Computational Physics (PHYS6350)

Lecture 18: Random numbers: Part II

- Non-uniformly distributed random numbers
- Importance sampling

Reference: Chapter 10 of *Computational Physics* by Mark Newman

April 1, 2025

Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu)

Course materials: https://github.com/vlvovch/PHYS6350-ComputationalPhysics/tree/spring2025

Nonuniformly distributed random numbers

In many cases we deal with random numbers ξ that are distributed non-uniformly.

Common examples are:

- Exponential distribution $\rho(x) = e^{-x}$.
- Gaussian distribution $\rho(x) \propto e^{-\frac{x^2}{2\sigma^2}}$.
- Power-law distribution $\rho(x) \propto x^{\alpha}$.
- Arbitrary peaked distributions.

There are two common methods for generating nonuniform random variates.
They both make use of uniformly distributed variates.

- Inverse transform sampling
- Rejection sampling

Inverse transform sampling

The basic idea is that if η is a uniformly distibuted random variable, some function of it, $\xi = f(\eta)$, is not. The idea is to sample η and calculate ξ via this function such that ξ corresponds to a desired probability density $\rho(\xi)$. How to find the function $f(\eta)$?

Without the loss of generality assume that $\xi \in (-\infty, \infty)$ and that $f(\eta)$ maps η to ξ such that $f(0) \to -\infty$. Consider now the cumulative distribution function

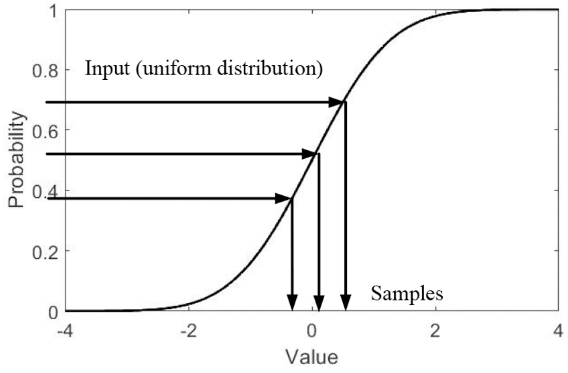
$$G(x) = Pr(\xi < x) = \int_{-\infty}^{x} \rho(\xi) d\xi.$$

It corresponds to the probability that $\eta < y$ where y is such that $x = f(y)$. Since η is uniformly distributed, this probablity equals to y. Therefore,

$$G[x = f(y)] = y,$$

thus

$$f(y) = G^{-1}(y).$$



If we can calculate the inverse of $G^{-1}(y)$ of the cumulative distribution function for ξ, we are good.

Inverse transform sampling

The algorithm follows these steps:

1. Calculate the cumulative distribution function (CDF)

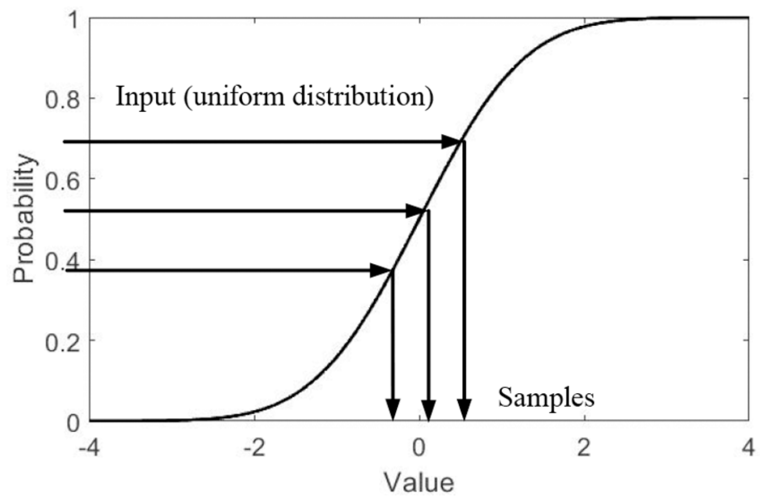
$$G(x) = \int_{-\infty}^{x} \rho(\xi) d\xi$$

2. Find the inverse function $G^{-1}(y)$ as the solution to the equation

$$G(x) = y$$

3. Sample uniformly distributed random variables η and compute ξ using the inverse function:

Challenges: Sometimes, evaluating G(x) and/or $G^{-1}(y)$ explicitly is difficult. In such cases, numerical integration and/or non-linear equation solvers may be required.



