Overview of lecture slides 01

1. Random numbers
   - Random number generators
   - Random numbers with non-uniform distributions
   - Inverse transform sampling
   - Rejection sampling
2. Summary

Random numbers

Why should we want random numbers?

- Simulate stochastic processes in nature
  - Brownian motion, crystal growth
  - Statistical physics, thermodynamics
  - Quantum mechanics, quantum field theory
  - Evolutionary processes, population dynamics
  - Stock markets, financial markets
- To simulate ‘unknowns’
- Randomised control trials, statistical analysis
- Monte Carlo integration
- Search algorithms

Random number generators

Some numbers are more random than others

No classical computer can create truly random numbers
— only physical processes can do that

- Throw dice, flip coins, roll roulette wheels
- Get ‘noise’ from the environment
- Or use series of numbers that only ‘look random’

Pseudo-random number generators

Generally, prngs use (integer) arithmetic to produce series of numbers
The series usually repeats itself after a finite number of steps

In computational physics we may need vast numbers of random numbers

- The period must be as long as we can get it
- There must be no ‘hidden’ correlations among numbers

Generating uniform random numbers: overview

Linear congruential algorithm

Simple, traditional algorithm: $X_{n+1} = (aX_n + c) \mod m$
a, c and m are integers.

Generates a sequence of integers between 0 and $m - 1$.
Period is at most m.

For a given m, sequence depends on choice of a, c, X_0.
Might look random-ish, or might look very repititive, depending on a, c, m.

Generating 'real' numbers in $[0, 1)$? Just divide by m.

Generating uniform random numbers: overview

Linear congruential algorithm

Simple, traditional algorithm: $X_{n+1} = (aX_n + c) \mod m$

a, c and m are integers. The period is at most m.

Correlations: Group into vectors: $\vec{x}_n = (x_n, x_{n+1}, x_{n+2})$.
Then the $\{\vec{x}_n\}$ will lie in distinct planes in 3-space. (Marsaglia, 1968)

For some choices, even worse correlations:
2D points $\vec{x}_n = (x_n, x_{n+1})$ fall along lines on a plane.

Examples on wikipedia.
Fun exercise: demonstrate Marsiglia phenomenon for choices of m, a, c.

Generating uniform random numbers: overview

Linear congruential algorithm

Simple, traditional algorithm: $X_{n+1} = (aX_n + c) \mod m$
a, c and m are integers. The period is at most m.

Correlations: Group into vectors: $\vec{x}_n = (x_n, x_{n+1}, x_{n+2})$.
Then the $\{\vec{x}_n\}$ will lie in distinct planes. (Marsaglia, 1968)

Considered unsuitable for serious Monte Carlo work

Modern algorithms

Mersenne twister, xorshift, ...
Even the best don't always pass all randomness tests
Good news: `numpy` uses good pnrg, based on Mersenne twister algorithm.

Lesson

Be suspicious of random number generators.
Make sure the one you use is good enough for your purpose.

PRNG's in python

Two different options :-(

package random

random.random() — return uniform real number in $[0.0, 1.0)$

random.gauss(mu, sigma) — return normally distributed real number with mean mu and width (standard dev) sigma.

random.randint(a,b) — return random integer $N \in [a, b]$

package numpy.random

numpy.random.rand() — uniform real number in $[0.0, 1.0)$

numpy.random.randn() — normally distributed real number, mean 0.0, st.dev. 1.0.

PRNG's in python

Two different package options :-(

random or numpy.random

Suggestion

Pick one and use it —
don't use both packages in the same code unless you really have to.
Maybe numpy.random has more options

Why? Scientific programming was not the top priority for python language. (Contrast: Fortran, matlab, julia languages)

Seeding

The ‘seed’ gives the starting point for the series

- If you want two identical sets of ‘random’ numbers, start with the same seed (eg to check your code)
- if you want two different sets of pseudo-random numbers, make sure you start with different seeds.

