Computer Simulations in Physics

Course for MSc physics students

Janos Török

Department of Theoretical Physics

February 17, 2022

Information

- Coordinates:
  - Török János
  - Email: torok.janos@ttk.bme.hu, torok72@gmail.com
  - Consultation:
    - F III building, first floor 6 (after the first stairs to the right, at the end of the corridor), Department of Theoretical Physics
    - Teams or Meet
    - Upon demand (Email)
- Webpage:
  http://physics.bme.hu/BMETE15MF74_kov?language=en
- Homework: http://edu.ttk.bme.hu/

Required knowledge

- Knowledge of basic statistical physics
- Knowledge of basic quantum mechanics
- C, C#, C++ or python language
- If you use C# please submit only the code part!

Programming environment

- Basic
  - Editor (not notepad!)
  - Compiler (gcc recommended)
- Advanced
  - Developer environment
  - Integrated developer environment (also compiles)

Integrated developer environment

- Visual studio (old version can be downloaded from `http://software.eik.bme.hu` only from bme.hu domain)
- Eclipse
- Netbeams
- CodeLite
- pyCharm
- Anaconda
- Google colab
- etc.

Install compiler

- Linux
  - Install development package, usually not installed when desktop installation was selected (libgcc-version-dev, plus any -dev packages you want)
- Windows
  - Visual studio
  - cygwin+gcc `http://preshing.com/20141108/how-to-install-the-latest-gcc-on-windows/`
  - Eclipse+gcc (eclipse does not come with a C compiler) `http://www.dcs.vein.hu/bertok/oktatas/cpp_by_eclipse/eclipse_for_cpp_on_windows.html`
  - Anaconda `https://www.anaconda.com/products/individual`
  - Linux in Virtualbox

Random numbers

- Gnu Scientific library
  - variable: `gsl_rng *r;`
  - reading environment variables `GSL_RNG_TYPE` and `GSL_RNG_SEED`: `gsl_rng_env_setup`
  - `gsl_rng_default=gsl_rng_mt19937` Mersenne twister algorithm period: $2^{19937} - 1$
  - Set seed: `gsl_rng_set(r,seed);`
  - Integer random numbers between `gsl_rng_max(r)` and `gsl_rng_min(r)`: `unsigned long gsl_rng_get(r);`
  - *double* random numbers from 0 to 1 (0 included, 1 excluded) `gsl_rng_uniform(r);`

Requirements

- Minimum requirements
  - 5 homeworks submitted and accepted ($> 50\%$)
  - Presented and accepted project
- Exam: mark
  - **500 point**: 5 homeworks (deadlines are **soft!**)
  - **400 points**: Small (30 min) test (individual) (must pass!) or 4 extra homeworks
  - **900 points**: From projects (pairs/groups) presented at the end of the course.
  - **20 points/piece** extra: practice exercises can earn you 20 extra points each. Deadlines are **hard!**
  - The marks will be calculated using the 1800 point threshold
  - Upon request please, be ready for a code checkup to verify ownership. During this check you will be shown parts of your code and you are supposed to explain wht it was meant to do.
  - Turn it in language: English, Hungarian, German, French

Requirements

- Homeworks
  - Must be individual work
  - Documented working codes (no extra libraries except for `gsl`)
  - If needed and/or a pdf documentation of the results and explanation, or images.
  - Please combine all files in a zip/rar/tgz/etc. archive and upload a sinlge file
  - Python notebooks are also accepted, please do not clear the results from the notebook!
  - Using fancy visualization techniques does not improve the mark which is given for the algorithm, the efficiency of the code and the solution of the problem
  - Please, keep in mind that I do not have time to compile and run your code. Make it human readable!