Inverse transform sampling: Exponential distribution

Example: Exponential Distribution

1. Recall the radioactive decay process. The time of decay is distributed according to the probability density function:

$$\rho(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}}.$$

2. The cumulative distribution function is given by:

$$F(x) = \int_0^x \frac{1}{\tau} e^{-\frac{t}{\tau}} dt = 1 - e^{-\frac{x}{\tau}}.$$

3. To apply inverse transform sampling, we need to invert F(x) by solving:

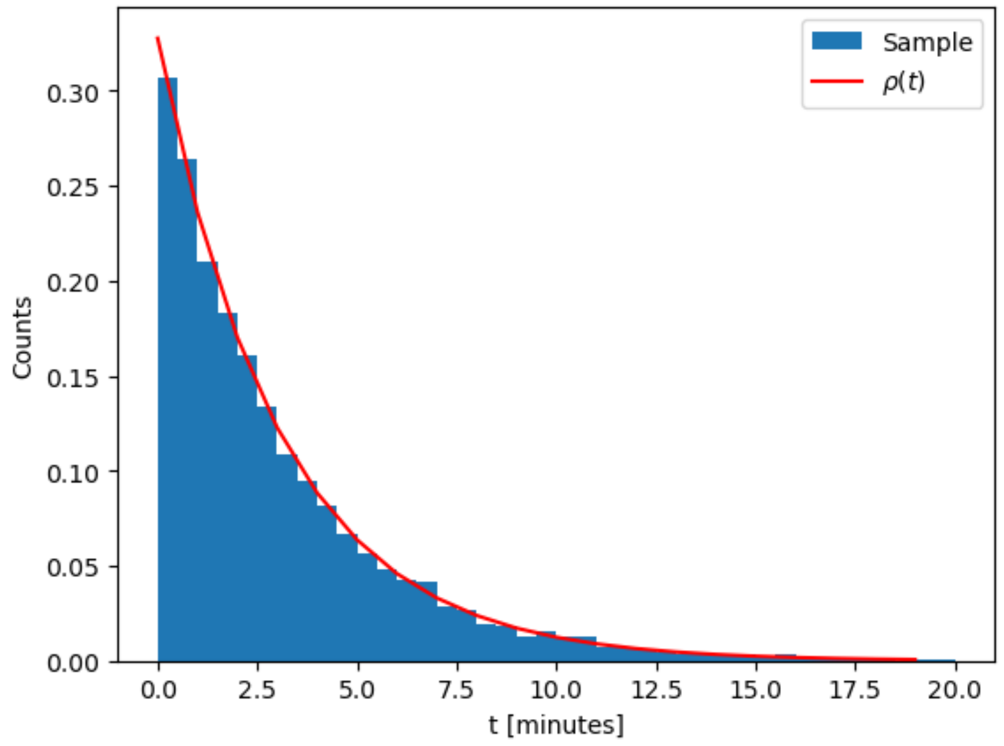
$$1 - e^{-\frac{t}{\tau}} = \eta.$$

4. Solving for t, we obtain:

$$\boxed{t(\eta) = -\tau \ln(1 - \eta).}$$

Sampling radioactive decay time

```python
## Radioactive decay sampler
def sample_tdecay(tau):
    eta = np.random.rand()
    return -tau * np.log(1-eta)


tau = 3.053 # Half-time in minutes
N = 10000   # Number of samples
tdecays = [sample_tdecay(tau) for i in range(N)]

# Show a histogram
plt.xlabel("t [minutes]")
plt.ylabel("Counts")
plt.hist(tdecays, bins = 40, range=(0,20), density=True)
```



Sampling points inside a circle

One way to sample points inside a unit circle is by switching to **polar coordinates**:

$$x = r\cos(\phi), \qquad y = r\sin(\phi),$$

where we sample r $\in [0,1)$ and $\phi \in [0, 2\pi)$.

