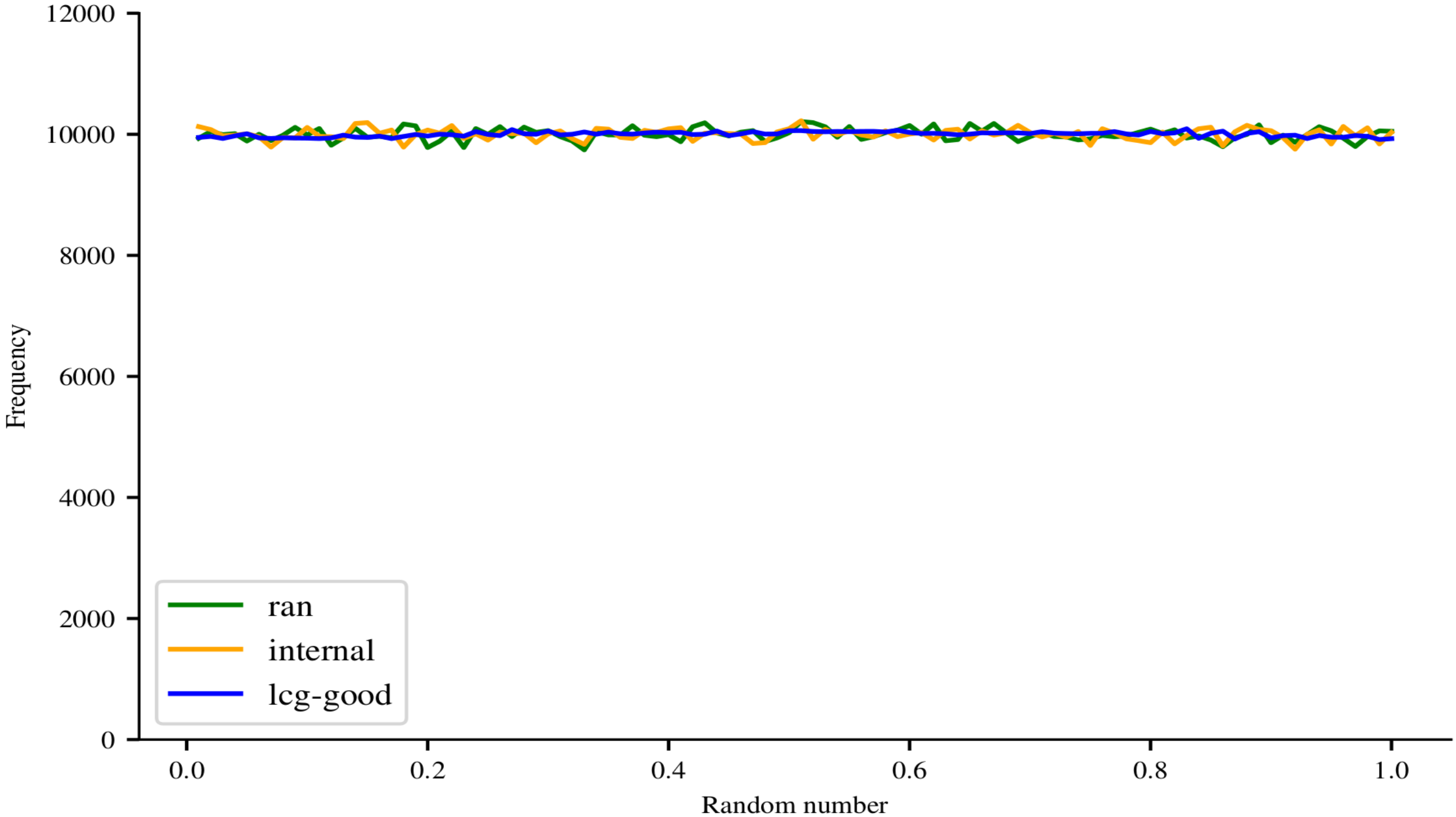
  - **Ran**
  - Numerical recipe's own RNG
  - "Minimal random number generator of Park and Miller combined with a Marsaglia shift sequence"

Random numbers

- A good RNG is fast, has a long period and passes various tests

  - **Fortran's internal**
  - No defined algorithm
  - Gfortran: xorshift1024*


11500
11000
10500
10000
9500
9000
8500
8000
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
Random number
ran
internal
lcg-good
lcg-bad