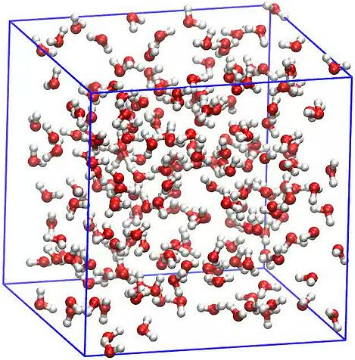
Literature

- There used to be a list of outdated books here.
- I will try to give you material on subject basis: For this one:
- http://www.lce.hut.fi/teaching/S-114.1100/lect_8.pdf
- https://arxiv.org/pdf/1005.4117.pdf
- https://www.ks.uiuc.edu/Services/Class/PHYS498NSM/

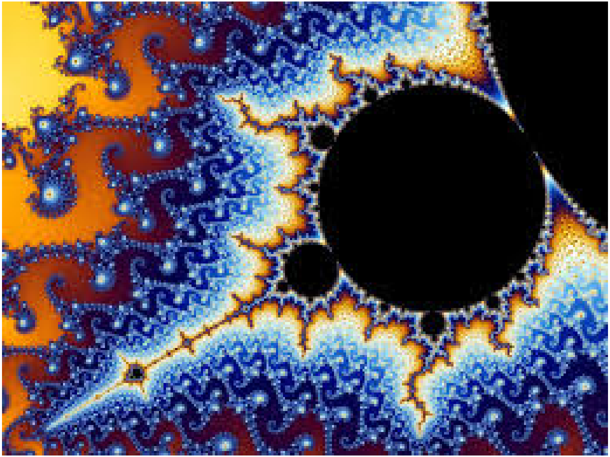
Subjects

1. Random numbers
2. Molecular dynamics
3. Other particle based methods
4. Percolation, Fractals
5. Ising, Schelling
6. Schrödinger equation
7. Quantum scattering
8. Optimization (annealing, genetic)
9. Complex networks
10. Clustering, community detection
11. Algorithmically defined models
12. Neural networks
13. Game models
14. Presentation