Naively, one could sample r and ϕ independently from two uniform distributions. Let's see what happens!

```
def sample_xy_naive():
    r = np.random.rand()
    phi = 2 * np.pi * np.random.rand()
    return r*np.cos(phi), r*np.sin(phi)

xplot = []
yplot = []
N = 1000
for i in range(N):
    x, y = sample_xy_naive()
    xplot.append(x)
    yplot.append(y)

plt.plot(xplot,yplot,'o',color='r')
plt.show()
```

Sampling points inside a circle

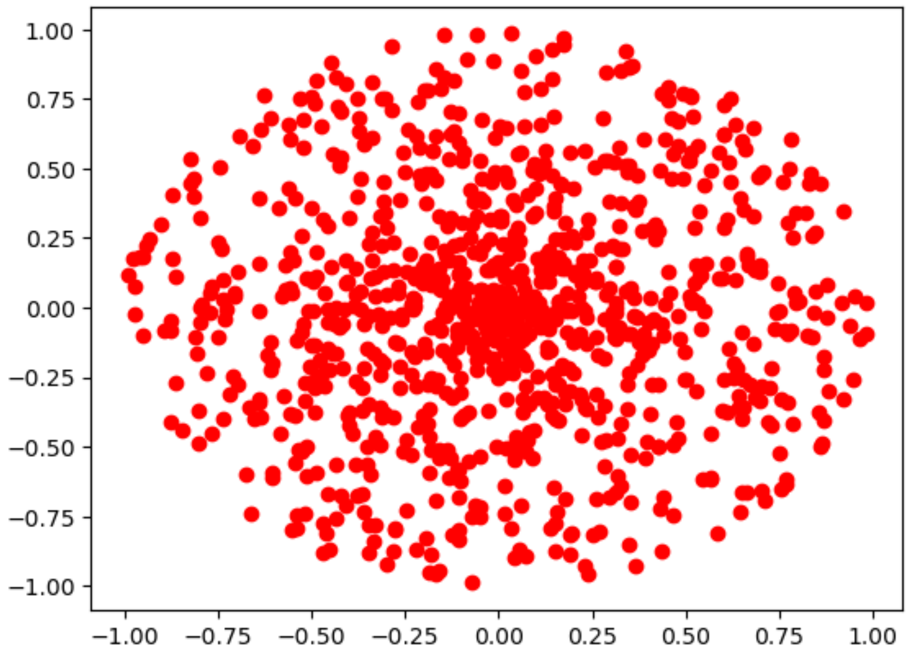
One way to sample points inside a unit circle is by switching to **polar coordinates**:

$$x = r\cos(\phi), \qquad y = r\sin(\phi),$$

where we sample r ∈ [0,1) and ϕ ∈ [0, 2π).

Naively, one could sample r and ϕ independently from two uniform distributions. Let's see what happens!

```
def sample_xy_naive():
    r = np.random.rand()
    phi = 2 * np.pi * np.random.rand()
    return r*np.cos(phi), r*np.sin(phi)

xplot = []
yplot = []
N = 1000
for i in range(N):
    x, y = sample_xy_naive()
    xplot.append(x)
    yplot.append(y)

plt.plot(xplot,yplot,'o',color='r')
plt.show()
```



The points clump more in the center!

Sampling points inside a circle

- The points **clump more in the center** because r is **not uniformly distributed**.
- Recall the **differential area element** in polar coordinates: $dxdy = rdrd\phi$
- This leads to the probability density functions: $\rho_r(r) = 2r, \qquad \rho_\phi(\phi) = \frac{1}{2\pi}.$
- **Cumulative Distribution Function (CDF)**

$$F_r(r) = \int_0^r \rho_r(r')dr' = r^2.$$

- To obtain a properly distributed r, we solve $F_r(r) = \eta$ which gives: $\boxed{r = \sqrt{\eta}}$

