Computational Physics – Lecture 21

(4 May 2020)

Recap

In the previous lecture, we learnt how to obtain uniformly distributed random numbers.

Today's plan

In today's class, we want to learn how to obtain non-uniformly distributed random numbers.

Seeds

As we saw in the case of the linear congruential generator, random number generators need a *seed*

The seed tells the generator where to start its pseudorandom sequence.

In the case of the linear congruential generator, the seed is the first number in the output sequency.

Seeds

For example, for the seed value `1`, our linear congruential number gives the sequence

```
          1
 1015568748
 1586005467
 2165703038
        ...
```

For the seed value `42`, we get

```
         42
 1083814273
  378494188
  2479403867
  ...
```

Seeds

But the seed need not be part of the output sequence at all.

For example, in Python, `random.randrange(100)` will give us a random integer between 0 and 99 using the Mersenne Twister generator.

We can set the seed value of this generator using the function `random.seed`.

For a seed value of `1`, I get `17, 72, 97, ...`, which does not start with `1`.

Seeds

While using library functions, should you choose a seed?

If you do not use a seed, Python's Mersene Twister generator will take an appropriate randomly chosen seed for you.

But if you do use a seed, your programme will get the same randon number sequence every time it runs.

This is useful while writing programmes.

Seeds

Sometimes it can happen that you are developing a programme that fails for a random number sequence.

But if you run the programme again it runs OK because it now gets a different random number sequence.

In such cases, it helps if you control the random number sequence by setting seeds.

You can choose to avoid seeding the generator once your programme development is complete.

Seeds

Another important reason to set a seed is to make stochastic computations reproducible.

But how do ensure that we have chosen a good seed?

By choosing a good generator. And by testing the random number sequence in your code.

Non-uniform deviates

For a uniform deviate, the probability of generating a number between x and $x + dx$ is

$$p(x)dx = \begin{cases} dx & \text{if } 0 < x < 1 \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where

$$\int_{-\infty}^{\infty} \mathrm{d}x \; p(x) = 1. \tag{2}$$

But we are often interested in *non-uniform* deviates—random numbers that follow a non-uniform probability distribution function.

Non-uniform deviates

For example, the time taken by a particle to decay follows an exponential distribution $p(x) \propto \exp(-x)dx$.

Noise, i.e., random errors in measurements, follows a Gaussian distribution $p(x) \propto \exp(-x^2)dx$.

The density of randomly chosen points on a flat surface follows a Poisson distribution.

Transformation method

One way to generate non-uniform deviates is by using the transformation method.

Suppose I have a uniform deviate x.

What is the distribution of a transformation $y(x)$?

It is just

$$p(y)dy = p(x)\left|\frac{\mathrm{d}x}{\mathrm{d}y}\right|, \tag{3}$$

because $p(x)\mathrm{d}x = p(y)\mathrm{d}y$ ("conservation of measure").

Transformation method

Now we just have to choose $y(x)$ so that $p(y)$ is our desired non-uniform random number distribution.

For example, if I take $y(x)$ to be $-\ln x$, then

$$p(y)dy = p(x)\left|\frac{\mathrm{d}x}{\mathrm{d}y}\right| \tag{4}$$

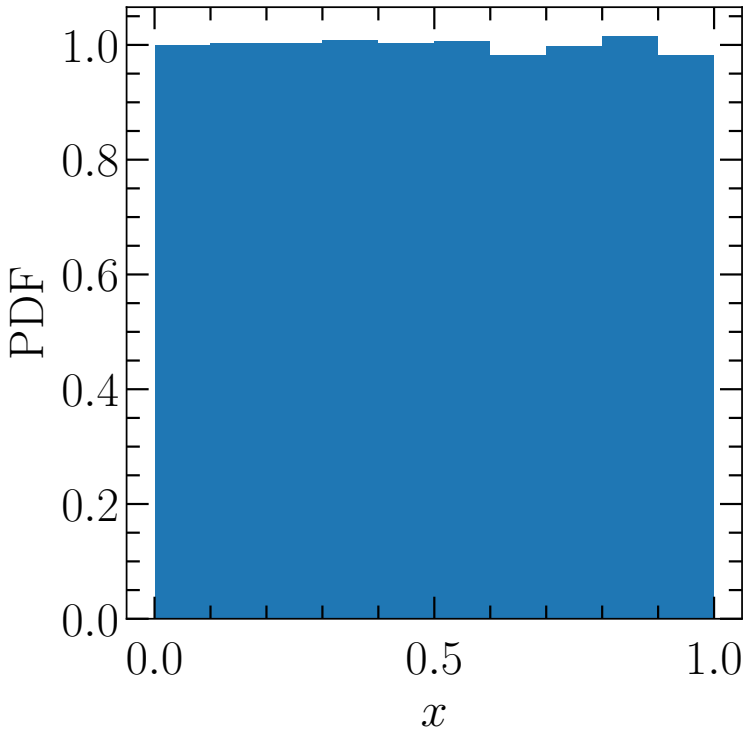
$$= \left|\frac{\mathrm{d}x}{\mathrm{d}y}\right| \tag{5}$$

