Lecture 2: Direct Sampling Methods

Deniz Akyildiz

MATH60047/70047 – Stochastic Simulation

October 11, 2022

Imperial College London

Lecture summary

Background

Pseudo-random number generation

Uniform random number generation

Direct sampling for other distributions

We will denote densities here with $p(\cdot)$. For generic densities, we use the notation $X \sim p(x)$, **not** $p_X(x)$ (unless necessary).

In this context, $Y \sim p(y)$ will denote *another density*. Sometimes we will distinguish $p_X(x)$ and $p_Y(y)$ (especially when we do transformation of r.v.s).

Why? When we go into Bayesian inference, this makes it slightly tedious to write things down.

Recall that $p(x)$ is a *function* that integrates to one:

$$\int p(x)\mathrm{d}x = 1.$$

A probability measure is when we write $\mathbb{P}$. We won't need the notion throughout the course.

But let's clarify one thing:

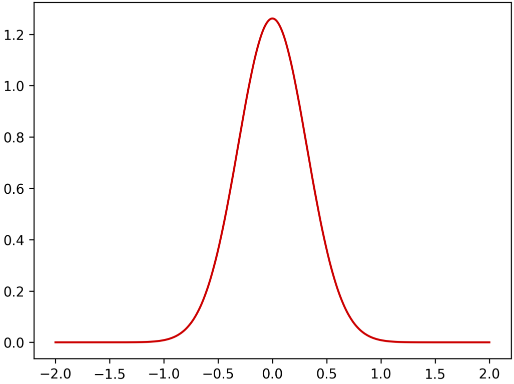
$$\mathbb{P}(x_1 \leq X \leq x_2) = \mathbb{P}(X \in [x_1, x_2]) = \int_{x_1}^{x_2} p(x)\mathrm{d}x.$$

$\mathbb{P}(\cdot)$ *measures* sets (intervals etc.) using the density.

What is the probability of a point for a continuous variable X?

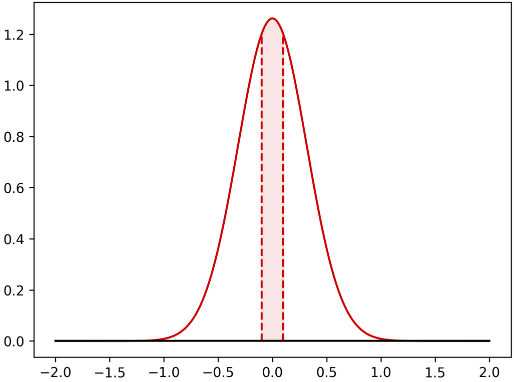
$$\mathbb{P}(X = c) = ?$$

Is it $p(c)$?



$$p(0) = 1.2615$$

$p(0)$ is **not** the probability of $X = 0$. For continuous variables, probability only makes sense on intervals (or sets in higher dimensions).



$$\mathbb{P}(-0.1 \leq X \leq 0.1) = \int_{-0.1}^{0.1} p(x)\mathrm{d}x = 0.248$$

So pointwise evaluations *could be* bigger than 1 in continuous case. Don't get confused!

For discrete sets, we denote the state-space $\mathsf{S} = \{s_1, \dots, s_K\}$ for K possibilities.

- $K = 6$ for a die and $s_1 = 1, \dots, s_6 = 6$.

Probability mass function:

$$p(s_k) \in [0, 1]$$

and $\sum_{k=1}^{K} p(s_k) = 1$.

Discrete case is easy:

$$\mathbb{P}(X = s_k) = p(s_k).$$

The CDF is defined as (for continuous variables)

$$F_X(x) = \mathbb{P}(X \leq x) = \int_{-\infty}^{x} p(x')\mathrm{d}x'.$$

or for discrete variables

$$F(k) = \sum_{i=-\infty}^{k} p(s_i)$$

What are pseudo-random numbers?

Definition 1

A sequence of pseudo-random numbers $u_1, \ldots, u_n$ is a deterministic sequence of numbers with *good enough* statistical properties that match the distribution we want to simulate from.

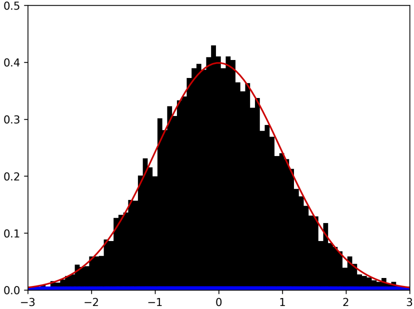
Good, but how do we test "statistical properties"?

