Random numbers and Monte Carlo techniques

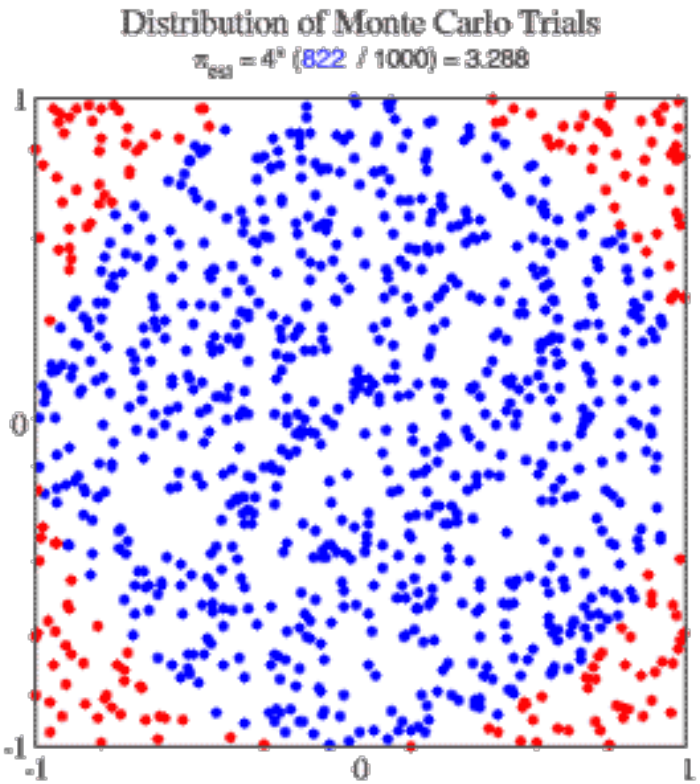
Alexander Khanov

PHYS6260: Experimental Methods is HEP
Oklahoma State University

October 11, 2023

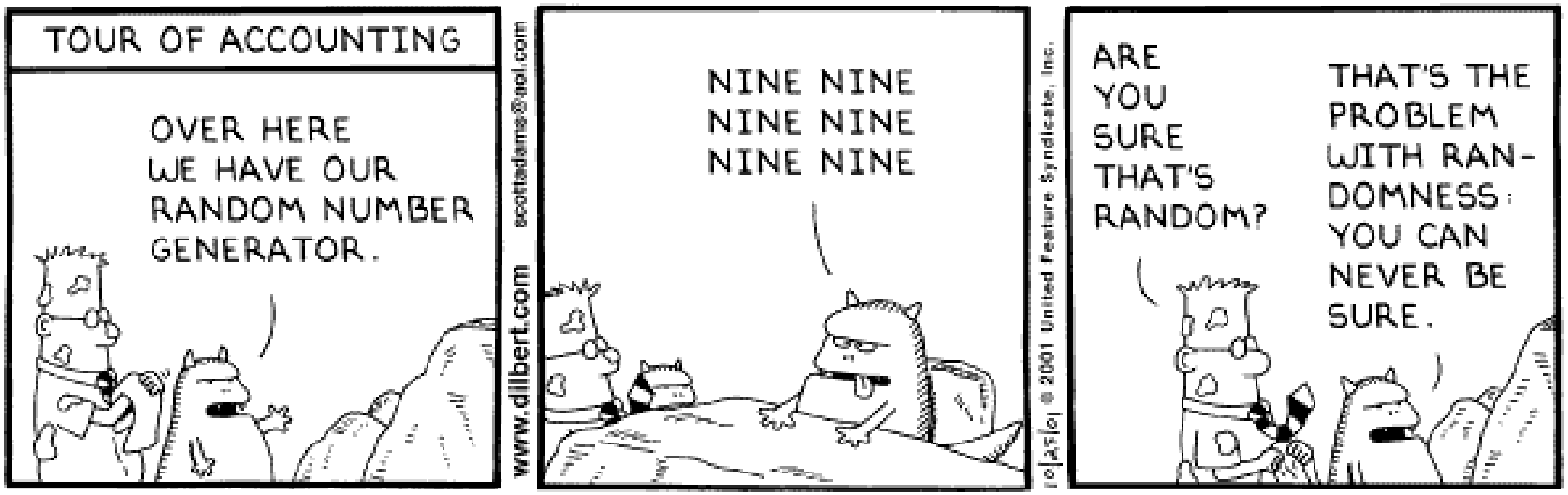
Random numbers

- What is a random number?
  - Is 3 a random number?
- Random numbers always come in sequences
- Why do we need random numbers?
  - to calculate integrals!



What is a good random number?