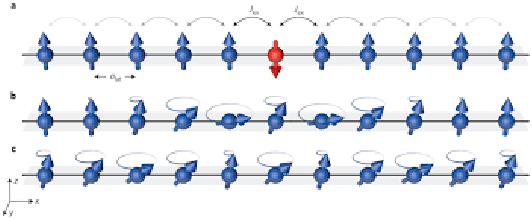
Discrete element methods



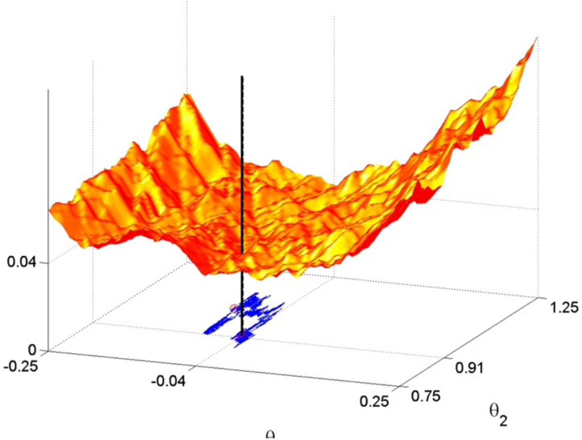
Percolation, Fractals



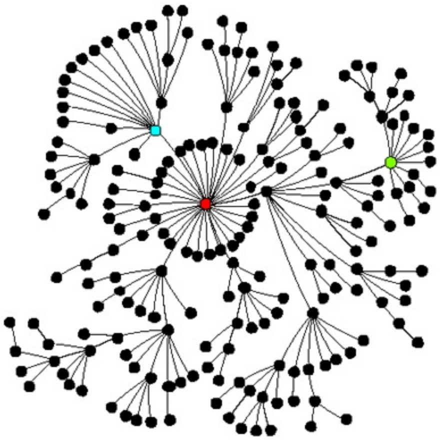
Ising, Heisenberg model



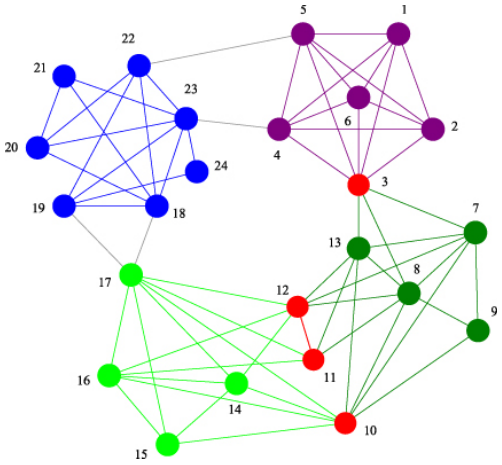
Optimization



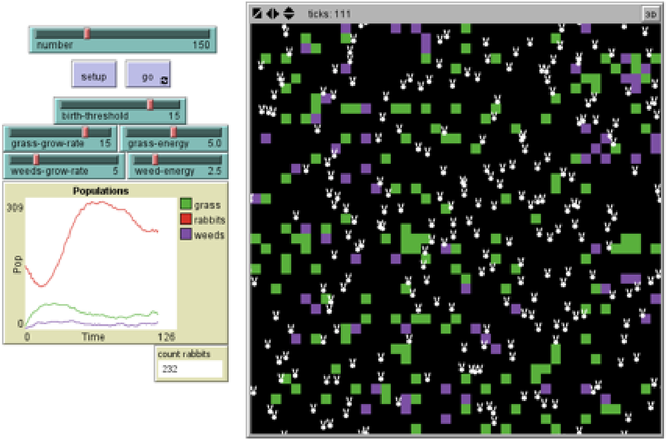
Complex networks



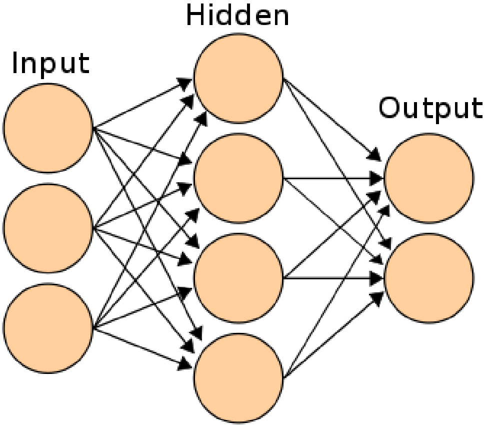
Clustering



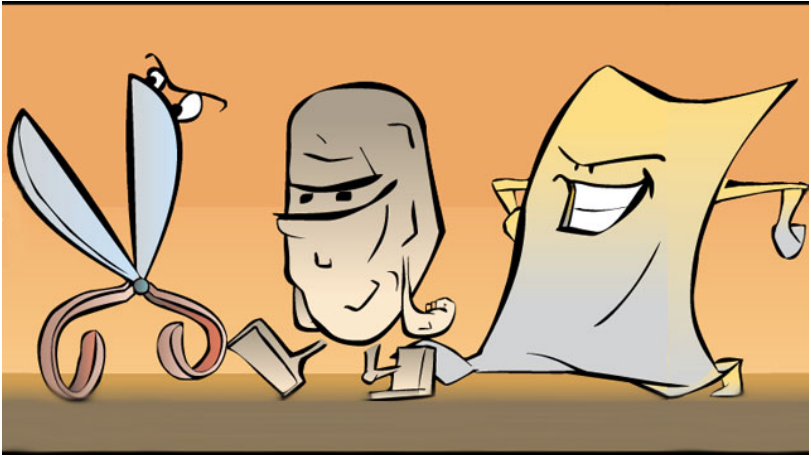
Algorithmically defined models



Neural networks



Game models



Simulations

Experiments	*Simulations*
Principle of measurement	Algorithm
Apparatus	Program + Hardware
Calibration	Calibration + Debugging
Sample	Sample
Measurement	Run

Data collection
Analysis

Programming languages

Simulations codes

- System size must be large
  - Phase transition $\xi \to \infty$
  - Real systems $N \sim 10^{23}$ (memory $< 10^{12}$)
- Simulation time should be long
  - Relaxation time
  - Interesting phenomena take long
  - Separation of time scales

Must be efficient!

It is not bad if program is readable and extensible...