Say you are given a sequence of numbers `x` (an `np.array`).

- Check moments: compute the mean (`np.mean(x)`) or the variance (`np.var(x)`) of the sample and check if they match the parameters
- Check histogram against the density (if you know it)
  - `plt.hist(x, bins=100, density=True)`

More on this later, but for now these will be enough.

What are pseudo-random numbers?



$$X^{(i)} \sim p(x)$$

What are pseudo-random numbers?

Why do we need them?

It is (literally) impossible to generate genuinely random numbers on computers.

- You can flip a coin every time you need a binary number
  - Is it really unbiased though?[1]
- Throw a die

What other things can give you a truly random number?

- You can use `www.random.org`
- On a computer
  - Try to measure some inner thermal noise (of circuits)
  - Measure atmospheric noise

As you can see, these are not very practical.

[1] Diaconis, P., Holmes, S., & Montgomery, R. (2007). Dynamical bias in the coin toss. *SIAM review*, 49(2), 211-235.

What are pseudo-random numbers?

Why do we need them?

If we want to simulate randomness, we need to obtain a way that is

- Repeatable
- Cheap

It has become an entire research topic to design *deterministic* algorithms which gives samples that match the desired characteristics.

We will start from the simplest: The uniform distribution.

Uniform pseudo-random numbers

The technology that underlies modern civilization

The key to simulate many (many) other random variables is to be able to simulate uniform random numbers.

We denote the task

$$U \sim \mathsf{Unif}(u; 0, 1).$$

More precisely

$$U \sim p(u) = 1 \quad \text{for } 0 \leq u \leq 1.$$

We will look into the standard way of doing it:

- Linear congruential random number generators

These methods are based on generating a *deterministic linear recursion* with a careful design.

Uniform pseudo-random numbers

The technology that underlies modern civilization

Linear congruential generators (LCGs from now on) are based on simulating a recursion:

$$x_{n+1} \equiv a x_n + b \qquad \pmod{m}$$

where x_0 is the **seed**, m is the **modulus**, b is the **shift**, and a is the **multiplier**.

- m is an integer
- $x_0, a, b \in \{0, \ldots, m-1\}$.

Given $x_n \in \{0, \ldots, m-1\}$, we generate the uniform random numbers

$$u_n = \frac{x_n}{m} \in [0, 1) \qquad \forall n.$$

Example code (try and make it work!)

```
import numpy as np
import matplotlib.pyplot as plt

def lcg(a, b, m, n, x0):
    x = np.zeros(n)
    u = np.zeros(n)
    x[0] = x0
    u[0] = x0 / m
    for k in range(1, n):
        x[k] = (a * x[k - 1] + b) % m
        u[k] = x[k] / m
    return u
```

A few things to know about LCGs:

- They generate *periodic* sequences.

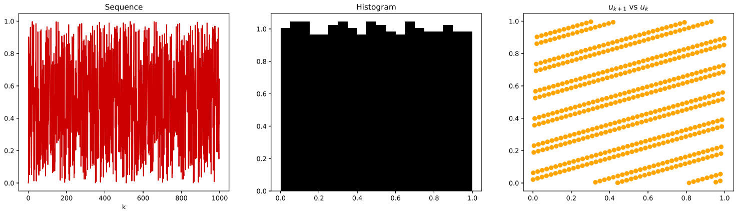


Figure: $m = 2048$, $a = 43$, $b = 0$, $x_0 = 1$.

period $T \leq m$ (m: the modulus).

- Full period: $T = m$

Choice of good parameters rely on some theory, some art.

Uniform pseudo-random numbers

The technology that underlies modern civilization

Wikipedia has a list of parameters for professional implementations:

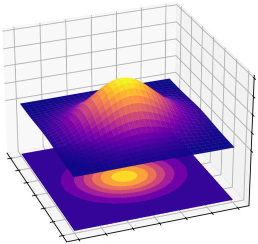
Parameters in common use [edit]

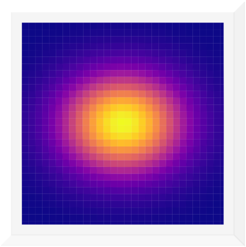
The following table lists the parameters of LCGs in common use, including built-in *rand()* functions in runtime libraries of various compilers. This table is to show popularity, not examples to emulate; *many of these parameters are poor.* Tables of good parameters are available.[10][2]