Python

Use `random.seed(x)`	to set the state (seed)
`random.seed()`	to set a seed based on the current time (useful for producing different numbers each
`random.getstate()`	to save the current state of the rng
`random.setstate()`	to reset to a saved state

Random numbers with non-uniform distributions

Simplest prngs produce a uniform distribution between 0 and 1
[or integers between 0 and `RAND_MAX`]

$$P(X \in [x_1, x_1{+}\Delta x]) = P(X \in [x_2, x_2{+}\Delta x]) = \Delta x \quad \forall (x_1, x_2) \in \langle 0, 1{-}\Delta x \rangle$$

We may want different distributions:

- exponential
- gaussian
- poisson
- linear
- more complicated, in one or more dimensions

Normalisation

All distributions must obey $\int_{-\infty}^{\infty} P(x) dx = 1$

Non-uniform random numbers

Producing random numbers with a desired distribution

Given a pnrg with uniform distribution, can we generate random numbers with some desired statistical distribution?

- inverse transform sampling
  a.k.a.: transformation method, inverse CDF sampling
- rejection sampling
- Markov chain Monte Carlo (Metropolis or Metropolis-Hastings)

Inverse transform sampling

Also known as:

- inverse probability integral transform,
- inverse transformation method
- Smirnov transform
- inverse CDF sampling

Numerical Recipes (+ previous versions of this module) calls this "Transformation method"

Basic idea:
given a uniform random variate X, transform it, $Y = f(X)$, so that Y has the desired probability distribution.

Inverse transform sampling

If X is uniformly distributed, and $Y = f(X)$, then how is Y distributed?

The probability of finding X in $(x, x + dx)$ must be the same as the probability of finding Y in corresponding $(y, y + dy)$,

$$|P_Y(y)dy| = |P_x(x)dx|$$

$$\implies \quad P_Y(y)|f'(x)dx| = P_x(x)|dx| \qquad \implies \quad P_Y(y) = \frac{P_x(x)}{|f'(x)|}$$

Stochastic variables

Note the difference between X and x:
X is a stochastic variable — takes random values
x is an ordinary variable — the argument of the probability distribution

Inverse transform sampling

Example

$$P_X(x) = \begin{cases} 1 & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$Y = f(X) = -\ln X \quad \Longrightarrow \quad 0 < Y < \infty$$

$$P_Y(y) = \frac{1}{|f'(x)|} = x = e^{-y} \qquad \text{when } P_X(x) \text{ is nonzero}$$

$\Longrightarrow$ Y has the exponential distribution

$$P_Y(y) = \begin{cases} e^{-y} & y > 0 \\ 0 & y < 0 \end{cases}$$

Warning: best to specify probability distributions for the full real line.
Check that $P_Y(y)$ above is normalized.

Inverse transform sampling

Example

$$Y = \sqrt{X} \quad \Longrightarrow \quad P_Y(y) = \frac{1}{1/2\sqrt{x}} = 2\sqrt{x} = 2y$$

when x is nonzero.

Exercise! specify $P_Y(y)$ on the full real line.

Check normalisation.

Obtaining a specific distribution

We want a certain $p(y)$. What is $y = f(x)$ if x is uniform?

$$\frac{1}{f'(x)} = \frac{dx}{dy} = p(y) \quad \Longrightarrow \quad dx = p(y)dy$$

$$\Longrightarrow \quad x = \int_{-\infty}^{y} p(z)dz \equiv \mathcal{C}(y)$$

Inverting this gives us: $y(x) = \mathcal{C}^{-1}(x)$

x is uniformly distributed.

What transformation $y = f(x)$ will provide variable y with distribution $p(y)$?

$$\mathcal{C}(y) = \int_{-\infty}^{y} p(z)dz \qquad\qquad y = f(x) = \mathcal{C}^{-1}(x)$$

Inverse transform sampling

x is uniformly distributed.

What transformation $y = f(x)$ will provide variable y with distribution $p(y)$?

$$\mathcal{C}(y) = \int_{-\infty}^{y} p(z)dz \qquad\qquad y = f(x) = \mathcal{C}^{-1}(x)$$

$\mathcal{C}(y)$ is the cumulative distribution function (CDF) of desired distribution.

Hence the name inverse CDF sampling

Inverse transform sampling

x is uniformly distributed.