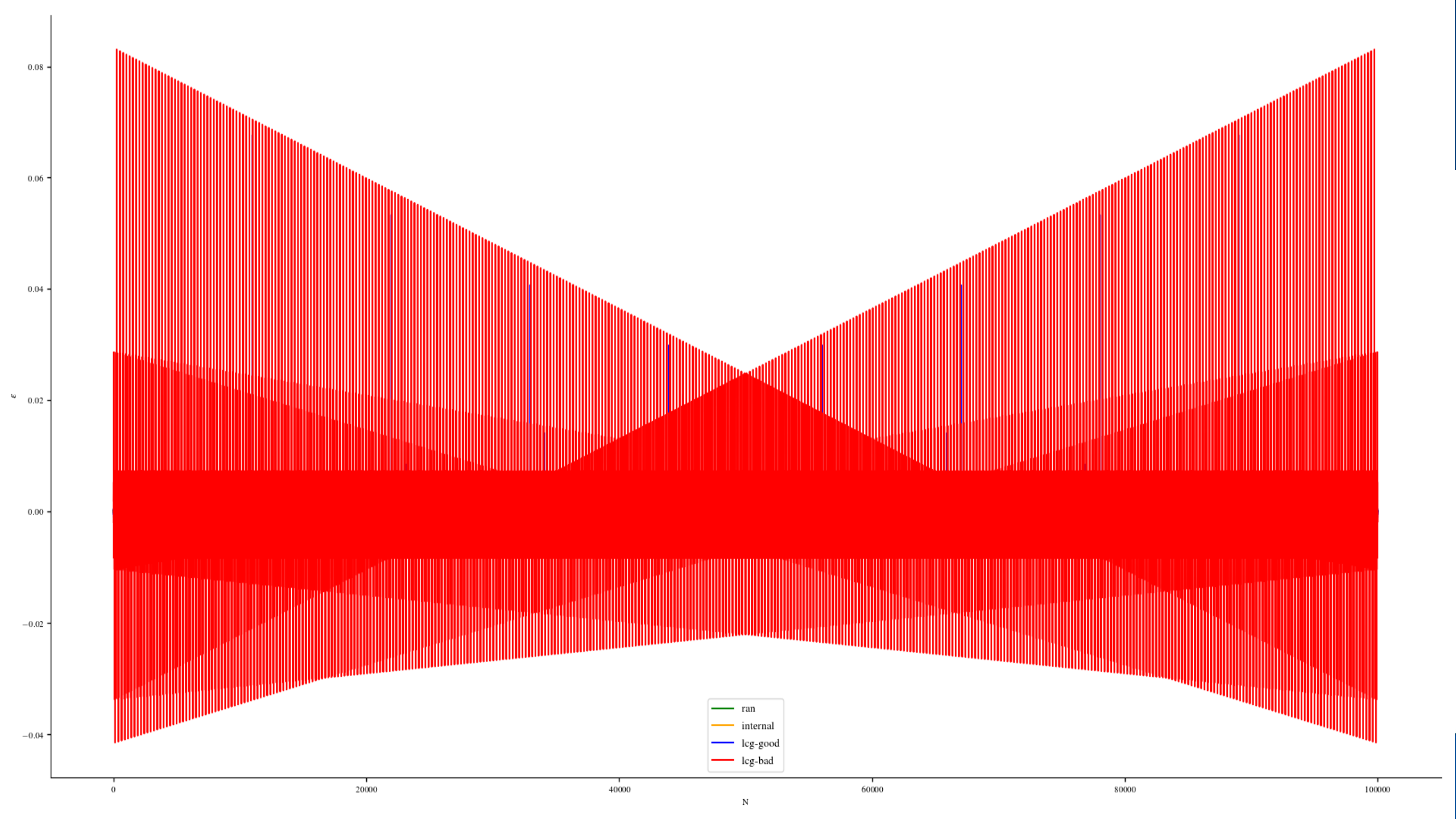

12000
10000
8000
6000
4000
2000
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
Random number
ran
internal
lcg-good

