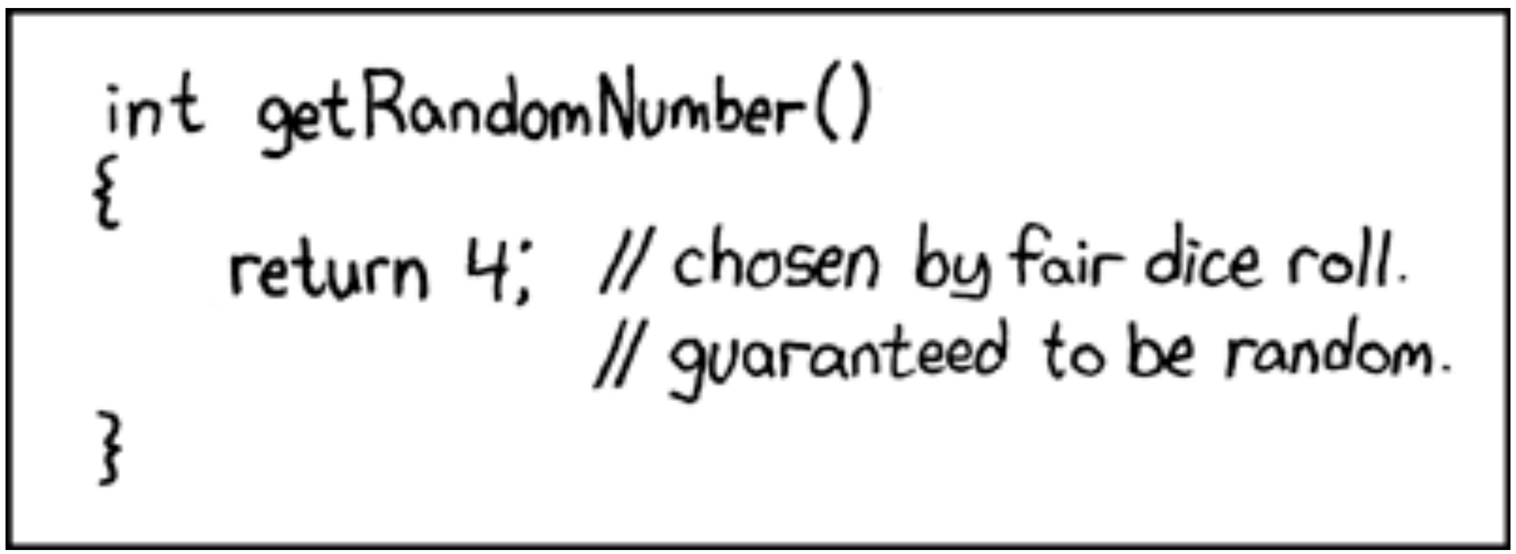
Random Number Generation

Introducing uncertainty on purpose

Based on: "Random Numbers in Scientific Computing: An Introduction", Katzgrabber, arXiv:1005.4117

Need Random Numbers

- For randomly sampling a domain
- Monte Carlo / MCMC simulations
- Stochastic algorithms



http://xkcd.com/221/

Required Properties

- What is a random sequence of numbers?
- Follow some desired distribribution
- Unpredictable
- Fast (we may need billions of them)
- Long period (we may need billions of them)
- Uncorrelated