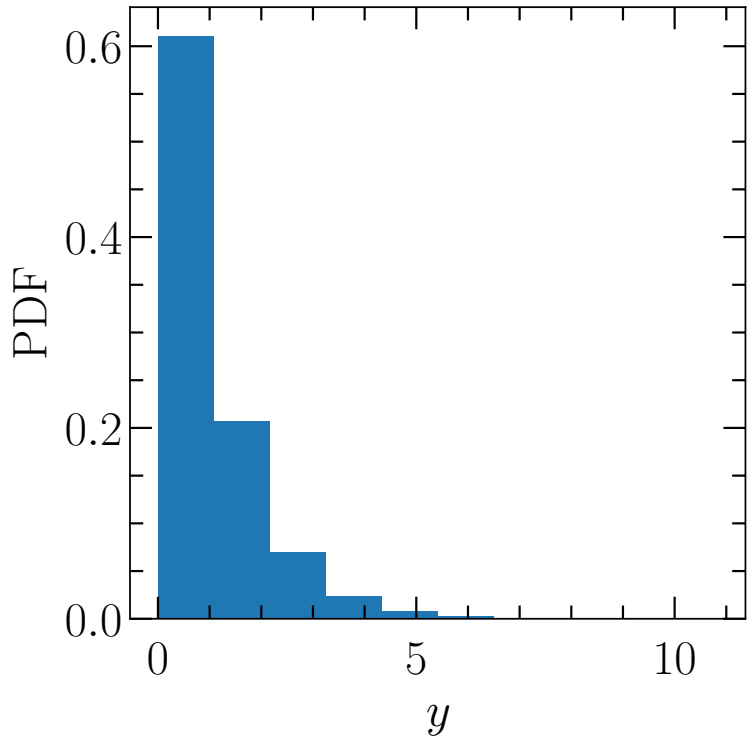
$$= \exp(-y)\,\mathrm{d}y, \tag{6}$$

so that y now has an exponential distribution.

(Recall that the PDF for the exponential distribution is $\lambda e^{-\lambda x}$.)

Transformation method






```
x = np.random.rand(100000)
y = -np.log(x)
```

Transformation method

In general, if we want to have random numbers with a distribution $f(y)$, we need to solve the differential equation

$$\frac{\mathrm{d}x}{\mathrm{d}y} = f(y). \tag{7}$$

The solution of this equation is $x = F(y)$, where

$$F(y) = \int_0^y f(y)\,\mathrm{d}y. \tag{8}$$

So our desired transformation is the inverse of F

$$y(x) = F^{-1}(x). \tag{9}$$

(Calculating F^{-1} is sometimes possible, sometimes impossible.)

Transformation method

We can generalise the transformation method to higher dimensions.

If $x_1, x_2, \ldots$ are random numbers with joint PDF $p(x_1, x_2, \ldots)$ and if $y_1, y_2, \ldots$ are functions of the $x_1, x_2, \ldots$, then the joint PDF of the $y_1, y_2, \ldots$ is given by

$$p(y_1, y_2, \ldots)\mathrm{d}y_1\mathrm{d}y_2 \ldots = p(x_1, x_2, \ldots) \left| \frac{\partial(x_1, x_2, \ldots)}{\partial(y_1, y_2, \ldots)} \right| \mathrm{d}y_1\mathrm{d}y_2 \ldots,$$

where the $|\partial()/\partial()|$ is the Jacobian determinant.