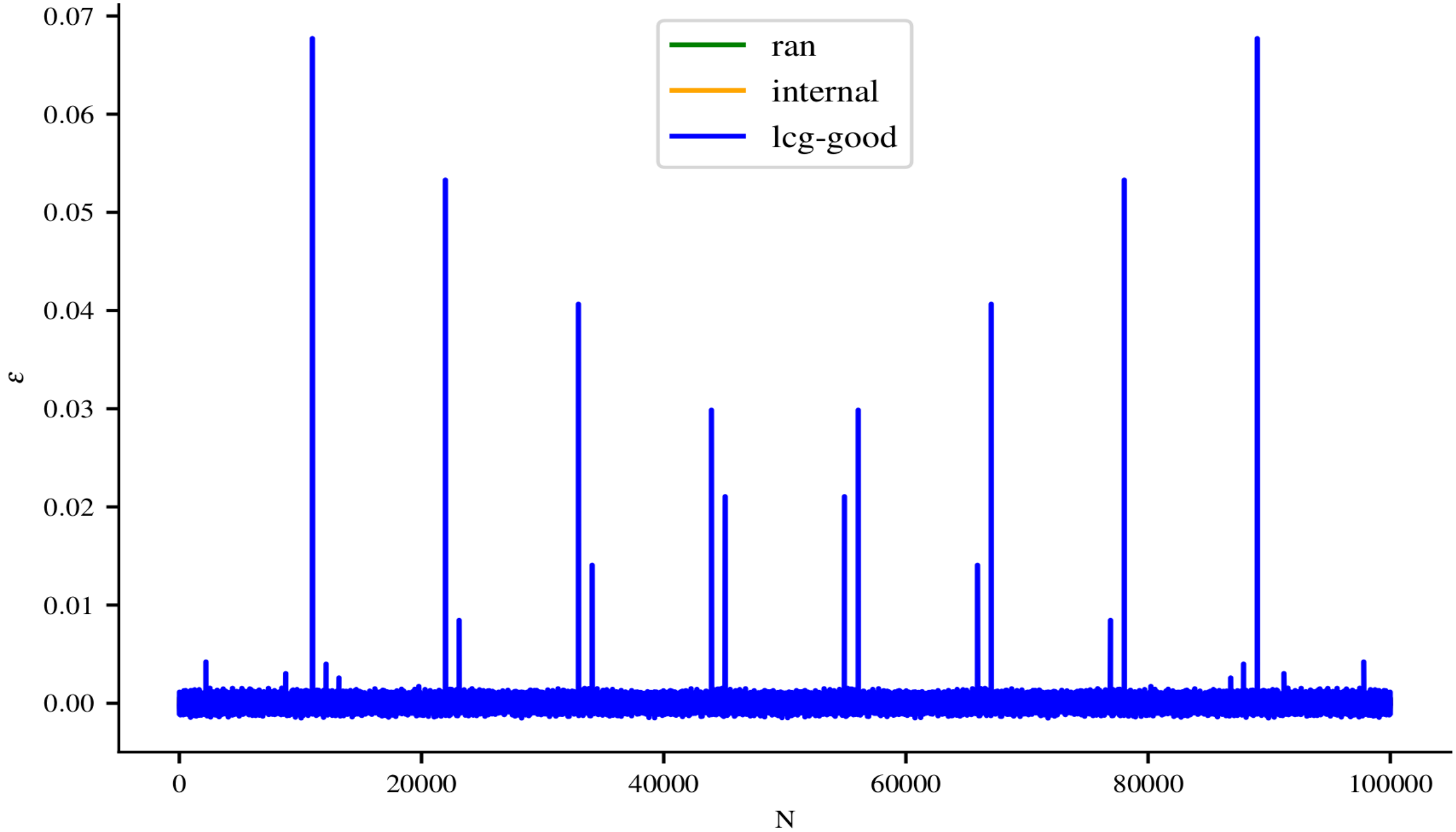
Random numbers

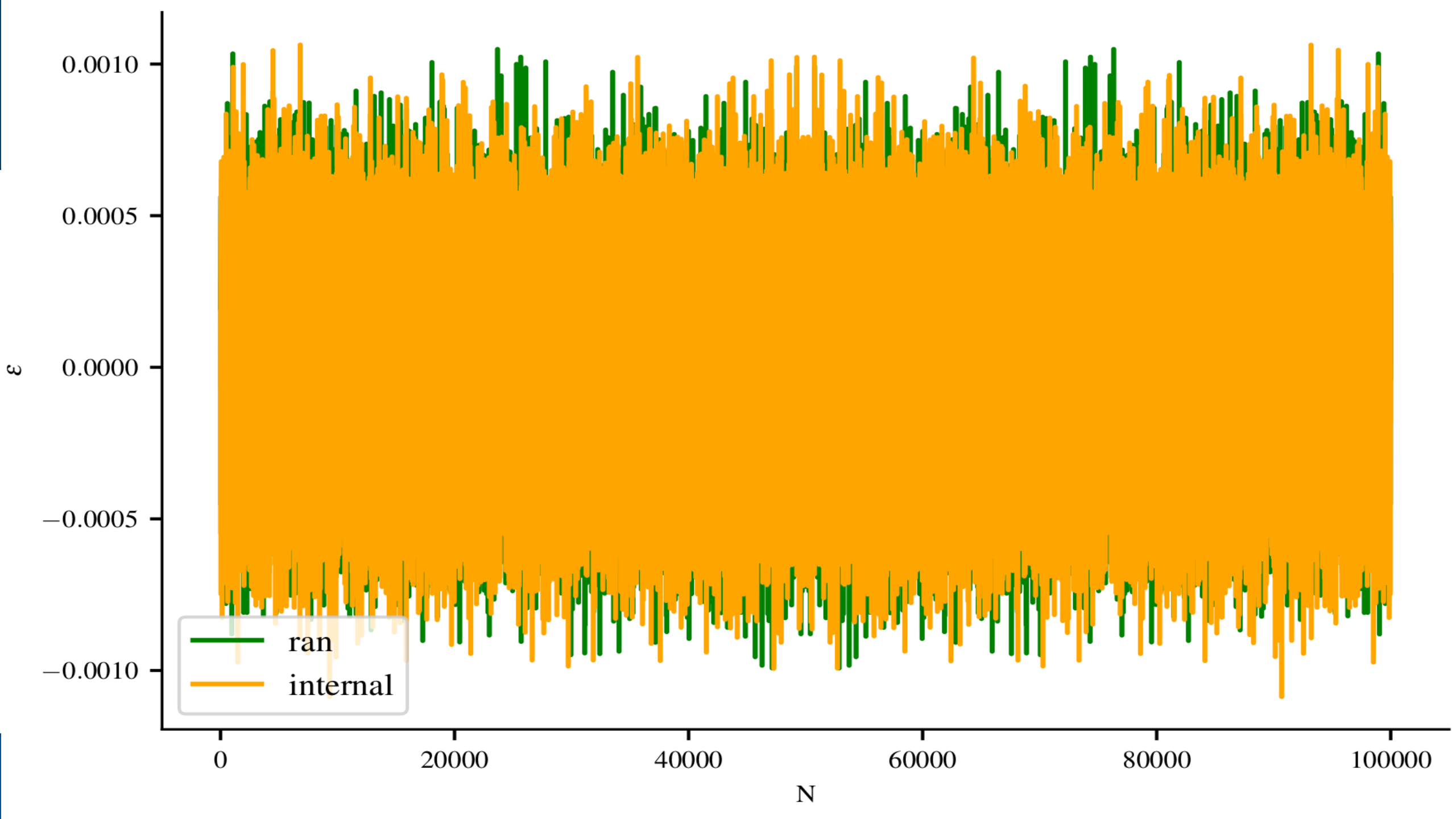
- Simple test: test correlation of pairs of numbers

$$\varepsilon(N,n) = \frac{1}{N}\sum_{i=1}^{N} x_i x_{i+n} - E(x)^2 \qquad n \in \mathbb{N}$$

- LCG
- Ran2
- Fortran's internal


0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
ε
0
20000
40000
60000
80000
100000
N
ran
internal
lcg-good
lcg-bad


ran
internal
lcg-good
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
ε
0
20000
40000
60000
80000
100000
N


0.0010
0.0005
0.0000
−0.0005
−0.0010
ε
0
20000
40000
60000
80000
100000
N
ran
internal

Random numbers

- From ‘Numerical Recipes’
  - Never use a LCG
  - Never use a RNG with period < 2^{64}
  - Never use a RNG that warns against using low-order bits

Random numbers

- Don’t use via http://www0.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf
  - Perl’s standards RNG
  - Python’s random() before v2.3
  - Java.util.Random
  - C-library rand(), random() or drand48()
  - Matlab’s rand
  - Mathematica’s SWB generator
  - ran0() or ran1() from ‘Numerical Recipes’

Random numbers

- Recommended by H.Katzgraber www.arxiv.org/abs/1005.4117
- Use WELL generators, Mersenne twister or multiplicative lagged Fibonacci generators
- Don’t use UNIX inbuilt or any of ‘Numerical Recipes’ RNG

Random numbers

- Further reading:
  - https://link.springer.com/chapter/10.1007/978-3-642-21551-3_3
  - http://www0.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf
  - https://arxiv.org/pdf/1005.4117.pdf
  - Numerical Recipes / W. Press, S. Teukolsky, W. Vetterling, and B. Flannery