Sample preparation

- Sometimes it is also a simulation

Data analysis

- Anything may happen

Programming languages

Problem to solve:

- Fill an array with product of two random numbers
- Calculate the average of them

python

```python
#!/usr/bin/python
import random

N = 20000000
s = []

for a in range(N):
  s.append(random.random() * random.random())

av = 0
for a in range(N):
  av += s[a]

print av / N
```

C

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>


int main(np,para)
char *para[];
{
  int N = 200000000;
  int a;
  double *s,av;

  s = (double *) malloc(sizeof(double) * N);
  av = 0.0;
  for (a=0;a<N;a++) {
    s[a] = (double)rand() / RAND_MAX * rand() / RAND_MAX;
  }
  for (a=0;a<N;a++) {
    av += s[a];
  }
  printf("%lg\n",av/N);
}
```

Programming languages

```python
#!/usr/bin/python
import random

N = 20000000
s = []

for a in range(N):
  s.append(random.random() * random.random())

av = 0
for a in range(N):
  av += s[a]

print av / N
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>


int main(np,para)
char *para[];
{
  int N = 200000000;
  int a;
  double *s,av;

  s = (double *) malloc(sizeof(double) * N);
  av = 0.0;
  for (a=0;a<N;a++) {
    s[a] = (double)rand() / RAND_MAX * rand() / RAND_MAX;
  }
  for (a=0;a<N;a++) {
    av += s[a];
  }
  printf("%lg\n",av/N);
}
```

```python
#!/usr/bin/python
import numpy

N = 200000000
s = numpy.random.random(N)
s *= numpy.random.random(N)

print s.mean()
```

6.9s	3.46s
4.51s	3.29s

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>


int main(np,para)
char *para[];
{
  int N = 200000000;
  int a;
  double *s,av;

  s = (double *) malloc(sizeof(double) * N);
  av = 0.0;
  for (a=0;a<N;a++) {
    s[a] = (double)rand() / RAND_MAX * rand() / RAND_MAX;
    av += s[a];
  }
  printf("%lg\n",av/N);
}
```

Optimization

- Programming language
  - In example C is 1.3-2 times faster than python
  - Matlab can be very efficient, but it is proprietary
  - Matlab, Maple, Mathematica are expensive
  - All clusters have C, and C++, and python
- Optimization
  - Parallelization
  - Indexing, careful usage of pointers
  - Reformulate operations
  - Does not always worth the pain
  - `gprof`