Source	modulus *m*	multiplier *a*	increment *c*	output bits of seed in *rand()* or *Random(L)*
ZX81	$2^{16}+1$	75	74	
Numerical Recipes from the "quick and dirty generators" list, Chapter 7.1, Eq. 7.1.6 parameters from Knuth and H. W. Lewis	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 in *rand()*, 30..0 in *lrand()*
glibc (used by GCC)[17]	2^{31}	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++[18] C90, C99, C11: Suggestion in the ISO/IEC 9899,[19] C17	2^{31}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 of *(seed × L)*
Turbo Pascal	2^{32}	134775813 (8088405_{16})	1	
Microsoft Visual/Quick C/C++	2^{32}	214013 ($343FD_{16}$)	2531011 ($269EC3_{16}$)	bits 30..16
Microsoft Visual Basic (6 and earlier)[20]	2^{24}	1140671485 ($43FD43FD_{16}$)	12820163 ($C39EC3_{16}$)	
RtlUniform from Native API[21]	$2^{31}-1$	2147483629 ($7FFFFFED_{16}$)	2147483587 ($7FFFFFC3_{16}$)	
Apple CarbonLib, C++11's				

from: `https://en.wikipedia.org/wiki/Linear_congruential_generator`

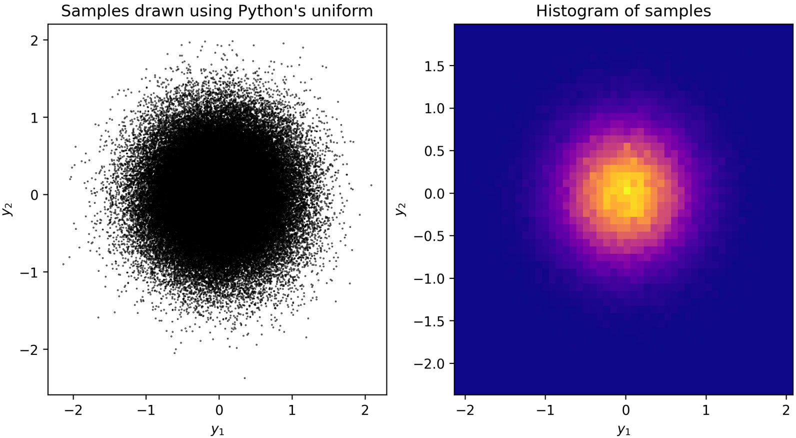
Why professional implementation? Consider this Gaussian:



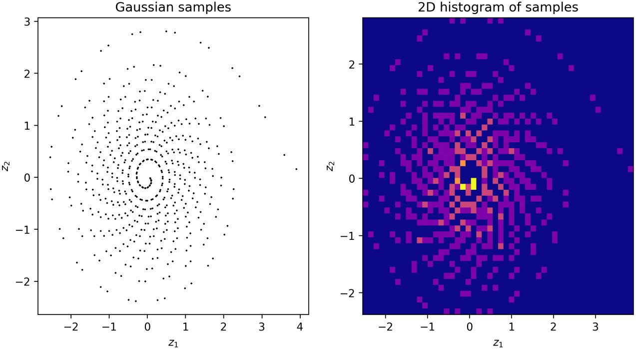


Uniform pseudo-random numbers

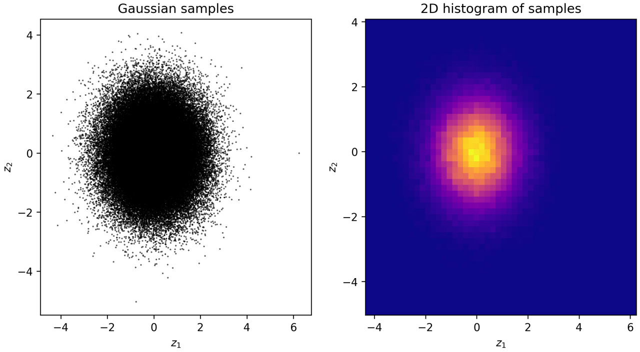
The technology that underlies modern civilization



Samples drawn using the random number generator I showed before:



Samples drawn using the second sampler on Wiki:



Going forward, we will mostly assume that we will have access to a uniform random number generator.