Transformation method

This higher-dimensional transformation method is used to get Normal deviates with

$$p(y)dy = \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy. \tag{10}$$

Consider two uniform deviates x_1 and x_2 and their transformations

$$\begin{aligned} y_1 &= \sqrt{-2\ln(x_1)}\cos(2\pi x_2) \\ y_2 &= \sqrt{-2\ln(x_1)}\sin(2\pi x_2) \end{aligned} \tag{11}$$

Transformation method

In other words,

$$x_1 = \exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right]$$
$$x_2 = \frac{1}{2\pi}\arctan\frac{y_1}{y_2} \tag{12}$$

Now the Jacobian determinant is given by

$$\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} \tag{13}$$

Transformation method

So in our case we get

$$\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} = -\left[\frac{1}{\sqrt{2\pi}} e^{-y_1^2/2} dy\right]\left[\frac{1}{\sqrt{2\pi}} e^{-y_2^2/2} dy\right] \tag{14}$$

Which means that the joint PDF of y_1 and y_2 is just

$$p(y_1, y_2) dy_1 dy_2 = \left[\frac{1}{\sqrt{2\pi}} e^{-y_1^2/2} dy\right]\left[\frac{1}{\sqrt{2\pi}} e^{-y_2^2/2} dy\right] dy_1 dy_2 \tag{15}$$

So y_1 and y_2 are independent and have Normal distributions.

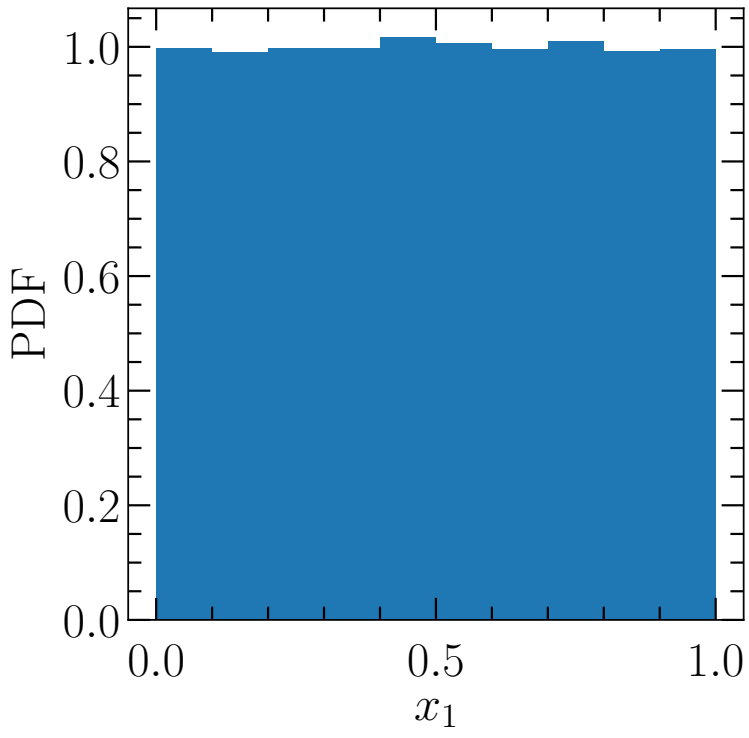
(This special case of the higher-dimensional transformation method is useful because calculating F^{-1} for the Gaussian is hard. This method is called the Box-Muller method.)