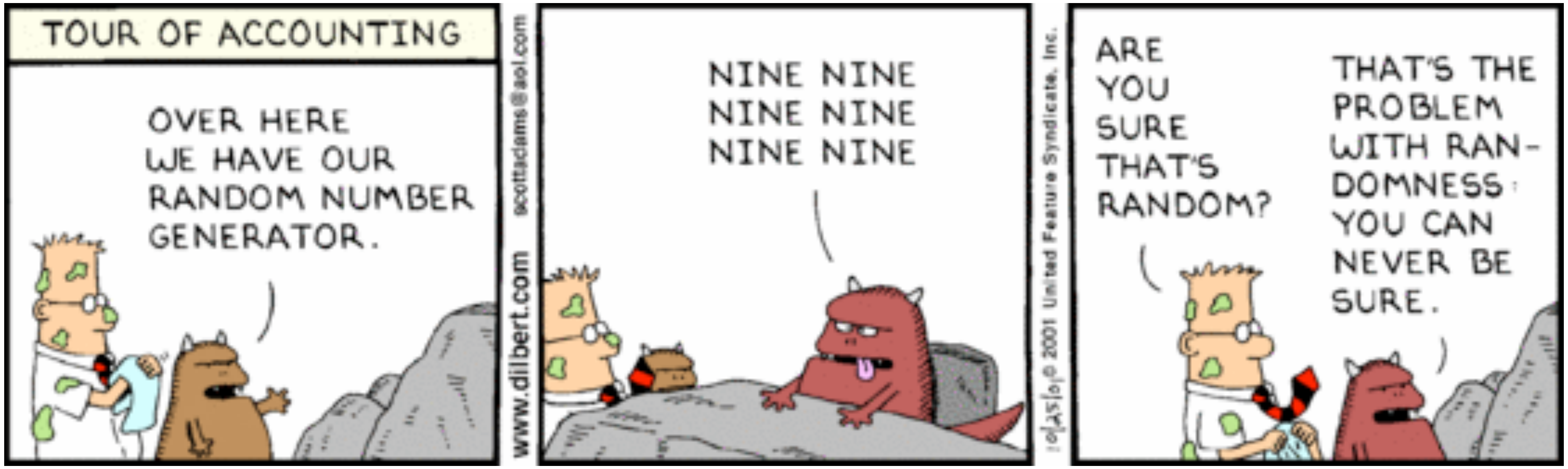
Real Random Numbers

- Can be generated by a physical process, and stored as a list or used in real-time by computer
- Physical process - lava lamp (lavarnd.org), quantum stuff
- Network process - /dev/urandom
- Generally slow, expensive, hard/impossible to reproduce for debugging
- Often hard to characterize underlying distribution

Pseudo Random Number Generators

- PRNG
- Software-based; deterministic sequences of numbers based on some starting seed
- “Seem” random, but reproducible (with same seed), often very fast.
- Will assume uniform distribution on [0,1); given this, can create other distributions

Randomness Tests



Common Tests: Correlations

- Simple pairwise correlations:
- Want to avoid correlations between pairs of numbers

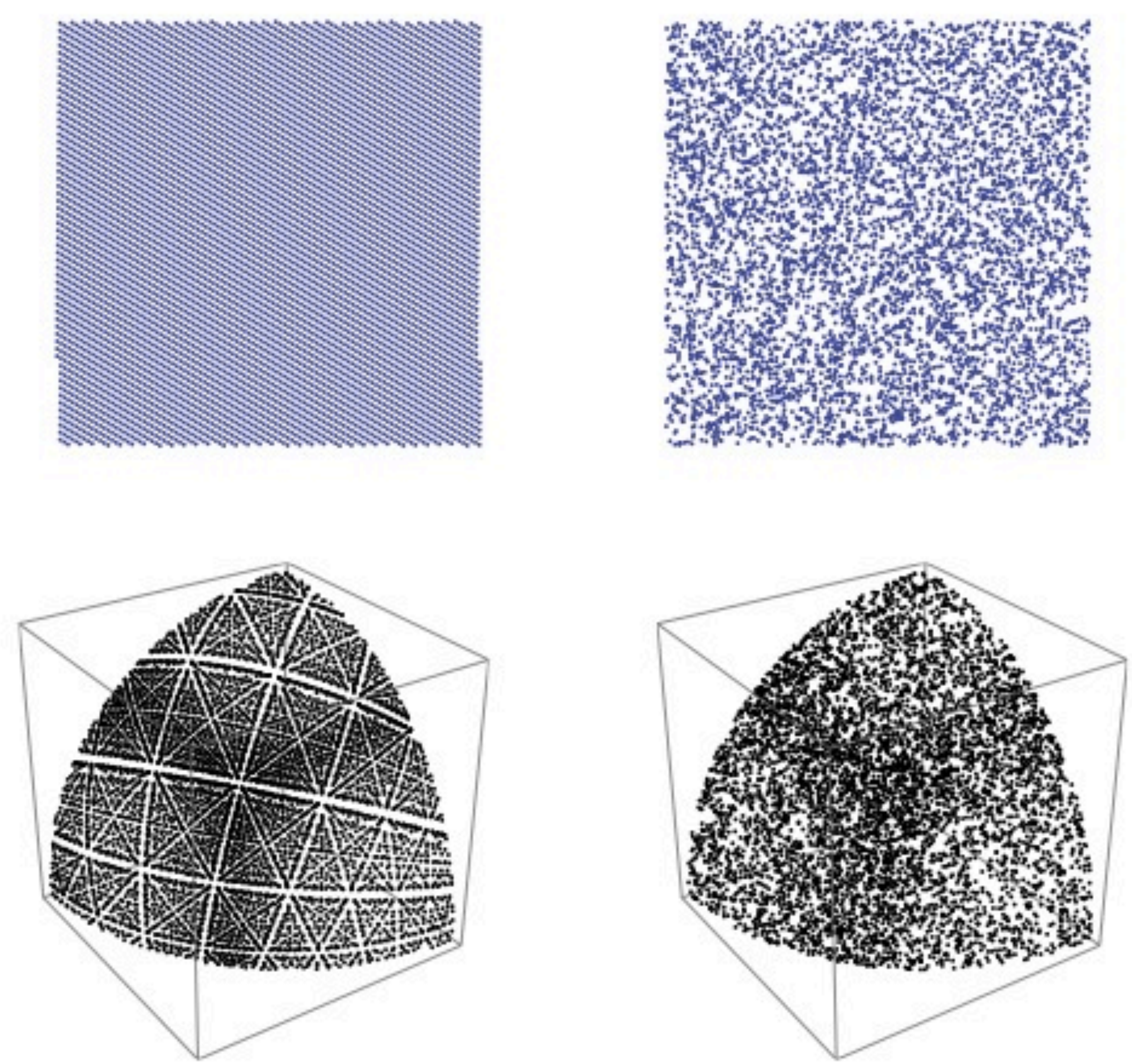
$$\varepsilon(N,n) = \frac{1}{N}\sum_{i=1}^{N} x_i x_{i+n} - E(x)^2$$