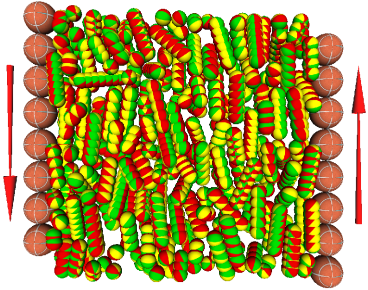
gprof

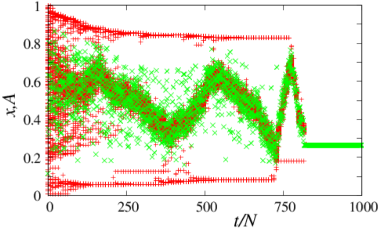
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 37.66     56.83    56.83 3248064862     0.00     0.00  is_in_community
 25.99     96.05    39.22  1000000     0.04     0.04  e_erode
 11.55    113.47    17.43 21355853     0.00     0.00  weighted_random_link
  6.33    123.03     9.55 11078805     0.00     0.00  weighted_random_link_ban_list
  3.02    127.58     4.55  8406648     0.00     0.01  e_info
  2.77    131.75     4.18                             main
  2.26    135.16     3.40 197988614     0.00     0.00  ct_weight
  2.10    138.33     3.17        4   792.50   792.50  clear_data
  1.85    141.12     2.79 12949626     0.00     0.00  e_single
  1.73    143.74     2.62 164260875     0.00     0.00  ranksz
  1.60    146.16     2.42 12774907     0.00     0.00  strengthen
  0.97    147.62     1.46 19359356     0.00     0.01  communicate
  0.88    148.94     1.33 248428917     0.00     0.00  is_internet
  0.32    149.43     0.48    15380     0.03     0.03  random_agent_with_group_sex
  0.31    149.90     0.47  2042439     0.00     0.00  e_share
  0.24    150.25     0.36                             seed3
```

Simulations

- Do what nature does
  - Molecular dynamics
  - Hydrodynamics
- Make use of statistical physics
  - Monte-Carlo dynamics
  - Simulate simplified models
  - Much smaller codes!





Random numbers

- Why?
  - Ensemble average:
    $$\langle A \rangle = \sum_i A_i P_i^{\mathrm{eq}}$$
    Random initial configurations
  - Model: e.g. Monte-Carlo
  - Fluctuations
  - Sample
- How?

Generate random numbers

- We need good randomness:
  - Correlations of random numbers appear in the results
  - Must be fast
  - Long cycle
  - Cryptography

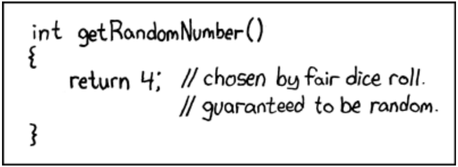
Random number generators

- True (Physical phenomena):
  - Shot noise (circuit)
  - Nuclear decay
  - Amplification of noise
    - Atmospheric noise (`random.org`)
    - Thermal noise of resistor
    - Reverse biased transistor
    - Lava lamps
  - Limited speed
  - Needed for cryptography
- Pseudo (algorithm):
  - Deterministic
    - Good for debugging!
  - Fast
  - Can be made reliable

Language provided random numbers

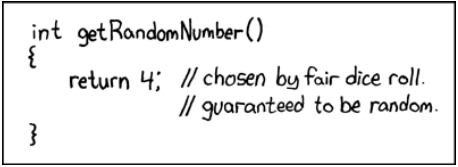
It is good to know what the computer does!