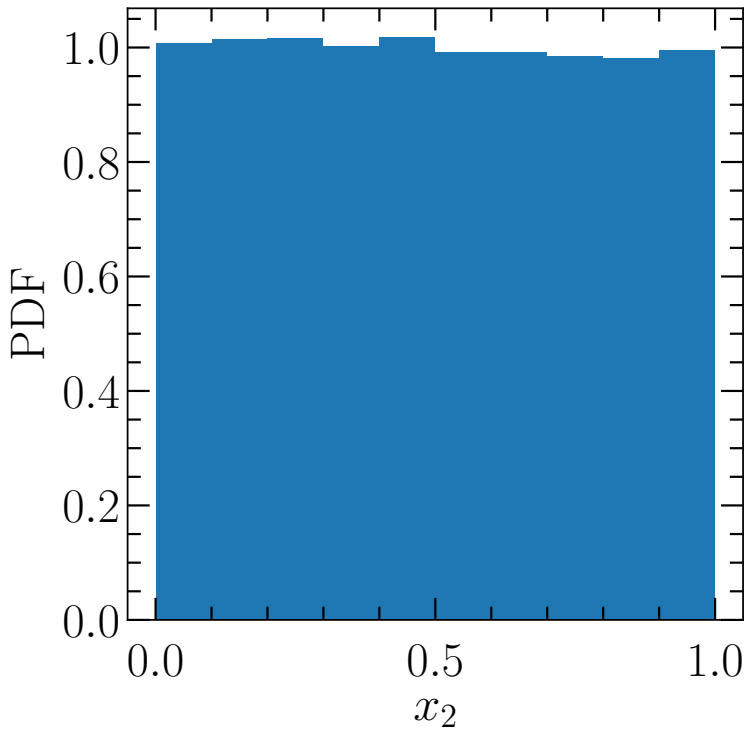
Transformation method

I can code this up as

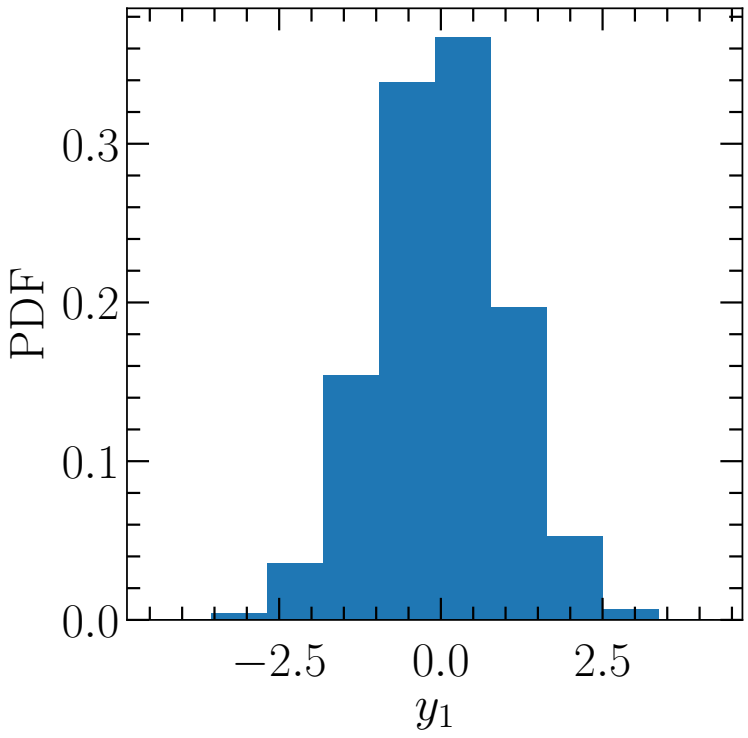
```
x1 = np.random.rand(100000)
x2 = np.random.rand(100000)
y1 = np.sqrt(-2*np.log(x1)) * np.cos(2*np.pi*x2)
y2 = np.sqrt(-2*np.log(x1)) * np.sin(2*np.pi*x2)
```

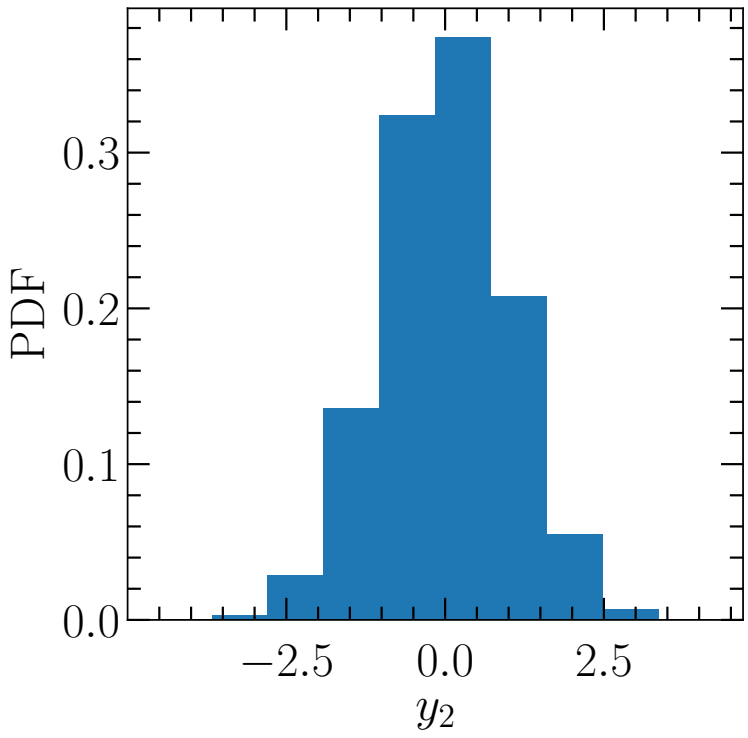
Transformation method





Transformation method





Rejection method

Another method of obtaining non-uniform deviates is the Rejection method.