$$E(x) = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$E(N,n) = O(N^{-1/2}) \quad \forall n$$

Correlations

- What correlations look like in 2d domain
- Left: bad LCG; right: Mersenne Twister



From Katzgraber

Common Tests: Moments

$$\mu(N,k) = \left| \frac{1}{N} \sum_{i=1}^{N} x_i^k - \frac{1}{k+1} \right|$$

- Ensure moments of random numbers also have desired properties

$$\mu(N,k) = O(N^{-1/2}) \quad \forall k$$

Other Tests

- Overlapping permutations: Analyze orders of five consequitive random numbers. The 5! possible permutations should occur with equal probability

- Parking lot test: pairs of random numbers placed in 2-d domain, exclude others within certain distance. After N attempts, points should follow well known distribution

- Spacings: spacings between random points should follow poisson integral if uniformly distributed

- Binary rank test - test ranks of 32x32 binary matrix

Test suites

- NIST test suite:
  http://csrc.nist.gov/groups/ST/toolkit/rng/index.html
  Very well documented, explain tests.

- Pierre L'Ecuyer, U de Montréal:
  http://www.iro.umontreal.ca/~simardr/testu01/tu01.html
  Test suite in C, includes several PRNGs

- Best test: one that is related to the properties you need for your problem.

SciNet compute • calcul CANADA

Linear Congruential Generators

- x_0 is a seed
- m - large integer; determines period of sequence
- For U(0,1), divide x_i by m.
- For good results: c relatively prime to m, a-1 a multiple of p for every prime divisor p of m, a-1 is multiple of 4 if m is multiple of 4.

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

SciNet compute • calcul CANADA

Linear Congruential Generators

- Common, but not very good
- Period limited by size of integers; not enough for some applications.
- Hard to do well in parallel
- **Easy** to mess up, with long history of bad LCGs in standard implementations, literature.

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

SciNet compute • calcul CANADA

Linear Feedback Shift Register Generators

- Generalization of LCG

$$x_i = (a_1 x_{i-1} + \cdots + a_n x_{i-n}) \mod p$$

- Good period iff characteristic polynomial defined by a_i is primitive modulo p
- Requires big seed (n x_is); typically use small seed + good small PRNG to seed
- Still not great - better period (p^n).
- Mersene Twister is a (good) generalization of this.

SciNet compute • calcul CANADA

Lagged Fibonacci

- Some binary operator between previous items in sequence
- Requires some memory
- Requires large seed block again
- m typically large power of 2

$$x_i = (x_{i-j} \odot x_{i-k}) \mod m$$

Lagged Fibonacci

$$x_i = (x_{i-j} \odot x_{i-k}) \mod m$$

- r1279: k=1279. Period is 10^{394} ; passess tests, and can be fast
- Standard in (eg) GSL

Lagged Fibonacci

- r250: k = 250, using xor.
- Also fast, passed all common tests at time
- In 1992, Ferreberg et al did MC simulation of Ising model
- Estimate of energy/per spin was 42σ off!
- PRNGs are hard; **don't** implement yourself.

$$x_i = (x_{i-j} \odot x_{i-k}) \mod m$$

Some good PRNGs

- r1279
- Mersenne twister (mt19937)
- WELL generators

Not-good PRNGs

- r250
- Anything from Numerical Recipies - short periods, slow, ran0 & ran1 spectacularly fail statistical tests.
- Standard Unix generators (rand(), drand48()) - not a disaster, but short period, correlations.