- Random numbers are "uniform"
  - uniformity: numbers are equally probable everywhere
- Random numbers are "unpredictable"
  - independence: current value is not related to previous values
- Random numbers are not like this:



Random number tests

- Making sure that random numbers are random is a big deal

  `https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software/guide-to-the-statistical-tests`
- We can't define what a random sequence is, only what it isn't
- General approach: calculate some properties of the random sequence (number of 1's and 0's, length of groups consisting of 1's, etc), compare with prediction for a "true" random sequence
- Procedure:
  - null hypothesis: the sequence is random
  - carry out the testing procedure
  - calculate the $\mathcal{P}$-value
  - the sequence is declared random (null hypothesis is successful) if $\mathcal{P} \geq \alpha$, where α is a predefined value (e.g. 0.01 for 99% C.L.)
- Popular tests: χ^2, Kolmogorov-Smirnov

Random number tests (2)

- χ^2 test
  - suppose we have n possible categories of observations, in each category i we expect e_i occurrences and observe o_i occurrences, then

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i}$$

    is distributed as χ^2 with $n-1$ degrees of freedom
  - χ^2 should be less than certain value
- Kolmogorov-Smirnov test
  - suppose $F(x)$ is cumulative distribution function (c.d.f.)
    $F(x) = \int_{-\infty}^{x} f(x)\,dx$ where $f(x)$ is probability density function (pdf)
  - compare observed and expected cdf (usually in bins of x)
  - calculate maximum deviations
    $K^+ = \sqrt{n}\max_x(F_o(x) - F_e(x)) \quad K^- = \sqrt{n}\max_x(F_e(x) - F_o(x))$
  - K^+, K^- should be less than certain value

Methods to generate random numbers

- Hardware random number generators
  - radioactivity, cosmic rays, thermal noise
- `/dev/random`
  - "entropy harvesting": collect environmental noise from LAN traffic, serial line traffic, HW and SW interrupts etc.
- Pseudo random generators

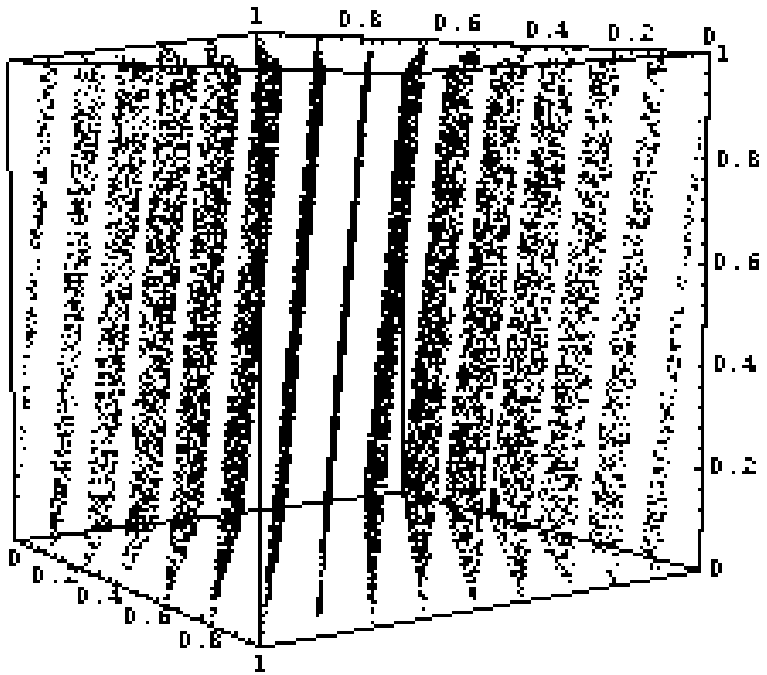
Pseudo random generators

- Linear congruential generators LCG(m, a, c, x_0)

$$x_{i+1} = (ax_i + c) \mod m$$

- If (and only if)
  - c is relatively prime to m,
  - $a - 1$ is a multiple of p, for every prime p dividing m, and
  - $a - 1$ is a multiple of 4, if p is a multiple of 4,

  then LCG has a period of m
- LCG may or may not be good
  - infamous example: IBM generator RANDU, LCG(2^{31},65539,0,1)



Mersenne Twister

- State-of-the-art in random number generation (since 1997)
  - ROOT: TRandom3
- Very fast and short (but not very easy to understand)
- Period is a Mersenne number (a prime number of the form $2^m - 1$; for the popular implementation, $m = 19937$)
- Can be improved (cf SFMT)

Integer and floating point random numbers

- Random number generators produce sequences of integers r_i, $0 \leq r_i < \mathrm{MAX}$
- We are usually interested in real random numbers that follow some distribution
- Integer to uniform: pick $\mathrm{MAX} = 2^p$, p=precision (e.g. 32), then $q_i = r_i/\mathrm{MAX}$ is uniform on (0,1) (and $a + (b - a)q_i$ is uniform on (a, b)