Simplest incarnation of this method is the biased coin.

Suppose you want to model a coin for which the probability of getting heads is 0.2.

This is like sampling two random integers 0 and 1 but with probability 0.8 and 0.2, respectively.

Rejection method

We can get such non-uniformly distributed numbers by generating a uniform random number between 0 and 1 and calling it heads only if the number is less than 0.2.

For example, something like

```
if np.random.rand() < 0.2:
    print('Heads')
else:
    print('Tails')
```

Rejection method

We can generalise this for more powerful cases.

Suppose we want random numbers fro a distribution $p(x)$.

We can draw a graph of $p(x)$, that is, plot $y = p(x)$ versus x.

Now if we sample random numbers in 2 dimensions with uniform probability under the area of the curve then the x value of the points will have the distribution $p(x)$.

Rejection method

The advantage of this method is that it does not require the inverse cumulative distribution function of $p(x)$.

Now consider a function $f(x)$ for which we *can* use the transformation method and which is such that its graph lies everywhere above the graph of $p(x)$.

Let us call this $f(x)$ a comparison function.

Now we choose random points on 2 dimensions that are uniform in the area under $f(x)$.

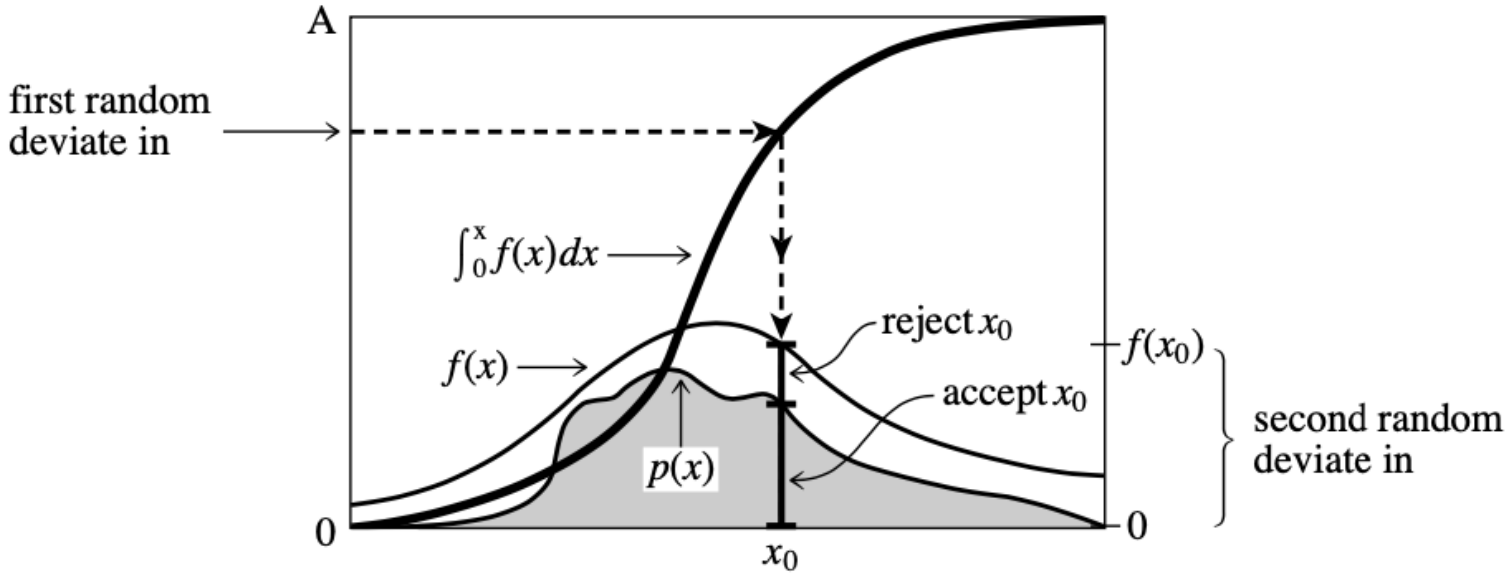
Rejection method

When a random point lies below the original function $p(x)$, we accept it; otherwise, we reject it.