Shifting distribution

- If just need to shift distribution, easy
- U(a,b): (b-a)*(u + a) where u from U(0,1)
- Can similarly shift gaussian distribution from unit, zero-mean gaussian to others

Non-Uniform Distributions

- Transformation law of probabilities
- Starting with a known distribution (eg, uniform, p(u) = 1 in 0..1), can transform to another distribution (q(y)) if can invert function

$$|q(y)dy| = |p(u)du|$$
$$\Rightarrow q(y) = p(u)\left|\frac{du}{dy}\right|$$

SciNet compute • calcul CANADA

Exponential Dist.

- Example: exponential distribtion
- Easy to invert, differentiate
- Can get exponential distribution by taking ln of uniform random numbers.

$$|q(y)dy| = |p(u)du|$$
$$\Rightarrow q(y) = p(u)\left|\frac{du}{dy}\right|$$

$$q(y) = a\exp(-ay)$$
$$\left|\frac{du}{dy}\right| = a\exp(-ay)$$
$$u(y) = \exp(-ay)$$

$$y = -\frac{1}{a}\ln(u)$$

Box-Muller: Gaussian Random Numbers

- Same process can be applied to more complex dists, with some tricks.
- For gaussian, can't do it in 1d, but can in 2
- Generate 2 gaussian RNs (unit σ, zero mean) from 2 uniform

$$x = \sqrt{-2\ln(u_2)}\cos(2\pi u_1)$$

$$y = \sqrt{-2\ln(u_2)}\sin(2\pi u_1)$$

Acceptance/Rejection

- If can't invert your desired distribution g(x), can still generate RN
- Numerically invert (tabulate)
- Or:
  - Generate distribution you can on same domain, g(x)
  - Reject numbers with probability 1-f(x)/g(x) (eg, generate random number u[0,1], x from g;
    accept if u< f(x)/g(x)
  - Faster if g tightly bounds f (less rejected guesses)

GSL - Gnu Scientific Library

- Gsl has several good implementations of good PRNGs
- Seperates the generator from the distribution you want

```c
#include <stdio.h>
#include <gsl/gsl_rng.h>

int main(int argc, char **argv) {
    gsl_rng *rng;
    int i;
    double u;

    rng = gsl_rng_alloc(gsl_rng_mt19937);
    gsl_rng_set(rng, 1);

    for (i=0; i<100; i++) {
        u = gsl_rng_uniform(rng);
        printf("%d %f\n", i, u);
    }

    gsl_rng_free(rng);

    return 0;
}
```

Create, seed PRNG

Generate Random #s

Clean up

```
$ gcc -o gsl gsl.c -I/path/to/gsl/include
  -L/path/to/gsl/lib -lgslcblas -lgsl
```

SciNet compute • calcul CANADA

Python

- Numpy.random - series of random number generators, distributions.
- Based on mersenne twister
- Good, but would be nice to have choice...

```
import numpy
import numpy.random

numpy.random.seed(1)

#uniform floats 0..1
nums = numpy.random.ranf(100)

print nums

#standard normal distribution
nums = numpy.random.randn(100)

print nums
```

Notes on Seeding

- For random seeds, taking system time is common
- If doing in parallel, need to make sure different processes/threads have different seeds!
- Factor rank, thread num, pid, etc in there somehow

Homework

- Consider the sequence of numbers: 1 followed by 10^8 values of 10^{-8}
- Should Sum to 2
- Write code which sums up those values in order. What answer does it get?
- Add to program routine which sums up values in reverse order. Does it get correct answer?
- How would you get correct answer?
- Submit code, Makefile, text file with answers.

SciNet compute • calcul CANADA

Homework: 2

- Implement an LCG with a = 106, c = 1283, m = 6075 that generates random numbers from 0..1
- Compare that and MT (using gsl: gsl_rng_mt19937 or python): generate pairs (dx, dy) with dx, dy each in -.1.. +.1. Generate histograms of dx and dy (say 200 bins). Look ok? What would you expect variation to be?
- For 10,000 pts: take random walks from 0,0 of step (dx,dy) until exceed radius of 2, then stop. Plot histogram of final angles for the two PRNGs. What do you see?
- Submit makefile, code, plots, VC log