- In assignments, we won't ask you to implement LCG.
- When asked for $U \sim \text{Unif}(0, 1)$, you can instead use

```
np.random.uniform(0, 1, n)
```

  where `n` is the number of samples you want to draw.

Next up: Sampling from general $p(x)$

Exact sampling of distributions

How to simulate from any $p(x)$?

Simulating from a given $p(x)$ is an endless research area (simulation, sampling, generative models) and still flourishing.

We will start by describing some general methods to sample from more general distributions.

This lecture (direct sampling):

- Inversion
- Transformation

THE INSTITUTE FOR ADVANCED STUDY
SCHOOL OF MATHEMATICS
PRINCETON, NEW JERSEY

May 21, 1947

Mr. Stan Ulam
Post Office Box 1663
Santa Fe
New Mexico

Dear Stan:

Thanks for your letter of the 19th. I need not tell you that Klari and I are looking forward to the trip and visit at Los Alamos this Summer. I have already received the necessary papers from Carson Mark. I filled out and returned mine yesterday; Klari's will follow today.

I am very glad that preparations for the random numbers work are to begin soon. In this connection, I would like to mention this: Assume that you have several random number distributions, each equidistributed in $0, 1$: (x^i), (y^i), (z^i), ... Assume that you want one with the distribution function (density) $f(\xi)\,d\xi$: (ξ^i) . One way to form it is to form the cumulative distribution function: $g(\xi) = \int^{\xi} f(\xi)\,d\xi$ to invert it $h(x) = \xi \rightleftarrows x = g(\xi)$, and to form $\xi^i = h(x^i)$ with this $h(x)$, or some approximant polynomial. This is, as I see, the method that you have in mind.

The inversion technique is based on the following theorem:

Theorem 2

Consider a random variable X with a CDF F_X. Then the random variable

$$Y = F_X(X)$$

is uniformly distributed.

Proof. Consider any continuous random variable X, we define $Y = F_X(X)$. For $y \in [0, 1]$,

$$\begin{aligned} F_Y(y) &= \mathbb{P}(Y \le y) \\ &= \mathbb{P}(F_X(X) \le y) \\ &= \mathbb{P}(X \le F_X^{-1}(y)) \qquad \text{Why?} \\ &= F_X(F_X^{-1}(y)) \\ &= y, \end{aligned}$$

which is the CDF of the standard uniform distribution.

Note that above result is written for the case where F_X^{-1} exists, i.e., the CDF is continuous. If this is not the case, one can define the generalised inverse function,

$$F_X^{-}(u) = \min\{x : F_X(x) \geq u\}.$$

Going back to statement: If $X \sim p(x)$ with $F_X(x) = \int_{-\infty}^{x} p(x')\mathrm{d}x'$, we know that

$$Y = F_X(X)$$

is uniform. Then this suggests *inverting* this process:

- Sample $U \sim$ Unif$([0, 1])$,
- Draw $X = F_X^{-1}(U)$.

Of course, this is limited to the cases where we *can* invert the CDF.

Let us consider some examples.

The most generic one is the discrete (categorical) distribution. For $K \geq 1$ (integer), define K states $s_1, \ldots, s_K$ where

$$p(s_k) \in [0, 1] \qquad \text{where} \qquad \sum_{k=1}^{K} p(s_k) = 1.$$

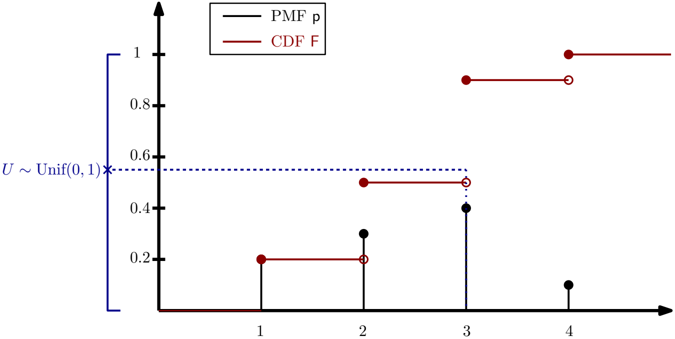
Simpler than it looks, consider the die:

$$s_k = k \text{ (the face of die)}$$

and their probabilities

$$p(s_k) = 1/6.$$

How does the sampling work?



- Draw $U \sim \mathrm{Unif}(0,1)$
- Choose $F_X^-(u) = \min\{x : F_X(x) \geq u\}$ generic for discrete dist.