Now the accepted points will be uniformly distributed under $p(x)$ and their x values will have the desired PDF.

How to get the points uniformly distributed under $f(x)$? Just use the transformation method.

Rejection method



(Figure from "Numerical Recipes".)

Uniform distribution in curvilinear spaces

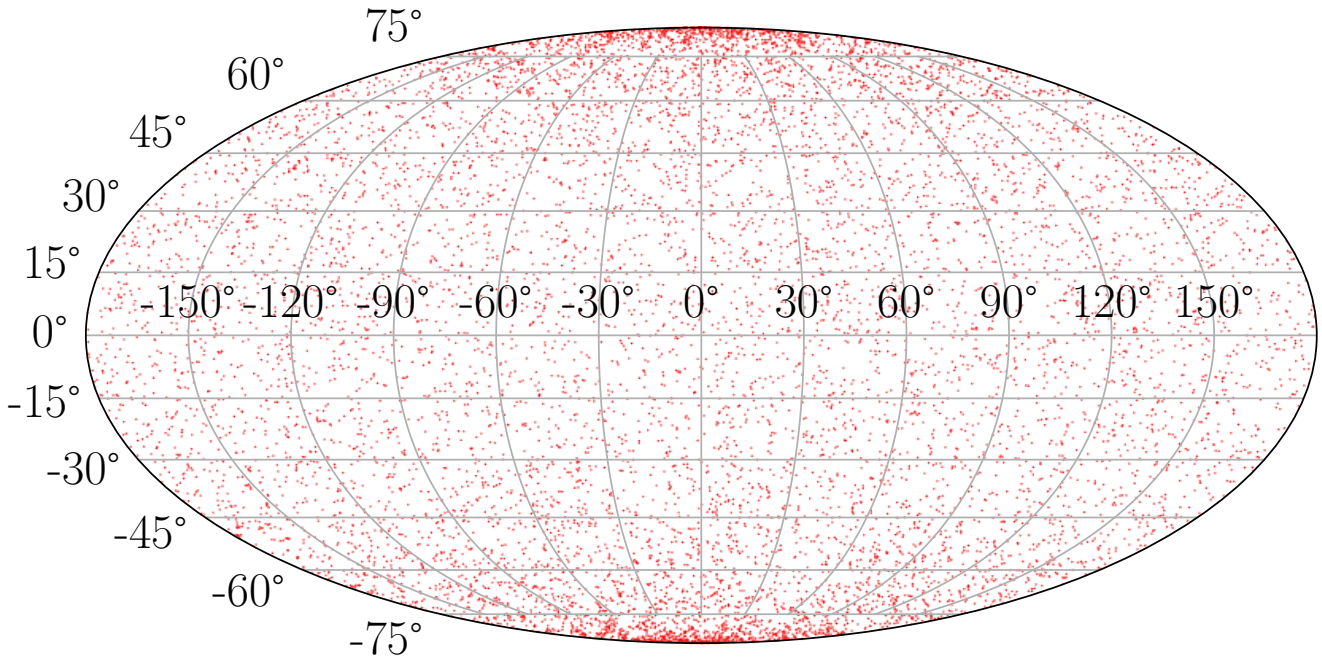
Suppose we want to get two-dimensional uniform random numbers on a unit-sphere.

We need uniform numbers (θ, ϕ) where θ goes from $-\pi/2$ to $\pi/2$ and ϕ goes from $-\pi$ to π.

We could naively do $\phi = 2\pi u - \pi$ and $\theta = \pi v - \pi/2$, where u and v and uniform deviates over $(0, 1)$.

But this would be wrong because the area element has changed!

Uniform distribution in curvilinear spaces



Uniform distribution in curvilinear spaces

What works is

$$\phi = 2\pi u - \pi, \tag{16}$$

and

$$\theta = \cos^{-1}(2v - 1) - \pi/2. \tag{17}$$

Uniform distribution in curvilinear spaces