Algorithms

- Brief tour of:
  - Monte Carlo Method
  - Finite difference method
  - Bounding volume hierarchy

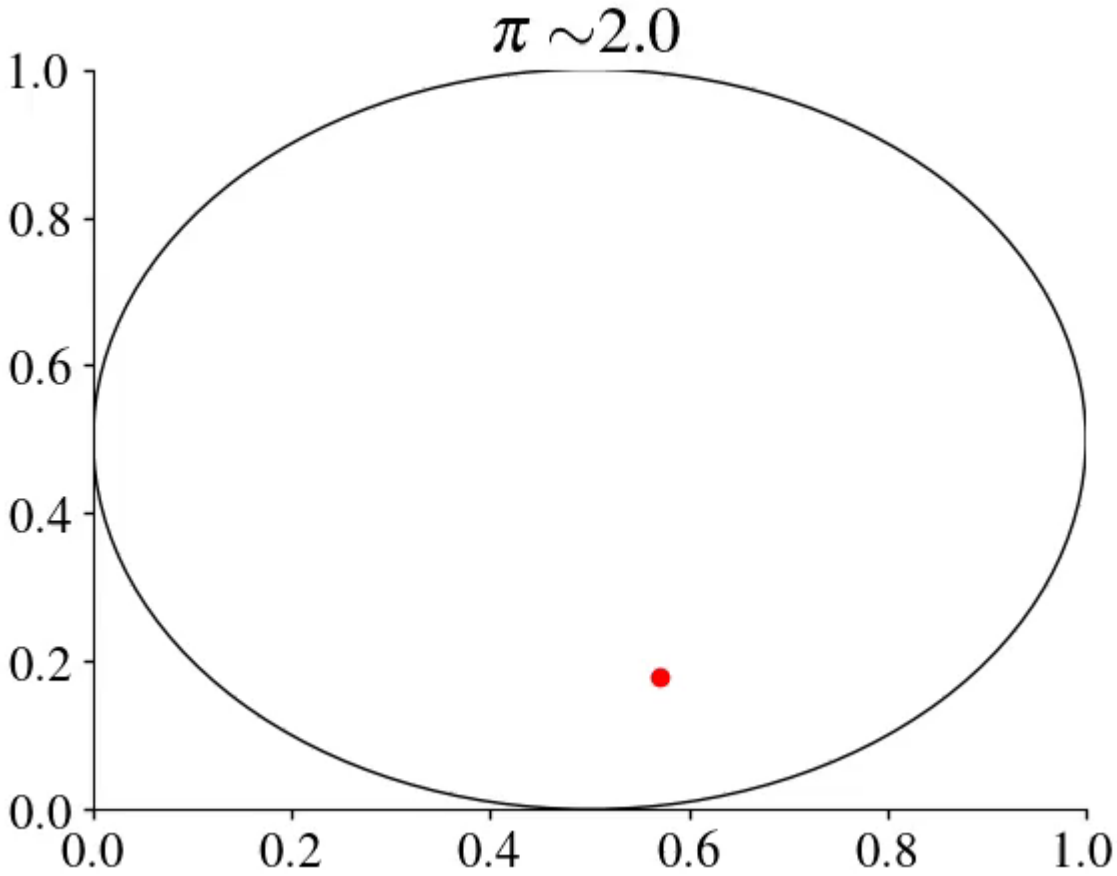
```
if(size(neighbours) == 0)then
do while(size(neighbours) == 0) ; x = this%x%pop() ; y = this%y%pop() ; if(this%x%empty())return ; deallocate(neighbours) ; call this%get_neigh(y, x, neighbours) ; end do
neighbour = irand(1, size(neighbours)) ; end if ; neighbour = neighbours(neighbour)
if(neighbour == 1)then ; if(this%vis(y,x+1) /= 1)then ; this%cell(y,x+1)%left = 1 ; deallocate(neighbours) ; call this%gen_maze(y, x+1) ;end if
elseif(neighbour == 2)then ; if(this%vis(y+1,x) /= 1)then ;  this%cell(y+1,x)%up = 1 ; deallocate(neighbours) ; call this%gen_maze(y+1, x) ; end if
elseif(neighbour == 3)then ; if(this%vis(y,x-1) /= 1)then ; this%cell(y,x)%left = 1 ; deallocate(neighbours) ; call this%gen_maze(y, x-1) ; end if
elseif(neighbour == 4)then ; if(this%vis(y-1,x) /= 1)then ; this%cell(y,x)%up = 1 ; deallocate(neighbours) ; call this%gen_maze(y-1, x) ; end if
```

Algorithms – Monte Carlo Method

- What?
  - Random sampling to obtain numerical results
- Why?
  - Radiation transport, statistical physics
  - AI, CGI
- How?
  - Simplest example: estimation of π

Algorithms – Monte Carlo Method

- How?
  - Simplest example: estimation of π
  - Throw darts at board
  - Score darts that hit in circle
  - π~4*Nh/N