- Algorithm
  - Performance
  - Precision
  - Limit cycle
  - Historically(?) a catastrophe



Language provided random numbers

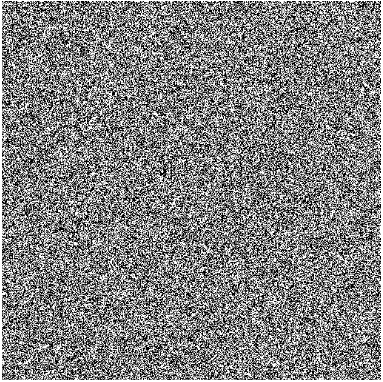
It is good to know what the computer does!



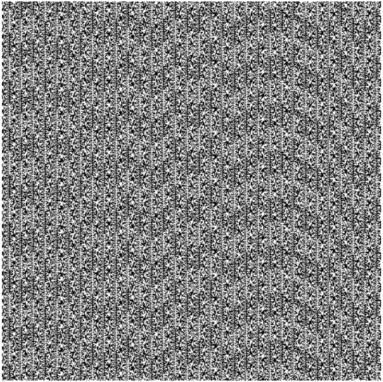
Language provided random numbers

It is good to know what the computer does!

Random



php rand() on Windows



Language provided random numbers

It is good to know what the computer does!

- Algorithm
  - Performance
  - Precision
  - Limit cycle
  - Historically a catastrophe
- Seed
  - From true random source
  - Time
  - Manual
    - Allows debugging
    - Ensures difference

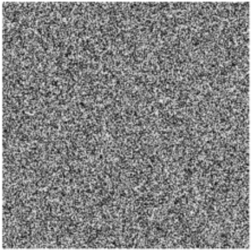
First only uniform random numbers

Seed

- From true random source
- Time
- Manual

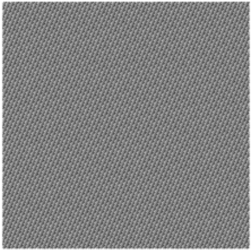
Random number generator of Python with different seeds:

System.Random
numbers 0...n of seed 0



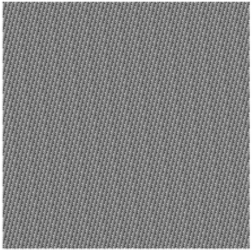
Sequence of 65536 random values.

System.Random
0th number of seed 0...n



Sequence of 65536 random values.

Linear function i * 19969 / 207
numbers 0...n

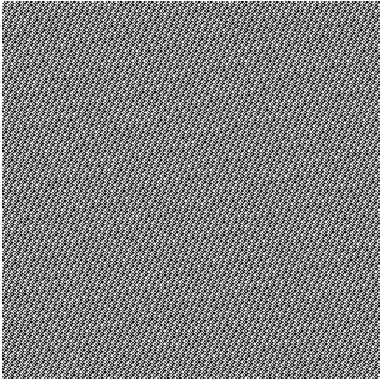


Sequence of 65536 random values.

Seed

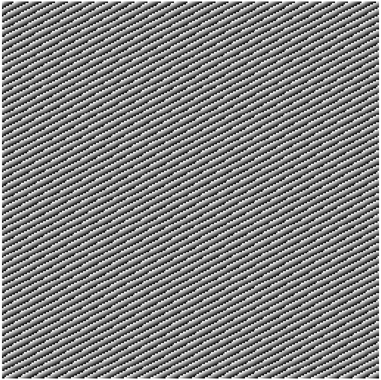
- Ensemble average: Include in the code if possible instead of restarting it with different seeds!

System.Random
0th number of seed 0...n



Sequence of 65536 random values.

System.Random
100th number of seed 0...n



Sequence of 65536 random values.

Multiplicative congruential algorithm

- Let r_j be an integer number, the next is generated by

$$r_{j+1} = (ar_j + c)\text{mod}(m),$$

- Sometimes only k bits are used
- Values between 0 and $m - 1$ or $2^k - 1$
- Three parameters (a, c, m).
- If $m = 2^X$ is fast. Use AND (&) instead of modulo (%).
- Good:
  - Historical choice:
    $a = 7^5 = 16807,\ m = 2^{31} - 1 = 2147483647,\ c = 0$
  - gcc built-in $(k = 31)$:
    $a = 1103515245,\ m = 2^{31} = 2147483648,\ c = 12345$
- Bad:
  - RANDU: $a = 65539,\ m = 2^{31} = 2147483648,\ c = 0$

Tausworth, Kirkpatrick-Stoll generator

- Fill an array of 256 integers with random numbers

$$J[k] = J[(k-250)\&255] \hat{\ } J[(k-103)\&255]$$

- Return $J[k]$, increase k by one
- Can be 64 bit number
- Extremely fast, but short cycles for certain seeds