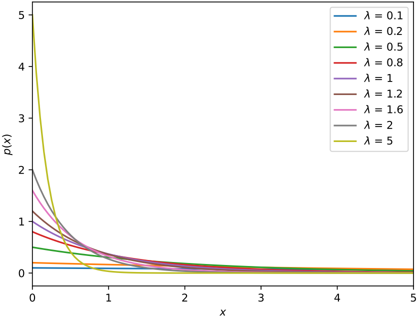
Some starters:

```
import numpy as np
import matplotlib.pyplot as plt

w = np.array([0.2, 0.3, 0.4, 0.1]) # pmf
s = np.array([1, 2, 3, 4])          # support (states)

def discrete_cdf(w):
    return np.cumsum(w)

cw = discrete_cdf(w)

def plot_discrete_cdf(w, cw):
    fig, ax = plt.subplots(1, 2, figsize=(20, 5))
    ax[0].stem(s, w)
    ax[1].plot(s, cw, 'o-', drawstyle='steps-post')
    plt.show()

plot_discrete_cdf(w, cw)
```

The exponential density

$$p(x) = \mathsf{Exp}(x; \lambda) = \lambda e^{-\lambda x}.$$

for $x \geq 0$. Otherwise $p(x) = 0$.



Exact sampling of distributions

Inversion: Exponential distribution

$$p(x) = \mathsf{Exp}(x;\lambda) = \lambda e^{-\lambda x}.$$

We calculate the CDF

$$\begin{aligned} F_X(x) &= \int_0^x p(x')\mathrm{d}x', \\ &= \lambda \int_0^x e^{-\lambda x'}\mathrm{d}x', \\ &= \lambda \left[-\frac{1}{\lambda} e^{-\lambda x'} \right]_{x'=0}^{x} \\ &= 1 - e^{-\lambda x}. \end{aligned}$$

Calculate the reverse?

Deriving the inverse:

$$
\begin{aligned}
u &= 1 - e^{-\lambda x} \\
\implies x &= -\frac{1}{\lambda} \log(1-u) \\
\implies F_X^{-1}(u) &= -\lambda^{-1} \log(1-u).
\end{aligned}
$$

So what is the algorithm?

- Generate $u_i \sim \text{Unif}([0,1])$
- $x_i = -\lambda^{-1} \log(1 - u_i)$.

Implement this (exercises).

Consider the Cauchy distribution

$$p(x) = \frac{1}{\pi(1 + x^2)}.$$

Given

$$p(x) = \frac{1}{\pi(1 + x^2)},$$

the CDF is given by:

$$F_X(x) = \int_{-\infty}^{x} p(x')\mathrm{d}x.$$

How do we compute this integral? (at the end, if time permits)

We obtain:

$$F_X(x) = \frac{1}{2} + \pi^{-1} \tan^{-1} x.$$

The inverse is straightforward:

$$F_X^{-1}(u) = \tan\left[\pi\left(u - \frac{1}{2}\right)\right].$$

Another one: Given

$$p(k) = \mathsf{Pois}(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}.$$

Design a sampling scheme using inversion.

The CDF is given:

$$F(k) = \sum_{i=1}^{k} \mathsf{Pois}(k;\lambda) = e^{-\lambda} \sum_{i=0}^{k} \frac{\lambda^i}{i!}.$$

How do you use inversion? Recall the discrete example.

Wait until it hits:

- Draw $U \sim$ Unif$([0,1])$
- $$P = \min\left\{k \in \mathbb{N} : U \leq e^{-\lambda} \sum_{i=0}^{k} \frac{\lambda^k}{k!}\right\}.$$

  This is done via a `while` statement.

Exact sampling of distributions

Inversion: Is Gaussian possible?

Let $p(x) = \mathcal{N}(x; \mu, \sigma^2)$. Can we use inversion?

No. F_X^{-1} is impossible to compute and hard to approximate.

Inversion is a special case of a general method called *transformation method*.

Transformation method:

- Sample $U_i \sim$ Unif$(u; 0, 1)$
- Transform: $X_i = g(U_i)$.

Inversion is just setting $g = F_X^{-1}$.

The simplest example can be seen from sampling a uniform on $[a, b]$ using a uniform on $[0, 1]$.