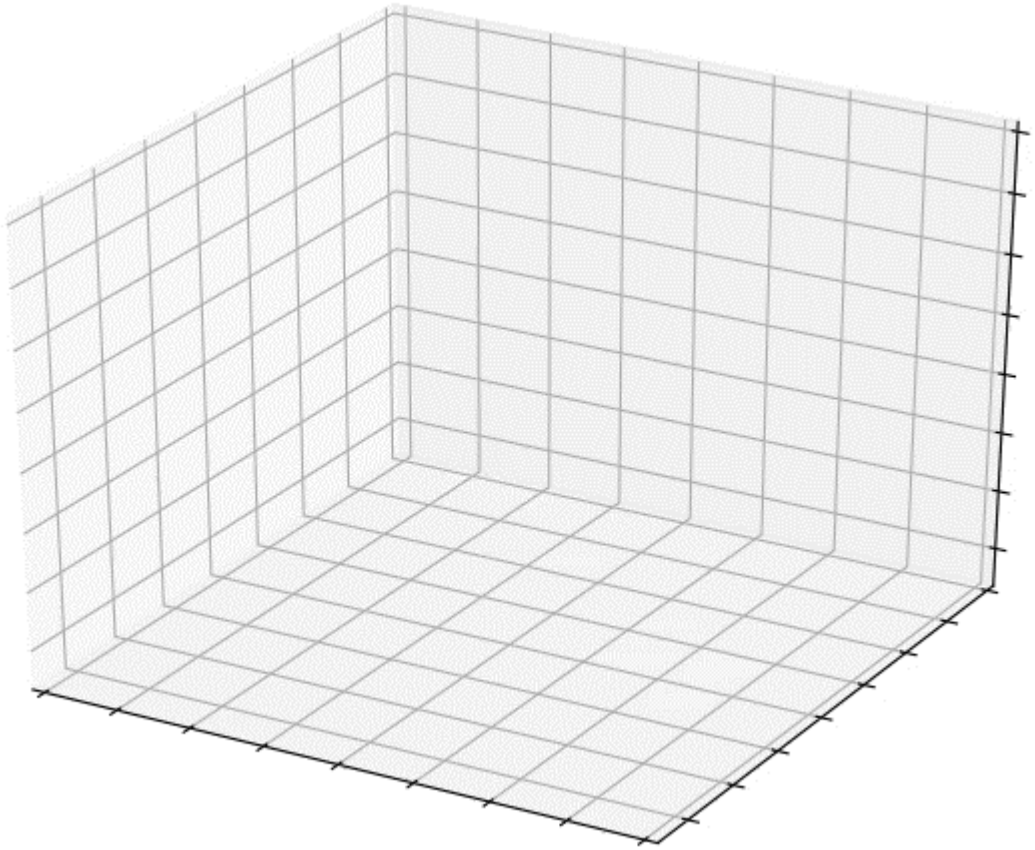
Algorithms – Monte Carlo Method

- How?
  - Compute CDF from PDF $CDF(x) = \int_0^x p(x')dx'$
  - Compute inverse of CDF $CDF^{-1}(x)$
  - Obtain random number ξ
  - Compute $X_i = CDF^{-1}(\xi)$

Algorithms – Monte Carlo Method

- How?
  - Photon interaction probability $e^{-\tau}$
  - Random interaction probability

$$= -\ln(\xi)$$



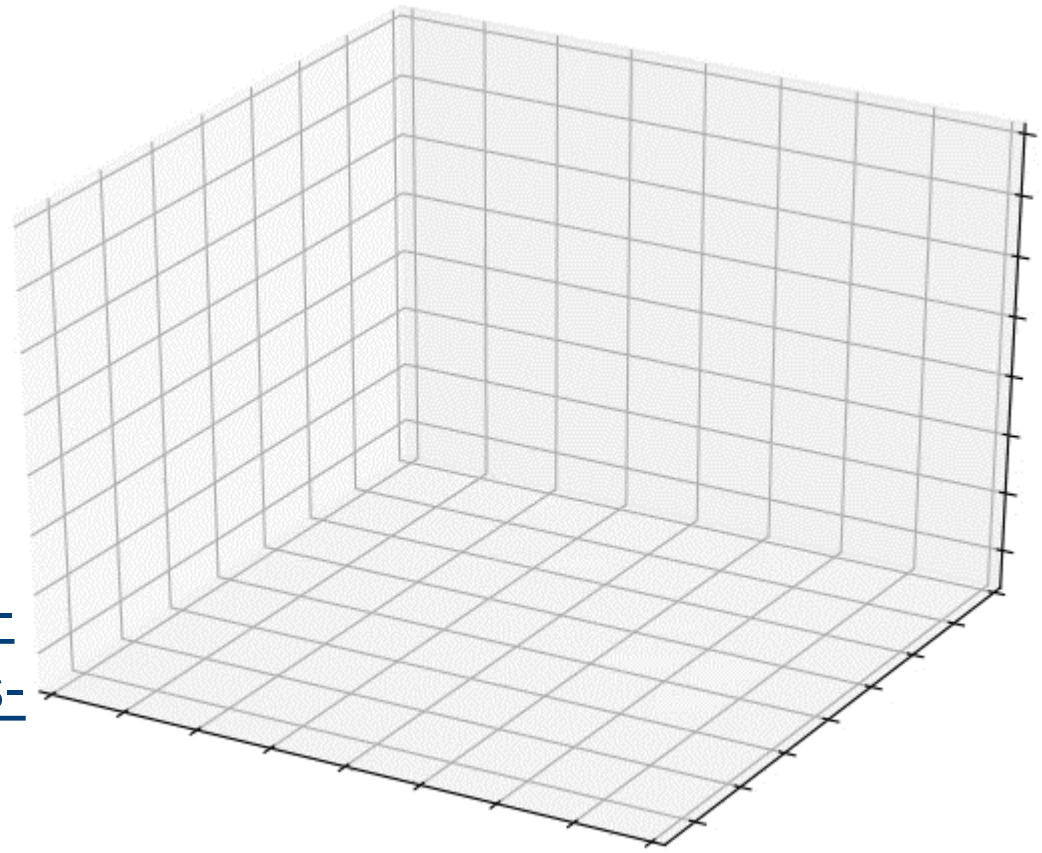
Algorithms – Monte Carlo Method

- Further reading:

http://www.pbr-book.org/3ed-2018/Monte_Carlo_Integration.html

http://www-star.st-and.ac.uk/~kw25/teaching/mcrt/mcrt.html

https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-mathematical-foundations/quick-introduction-to-monte-carlo-methods



Algorithms – Finite difference Method

- What?
  - Numerical method to solve PDE's & ODE's
- Why?
  - Easily adaptable method
- How?
  - Taylor series approximation of derivatives