XOR function

^	1	0
1	0	1
0	1	0

Tausworth, Kirkpatrick-Stoll generator corrected by Zipf

The one the lecturer uses

- Fill an array of 256 integers with random numbers

$$J[k] = J[(k-250)\&255]\hat{\ }J[(k-103)\&255]$$

  Increase k by one

$$J[k] = J[(k-30)\&255]\hat{\ }J[(k-127)\&255]$$

- Return $J[k]$, increase k by one
- Extremely fast, reliable also on bit level
- General transformation $x \in [0:1[$

$$x = r/(RAND\_MAX + 1)$$

Floating point random numbers

- General transformation $x \in [0:1[$

$$x = r/(RAND\_MAX + 1)$$

- It is important to know whether limits are included or not
- General feature: 0 included 1 not
- Generate integer number from 1,2,3. use $i = r\%3$ (modulo) result: 1 will be $1 + 10^{-9}$ more probable than 2 or 3.
- General practice use division instead of percentage, higher bits are more reliable for LCG

Tests

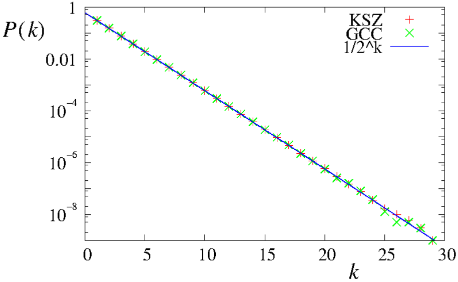
- General: e.g. TESTU01
- Diehard tests:
  - Birthday spacings (spacing is exponential)
  - Monkey tests (random typewriter problem)
  - Parking lot test
  - Moments: $m = \int_0^1 \frac{1}{n+1}$
  - Correlation
    $$C_{q,q'}(t) = \int_0^1 \int_0^1 x^q x'^{q'} P[x, x'(t)] dx dx' = \frac{1}{(q+1)(q'+1)}$$
  - Fourier-spectra
  - Fill of d dimensional lattice
  - Random walks

Red ones are not always fulfilled!

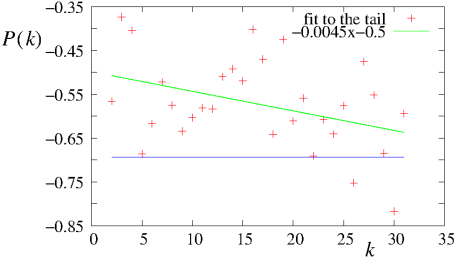
- Certain Multiplicative congruential generators are bad on bit series distribution, not completely position independent.

Bit series distribution

Probability of having k times the same bit

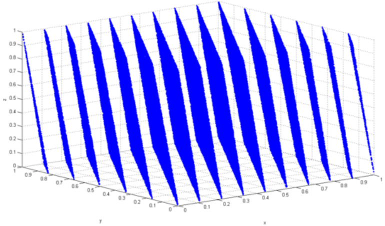


Fit to the tail for different bit positions (gcc)



Fill of d dimensional lattice

- Generate d random numbers $c_i \in [0, L]$
- Set $x[c_1, c_2, \ldots, c_d] = 1$
- The Marsaglia effect is that for all congruential multiplicative generators there will be unavailable points (on hyperplanes) if d is large enough.
- For RANDU $d = 3$



Solution for Marsaglia effect

- Instead of d random numbers only 1 (x)
- Divide it int d parts: k=int($\log_d$(RANDMAX))
  `c_1=x%k, x/=k`
  `c_2=x%k, x/=k`
  ...
- Better to have $L = 2^k$. Which is much faster because of AND and SHIFT operations

General advice: Save time by generating less random numbers

Random numbers with different distributions