- Draw $U_i \sim \mathsf{Unif}(u; 0, 1)$
- Set $X_i = g(U_i) = (b - a)U_i + a$

then $X_i \sim \mathsf{Unif}(x; a, b)$.

For general g, how do we compute the density?

If $X \sim p_X(x)$ and $Y = g(X)$, what is $p_Y(y)$?

$$p_Y(y) = p_X(g^{-1}(y)) \left| J_{g^{-1}}(y) \right|$$

where J is the Jacobian of the inverse mapping g^{-1}, evaluated at y.

Theorem 3 (Box-Müller method)

Let X_1, X_2 *be independent r.v.'s respectively where*

$$X_1 \sim \mathsf{Exp}\left(\frac{1}{2}\right),$$

$$X_2 \sim \mathsf{Unif}(0, 2\pi).$$

Then $Y_1 = \sqrt{X_1}\cos X_2$ *and* $Y_2 = \sqrt{X_1}\sin X_2$ *are independent and* $\mathcal{N}(0,1)$*-distributed.*

A transformation method with

$$(y_1, y_2) = g(x_1, x_2) = (\sqrt{x_1}\cos x_2, \sqrt{x_1}\sin x_2).$$

Proof. How to compute the density $p(y_1, y_2)$? Use the transformation of random variables

$$p_{y_1,y_2}(y_1, y_2) = p_{x_1,x_2}(g^{-1}(y_1, y_2)) \left| J_{g^{-1}}(y_1, y_2) \right|$$

where $J_{g^{-1}}$ is the Jacobian of the inverse. What is g^{-1}?

Recall $(y_1, y_2) = g(x_1, x_2) = (\sqrt{x_1} \cos x_2, \sqrt{x_1} \sin x_2)$. We have

$$x_1 = y_1^2 + y_2^2, \qquad \text{as } \cos^2 + \sin^2 = 1.$$

and

$$\frac{\sin x_2}{\cos x_2} = \frac{y_2}{y_1}$$

which leads to

$$x_2 = \arctan(y_2/y_1).$$

Therefore, $g^{-1} : \mathbb{R}^2 \to \mathbb{R}^2$

$$g^{-1}(y_1, y_2) = (g_1^{-1}, g_2^{-1}) = \left(y_1^2 + y_2^2, \arctan(y_2/y_1)\right).$$

Compute the Jacobian

$$\begin{aligned} J_{g^{-1}} &= \begin{bmatrix} \partial g_1^{-1}/\partial y_1 & \partial {g_1}^{-1}/\partial y_2 \\ \partial {g_2}^{-1}/\partial y_1 & \partial g_2^{-1}/\partial y_2 \end{bmatrix} \\ &= \begin{bmatrix} 2y_1 & 2y_2 \\ \frac{1}{1+(y_2/y_1)^2}\frac{-y_2}{y_1^2} & \frac{1}{1+(y_2/y_1)^2}\frac{1}{y_1} \end{bmatrix} \end{aligned}$$

Hence, the determinant is:

$$\left|J_{g^{-1}}\right| = 2.$$

Remember...

$$p_{y_1,y_2}(y_1, y_2) = p_{x_1,x_2}(g^{-1}(y_1, y_2)) \left| J_{g^{-1}}(y_1, y_2) \right|.$$

and

$$g^{-1}(y_1, y_2) = (g_1^{-1}, g_2^{-1}) = \left(y_1^2 + y_2^2, \arctan(y_2/y_1) \right).$$

Let's write it out

$$\begin{aligned} p_{y_1,y_2}(y_1, y_2) &= \mathsf{Exp}(g_1^{-1}; 1/2)\mathsf{Unif}(g_2^{-1}; 0, 2\pi) \left| J_{g^{-1}} \right| \\ &= \frac{1}{2} e^{-\frac{1}{2}(y_1^2 + y_2^2)} \frac{1}{2\pi} 2 \\ &= \mathcal{N}(y_1; 0, 1)\mathcal{N}(y_2; 0, 1). \end{aligned}$$