Generating distributions (1)

- Gaussian distribution: Box-Muller transform
  - if u_1, u_2 are independent random variables uniform on (0,1), then $\sqrt{-2\ln x_1}\cos(2\pi x_2)$, $\sqrt{-2\ln x_1}\sin(2\pi x_2)$ are independent random variables with normal distribution ($x_0 = 0$, $\sigma = 1$)
- Gaussian distribution, multivariate case with covariance matrix V:
  - generate n independent Gaussian variables $\mathbf{g}$ with mean 0 and variance 1 as above
  - $\mathbf{x} = \langle \mathbf{x} \rangle + L\mathbf{g}$ where L is the (unique) lower triangular matrix that satisfies $V = LL^T$, it can be found using Cholesky's decomposition

Generating distributions (2)

- Poisson distribution with mean λ: Knuth's algorithm
  - count the number of times you multiply uniform random numbers until the product becomes less than $e^{-\lambda}$

```
int poissonRandomNumber(int lambda) {
    double L = Math.exp(-lambda);
    int k = 0;
    double p = 1;
    do {
        k = k + 1;
        double u = Math.random();
        p = p * u;
    } while (p > L);
    return k - 1;
}
```

- if λ is large, the algorithm takes too long to complete
  - Gaussian is a good approximation

Generating distributions (3)

- Isotropic direction in 3d:
  - "isotropic" = density proportional to solid angle $d\Omega = d\varphi\, sin\theta\, d\theta = -d\varphi\, d(\cos\theta)$
  - $\cos\theta$ is uniform on (-1,1) and φ is uniform on $(0,2\pi)$

  . . . etc.
- What to do for an arbitrary distribution?

Acceptance-rejection

- The algorithm:
  - encapsulate the p.d.f. $f(x)$ in a box $a < x < b$, $0 < f(x) < f_{max}$
  - generate a random number x uniform on (a, b)
  - generate a random number y uniform on $(0, f_{max})$
  - accept x as result if $y < f(x)$
- Can be easily implemented for binned p.d.f.
- Problems:
  - doesn't work for $-\infty < x < +\infty$
  - inefficient for pole-like functions
- Possible improvements: split the x range in subranges, each with its own f_{max} ("adaptive rejection") – improves efficiency

Inverse transform

- Based on the fact that c.d.f. $F(x) == \int_a^x f(x)dx$ is by itself a random variable uniform on (0,1)
- The algorithm:
  - generate a random number y uniform on (0,1)
  - solve for x the equation $\frac{\int_a^x f(x)dx}{\int_a^b f(x)dx} = y$
- Handy if it's easy to find $F^{-1}(y)$
- Can be easily implemented for binned c.d.f.

Weights

- What if acception-rejection is impractical and inverting the integral is too much work?
- We can do weighted Monte Carlo
- The algorithm:
  - approximate the "bad" function $f(x)$ with a "good" function $f^*(x)$
  - generate a random number x with p.d.f. $f^*(x)$
  - assign weight $w = \dfrac{f(x)}{f^*(x)} \sim 1$
  - two options: do acceptance-rejection based on w/w_{max} (less efficient) or count "events" taking the weights into account

Multidimensional case

- It helps if the variables are separable (and therefore uncorrelated): $f(x_1, \ldots, x_n) = \prod_{i=1}^{n} f_i(x_i)$
- Otherwise, distribution along each dimension has to be calculated
- There are special methods to do it efficiently

Metropolis-Hastings algorithm

- This is an example of Markov Chain Monte Carlo
- It doesn't require p.d.f. $f(x)$ to be normalized to 1 – it will work with any function proportional to p.d.f.
- The algorithm:
  - pick a "proposal p.d.f." $q(x', x)$ which gives the probability for the next candidate x' provided the current sample value is x. $q(x', x)$ can be arbitrary (provided $q(x', x) = q(x, x')$), the usual choice is a (multivariate) Gaussian centered at x
  - pick a starting point x_0 (again, it can be arbitrary)
  - at each step, given current state x, generate next candidate state x' according to $q(x', x)$
  - calculate the "acceptance ratio" $\alpha = f(x')/f(x)$
  - if $\alpha > 1$ then the new state is more probable than the old one – accept it (i.e. assign x' to x)
  - otherwise, generate a random number u uniform on (0,1) and accept the new state if $\alpha > u$
  - otherwise, repeat with the old state x
- The Metropolis-Hastings algorithm is very close to simulated annealing – a minimization heuristic
  - Both methods work very well in highly multidimensional cases