What transformation $y = f(x)$ will provide variable y with distribution $p(y)$?

$$\mathcal{C}(y) = \int_{-\infty}^{y} p(z)dz \qquad y = f(x) = \mathcal{C}^{-1}(x)$$

We can find the transformation function if

1. we can integrate our distribution $\rightarrow$ cumulative distribution $\mathcal{C}(y)$
2. we can invert the cumulative distribution function

analytically

Shifting and scaling

Transforming variables is useful for shifting and scaling distributions:

$$y = x/a + b \quad \Longrightarrow \quad P_y(y) = aP_x(x) = aP_x\big(a(y-b)\big)$$

Example

X is gaussian with average 0 and variance 1.
We want Y to be gaussian with average μ and variance σ^2,

$$P_Y(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(y-\mu)^2/2\sigma^2}$$

We achieve this by $Y = \sigma X + \mu$

Intuitively:

- Multiplying by σ stretches or squeezes the distribution
- Adding μ shifts everything to the left or right

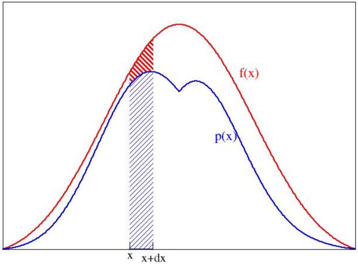
Rejection sampling

What if we cannot integrate or invert?

We want to generate random numbers distributed according to $p(x)$, given a prng distributed as $f_0(x)$.

Rescale $f_0(x)$: $f(x) = Af_0(x)$, so that $f(x) > p(x)$ everywhere.

$$[f(x) \text{ is not normalized} \implies \text{not a pdf}]$$



Idea: Select points under $f(x)$ curve, reject those in red shaded area

The ratio of areas is $p(x)/f(x)$

Rejection sampling

Implementation

1. Pick number X according to distribution $\frac{1}{A} f(x)$, where $A = \int_{-\infty}^{\infty} f(x) dx$
2. Accept X as your random number with probability $p(X)/f(X)$. i.e., reject X with probability $1 - p(X)/f(X)$.

Note:

- Given X, how to accept with probability $p(X)/f(X)$?
  - Use auxiliary random variable ξ: pick random uniform $\xi \in [0, 1]$
  - If $\xi < p(X)/f(X)$ then accept X as your random number else reject X
- Store each accepted value in a list/array.
- If $p(x)$ is normalised, $f(x)$ is *not* normalised; $\dfrac{1}{A} f(x) = f_0(x)$ is.
- Efficiency depends on A — choose A as small as possible while still satisfying $f(x) > p(x)$ everywhere.

Rejection sampling

Simplest version: $f(x) = \text{const} = \sup p(x)$
— just choose a uniform random number $X \in \langle x_{\text{min}}, x_{\text{max}} \rangle$

- will not work when X is unbounded
- can have very high rejection rate for peaked distributions

Variant: cover area with rectangles (+ exponential tail)
$\rightarrow$ ziggurat algorithm, common for gaussian-distributed prng's

Gaussian and exponential distributions are often useful covering functions

Rejection sampling

Example

Generate a pseudo-random number with the distribution

$$p(x) \propto \frac{e^{-x}}{1+x^2}\,, \quad x > 0\,,$$

assuming we already have a generator for the exponential distribution.

Algorithm:

1. Generate an exponentially distributed number z.
2. Generate a standard uniform deviate u.
3. If $u < 1/(1+z^2)$, set $x = z$, otherwise go back to 1.
4. Repeat this to generate as many numbers x as you require.

Rejection sampling

Numerical Recipes describes the case where the covering function itself must be generated using inverse transform sampling:

If $f_0(x)$ must be sampled by inverse CDF sampling

Alogorithm combining inverse transform sampling and rejection sampling:

1. Pick uniform $Z \in \langle 0, A \rangle$; $A = \int_{-\infty}^{\infty} f(x)dx$
2. Find $X = F^{-1}(Z)$ where $F(y) = \int_{-\infty}^{y} f(x)dx$
3. Pick random uniform $Y \in \langle 0, 1 \rangle$
4. If $Y < p(X)/f(X)$ then accept X as your random number
   else reject X and try again

Summary

- Random numbers are widely used in computational physics
- Good pseudo-random number generators exist, but check before using an inbuilt generator for serious business!
- Inverse transform sampling:
  - Obtain new distribution from old analytically
  - Only works for functions where the integral can be obtained and inverted analytically
- Rejection sampling
  - Can be used for any distribution
  - Pick random numbers distributed under curve $f(x) \geq p(x)$
  - Accept numbers with probability $p(x)/f(x)$.
  - Similar to Monte Carlo integration (next)