- Let us have a good random number $r \in [0, 1]$.
- The probability density function is $P(x)$
- The cumulative distribution is

$$D(x) = \int_{-\infty}^{x} P(x')dx'$$

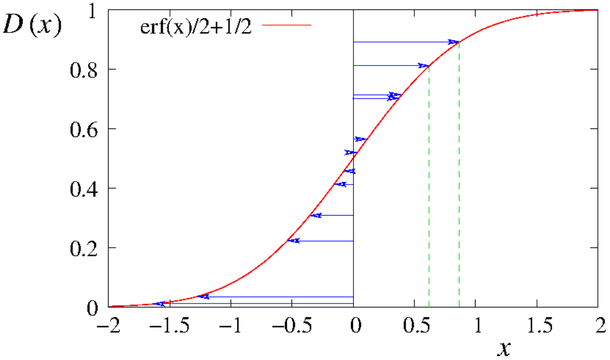
- Obviously:

$$P(x) = D'(x)$$

- The numbers $D^{-1}(x)$ will be distributed according to $P(x)$
- $D^{-1}(x)$ is the inverse function of $D(x)$ not always easy to get!

Random numbers with different distributions

Graphical representation

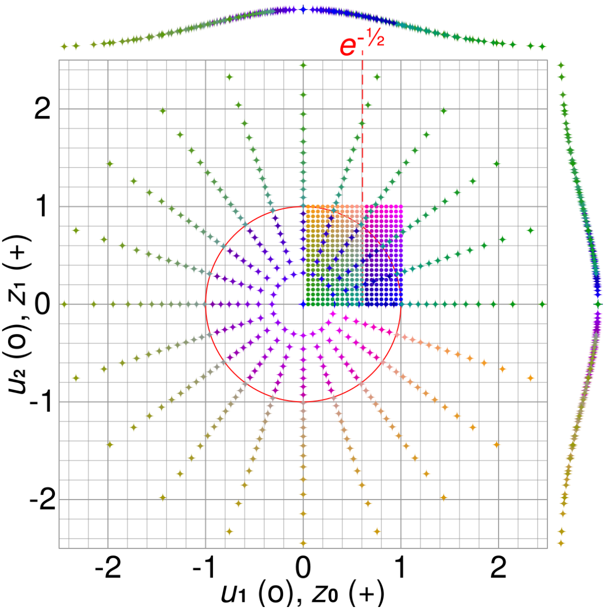


Random numbers with different distributions

A soluable example

- $$P(x) = \frac{1}{\pi}\frac{1}{1+x^2}$$
- $$D(x) = \frac{1}{\pi}\int_{-\infty}^{x}\frac{1+x'^2}{d}x' = \frac{1}{2} + \frac{1}{\pi}\mathrm{arctan}(x)$$
- $$x = D^{-1}(y) = \tan\left[\pi(y-1/2)\right]$$

Box-Müller method



Box-Müller method

- r_1, r_2 uniformly distributed between 0 and 1
- if $r_1^2 + r_2^2 \geq 1$ or $r_1 = r_2 = 0$ discard r_1 and r_2. So the generated point is inside the unit circle and is not at the origin
- $R^2 = r_1^2 + r_2^2$ is uniformly distributed between 0 and 1.
- $U_1 \equiv R^2$, $U_2 \equiv \mathrm{arctan}(r1/r2)/(2\pi)$
- Two independent normally distributed random numbers:

$$x_1 = \sqrt{-2\log U_1}\cos(2\pi U_2) = r_1\sqrt{\frac{-2\log(R^2)}{R^2}}$$

$$x_2 = \sqrt{-2\log U_1}\sin(2\pi U_2) = r_2\sqrt{\frac{-2\log(R^2)}{R^2}}$$

Power law distributed random numbers

Let $P(y)$ have uniform distribution in $[0, 1]$. We generate $P(x)$ such as

$$P(x) = Cx^n$$

for $x \in [x_0, x_1]$.

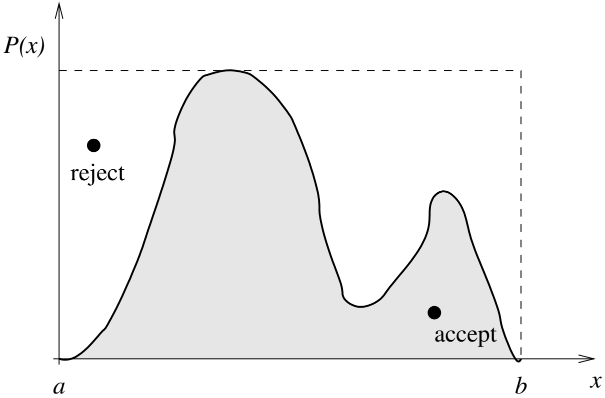
$$D(x) = \int_{x_0}^{x} P(x')dx' = \frac{C}{n+1}\left(x^{n+1} - x_0^{n+1}\right)$$

The inverse function is simple:

$$x = \left[\left(x_1^{n+1} - x_0^{n+1}\right) y + x_0^{n+1}\right]^{1/(n+1)}$$

Monte Carlo

- Identify base: $[a, b]$
- Identify minimum/maximum: $P_{\max} = \max_{x \in [a,b]} P(x)$, idem...
- Generate a point (x, y) in the rectangle $(a, P_{\min})$, $(b, P_{\max})$
- If $y < P(x)$ the return x otherwise generate new point



Error

- Ensemble average
- Example: estimate π
- Drop a needle of length $l \leq t$
- May or may not cross a line

$$p_{\text{cross}} = \frac{2l}{t\pi}$$

- Lazzarini in 1901 using $N = 3408$ tries got:

$$\pi \simeq 355/113 = 3.14159292 = \pi + \mathcal{O}(10^{-7})$$

- Impressive 10^{-7} error, but

$$\frac{1}{\sqrt{N}} \simeq 0.0017$$