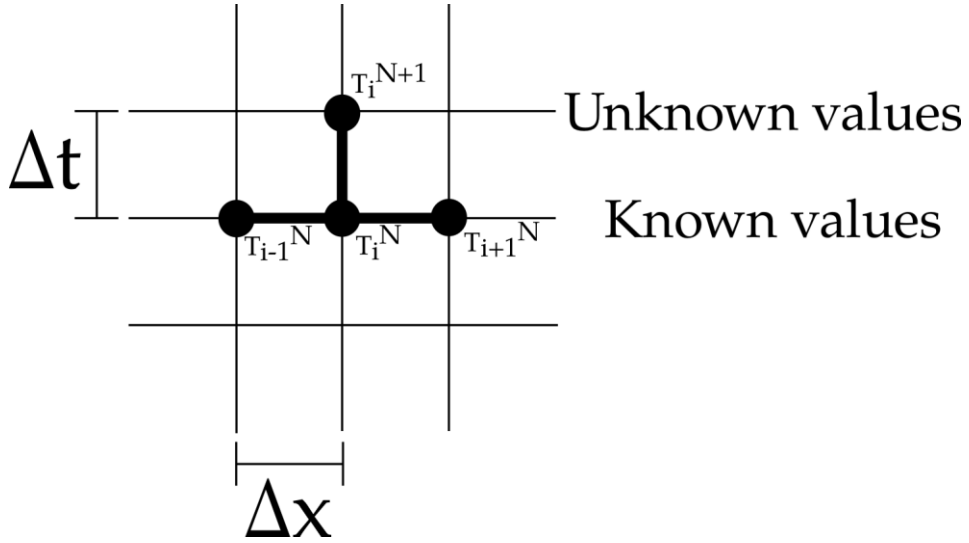
Algorithms – Finite difference Method

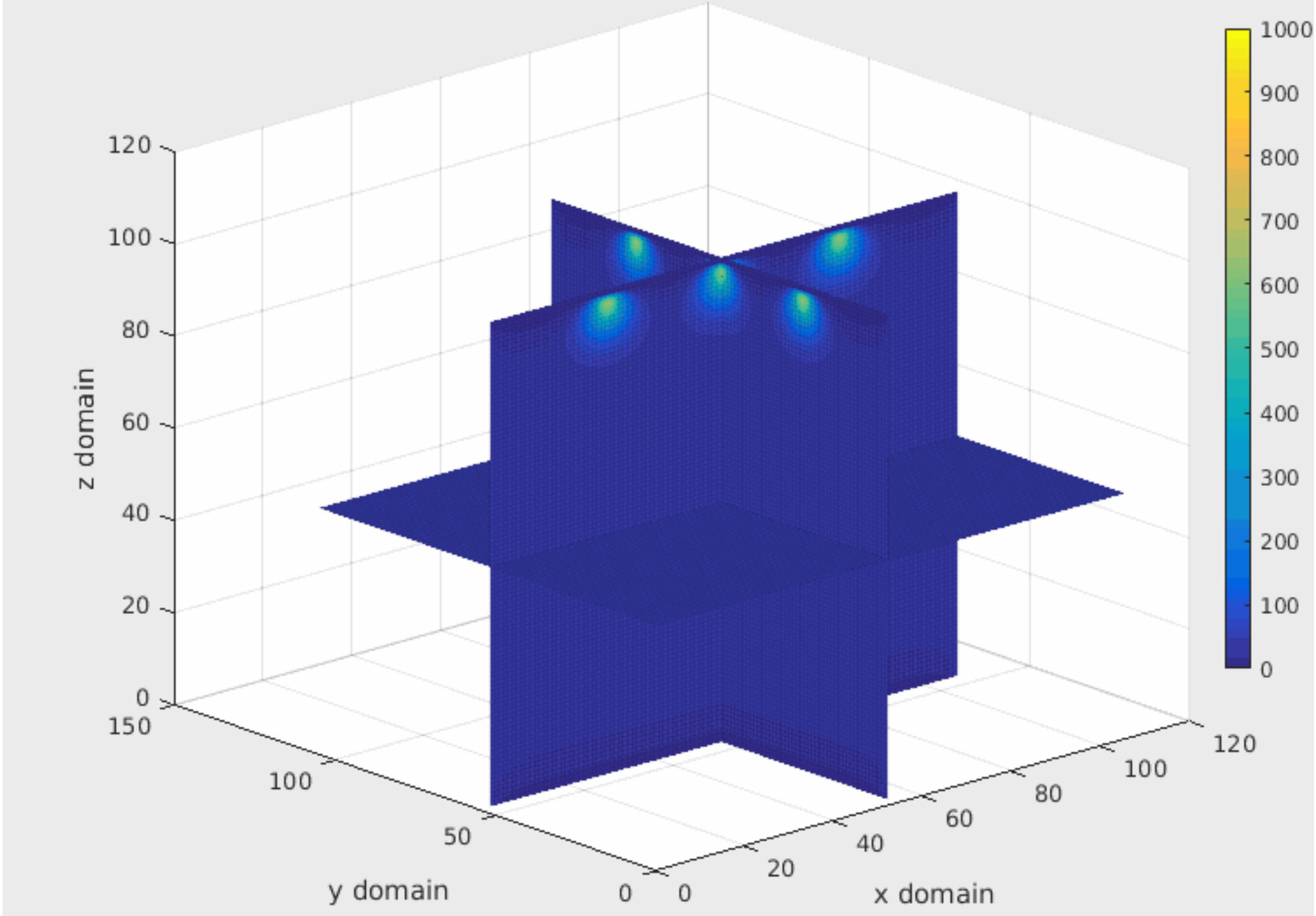
- Solve heat equation $\frac{\partial T}{\partial t} = \alpha \nabla^2 T$
- Forward $\frac{df}{dx} = \frac{f_{i+1} + f_i}{\Delta x}$
- 2nd order central $\frac{d^2 f}{dx^2} = \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta x^2}$