■

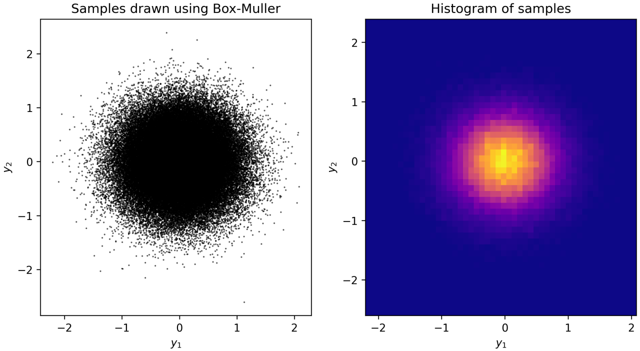
Exact sampling of distributions

Transformation method: Sampling a Gaussian (Box and Müller, 1958)

```
import matplotlib.pyplot as plt
import numpy as np

def box_muller(n):

    # Your code

    return y_1, y_2

n = 100000
y_1, y_2 = box_muller(n)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].scatter(y_1, y_2, s=0.1, c='k')
axs[1].hist2d(y_1, y_2, bins=50, cmap='plasma')
plt.show()
```

Exact sampling of distributions

Transformation method: Sampling a Gaussian (Box and Müller, 1958)



Define the problem. We draw

$$r \sim \mathsf{Unif}(0, 1),$$
$$\theta \sim \mathsf{Unif}(0, 2\pi).$$

How to get uniform points on a circle?

Let's have a little poll.

Define the problem. We draw

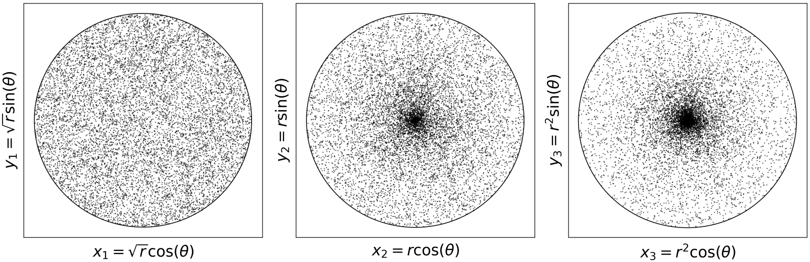
$$r \sim \mathsf{Unif}(0,1),$$
$$\theta \sim \mathsf{Unif}(0,2\pi).$$

How to get uniform points on a circle? Which one of the following?

- $x_1 = \sqrt{r}\cos\theta$ and $x_2 = \sqrt{r}\sin\theta$
- $x_1 = r\cos\theta$ and $x_2 = r\sin\theta$
- $x_1 = r^2\cos\theta$ and $x_2 = r^2\sin\theta$

Exact sampling of distributions

Transformation method: A puzzle



The transformation is *exactly* the same as Box-Müller (exercise).

It is possible to prove that

$$p_{x_1,x_2}(x_1, x_2) = \begin{cases} \frac{1}{\pi} & \text{if } x_1^2 + x_2^2 < 1 \\ 0 & \text{otherwise,} \end{cases}$$

which is the uniform distribution on the circle.

In class exercise: Prove this result.

$$
\begin{aligned}
p_{x_1,x_2}(x_1,x_2) &= \mathsf{Unif}(g_1^{-1};0,1)\mathsf{Unif}(g_2^{-1};0,2\pi)\left|J_{g^{-1}}\right| \\
&= \frac{1}{2\pi}2 \quad \text{for } x_1^2+x_2^2<1, \\
&= \frac{1}{\pi} \quad \text{for } x_1^2+x_2^2<1.
\end{aligned}
$$

If $X \sim \mathcal{N}(0,1)$, derive the distribution of

$$Y = \sigma X + \mu.$$

The inverse transform is:

$$g^{-1}(y) = \frac{y - \mu}{\sigma}.$$

Therefore,

$$p_Y(y) = p_X(g^{-1}(y)) \left| \frac{\mathrm{d}g^{-1}}{\mathrm{d}y} \right|,$$

which is

$$p_Y(y) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{(y-\mu)^2}{2\sigma^2}) \frac{1}{\sigma} = \mathcal{N}(\mu, \sigma^2)$$

Next step?

Follow von Neumann

... polynomial. This is, as I see, the method that you have in mind.

An alternative, which works if ξ and all values of $f(\xi)$ lie in 0, 1, is this: Scan pairs x^i, y^i and use or reject x^i, y^i according to whether $y^i \leq f(x^i)$ or not. In the first case, put $\xi^j = x^i$ in the second case form no ξ^j at that step.

The second method may occasionally be better ...

Rejection sampling

Box, GEP and Mervin E Müller (1958). "A Note on the Generation of Random Normal Deviates". In: *The Annals of Mathematical Statistics* 29.2, pp. 610–611.