Algorithms – Finite difference Method

- Solve heat equation $\frac{\partial T}{\partial t} = \alpha \nabla^2 T$

$$T_i^{n+1} = T_i^n + \alpha \Delta t \frac{T_{i-1}^n - 2T_i^n + T_{i+1}^n}{\Delta x^2}$$



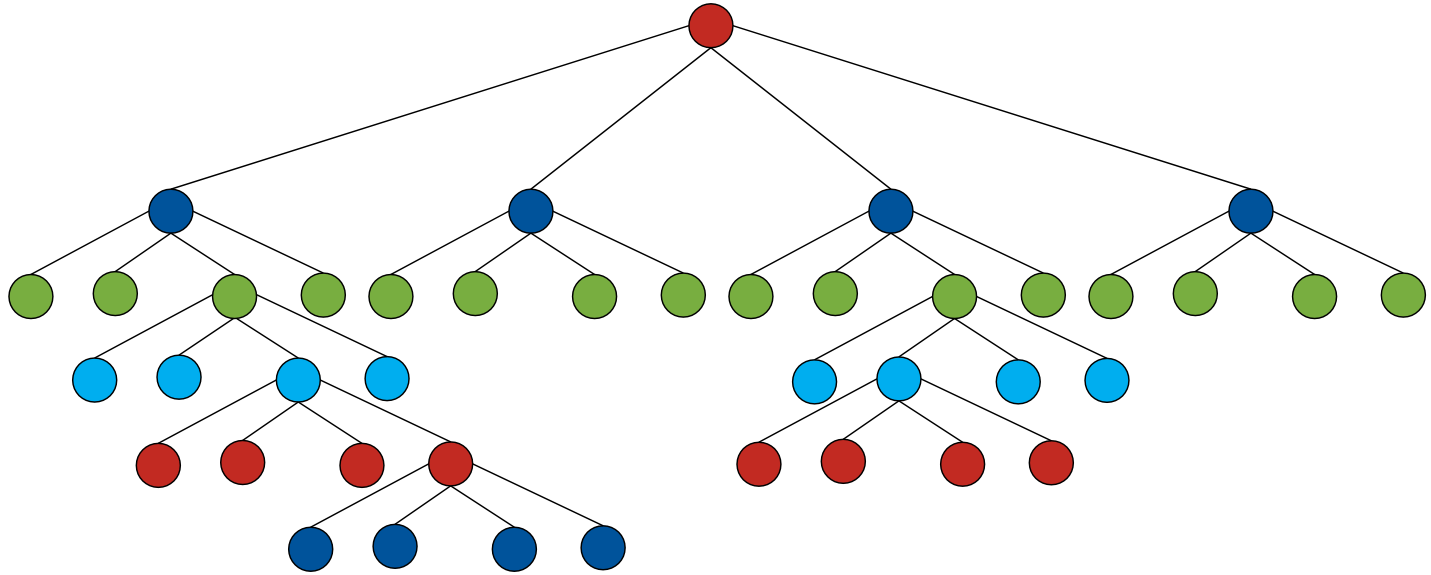

z domain
120
100
80
60
40
20
0
150
100
50
0
y domain
0
20
40
60
80
100
120
x domain
1000
900
800
700
600
500
400
300
200
100
0

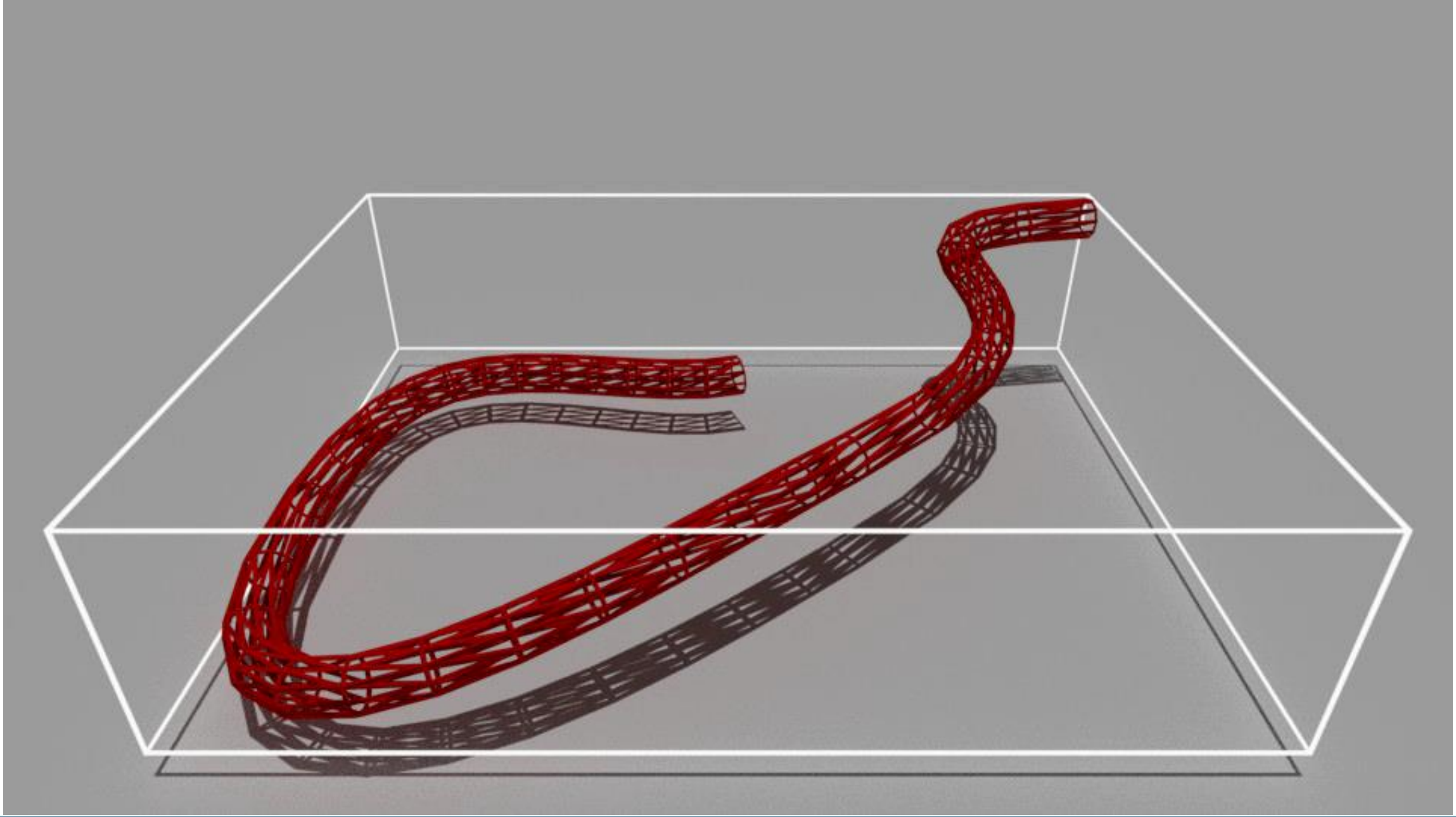
Algorithms – Finite difference Method

- Further reading:
  - Finite difference schemes and partial differential equations / J. Strikwerda
  - Numerical Solution of Partial Differential Equations / D.Mayers and K. W. Morton
  - Finite difference methods in heat transfer / M. Özişik

Algorithms - BVH

- What?
  - Tree based data structures
- Why?
  - Can drastically speed up computations
- How?
  - Partitions space more efficiently

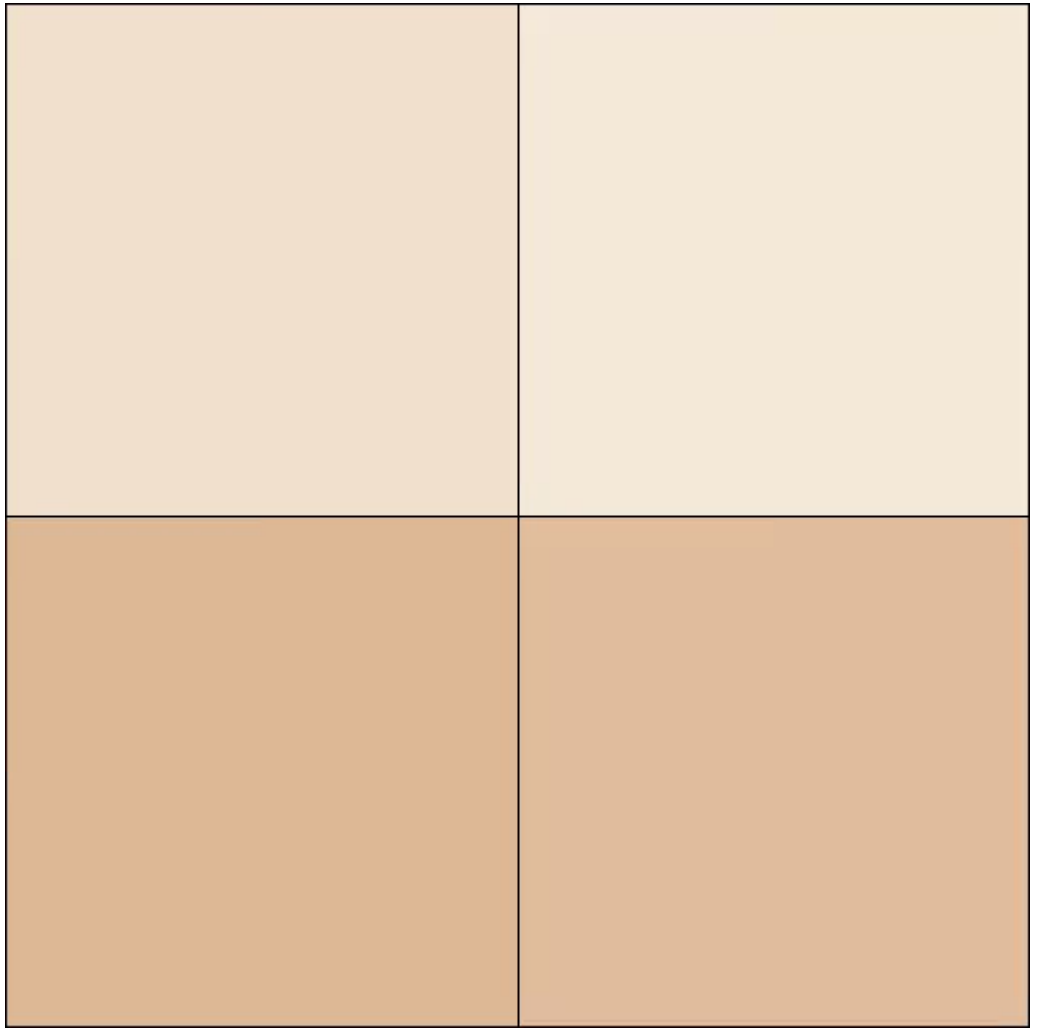




Algorithms - BVH

- Works in 2D and 3D
- Used in image compression

- Used in n-body simulations
- Used in fluid simulations
- Plus many other fields



Algorithms - BVH

- Further reading
- “Physics” based
  - https://doi.org/10.1038/324446a0
  - http://arborjs.org/docs/barnes-hut

- “Computer” based
  - http://www.pbr-book.org/3ed-2018/Primitives_and_Intersection_Acceleration/Further_Reading.html
  - https://www.scratchapixel.com/lessons/advanced-rendering/introduction-acceleration-structure

Questions

- Slides available at: https://code-and-cake.github.io