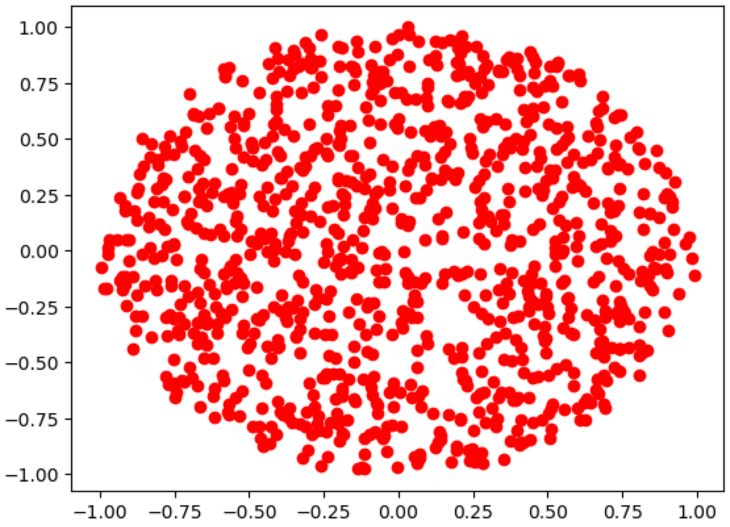
Sampling points inside a circle

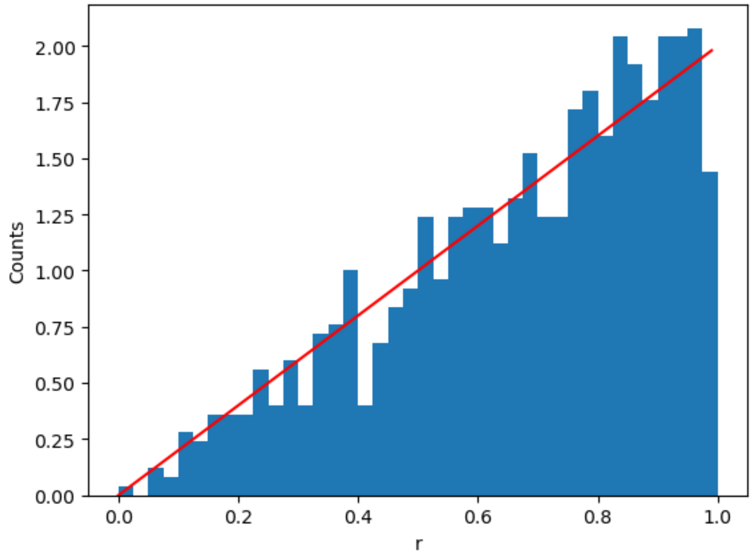
- The points **clump more in the center** because r is **not uniformly distributed**.
- Recall the **differential area element** in polar coordinates: $dxdy = rdrd\phi$
- This leads to the probability density functions: $\rho_r(r) = 2r, \qquad \rho_\phi(\phi) = \frac{1}{2\pi}.$
- **Cumulative Distribution Function (CDF)**

$$F_r(r) = \int_0^r \rho_r(r')dr' = r^2.$$

- To obtain a properly distributed r, we solve $F_r(r) = \eta$ which gives: $\boxed{r = \sqrt{\eta}}$

```python
def sample_xy_correct():
    eta = np.random.rand()
    r = np.sqrt(eta)
    phi = 2 * np.pi * np.random.rand()
    return r*np.cos(phi), r*np.sin(phi)
```





Sampling of an Isotropic Direction in 3D

One common problem in **Monte Carlo simulations** is the **random sampling of an isotropic direction in 3D space**. This issue arises in various contexts, such as:

- Sampling a random orientation of an **axially symmetric object** (e.g., a rod).
- Sampling the **momentum direction** of a particle.

This problem is equivalent to **choosing a random point on a unit sphere**. The coordinates x, y, z on the sphere can be parametrized using **azimuthal** and **polar angles**:

- $\phi \in [0, 2\pi)$ (azimuthal angle)
- $\theta \in [0, \pi)$ (polar angle)

Using these angles, the Cartesian coordinates are given by:

$$x = \sin\theta\cos\phi$$
$$y = \sin\theta\sin\phi$$
$$z = \cos\theta$$

Sampling of an Isotropic Direction in 3D

$$x = \sin\theta\cos\phi$$
$$y = \sin\theta\sin\phi$$
$$z = \cos\theta$$

- The solid angle element is:

$$d\Omega = \sin(\theta)d\theta d\phi,$$

- The random variables ϕ and θ are independent.
- ϕ is uniformly distributed in $[0, 2\pi]$, making its sampling straightforward.
- However, θ has a **weighted probability density function**: $\rho_\theta(\theta) = \frac{1}{2}\sin(\theta),$

The cumulative distribution function is:

$$F_\theta(\theta) = \int_0^\theta \frac{1}{2}\sin(\theta')d\theta' = \frac{1-\cos(\theta)}{2}$$

Solving $F_\theta(\theta) = \eta$, we obtain:

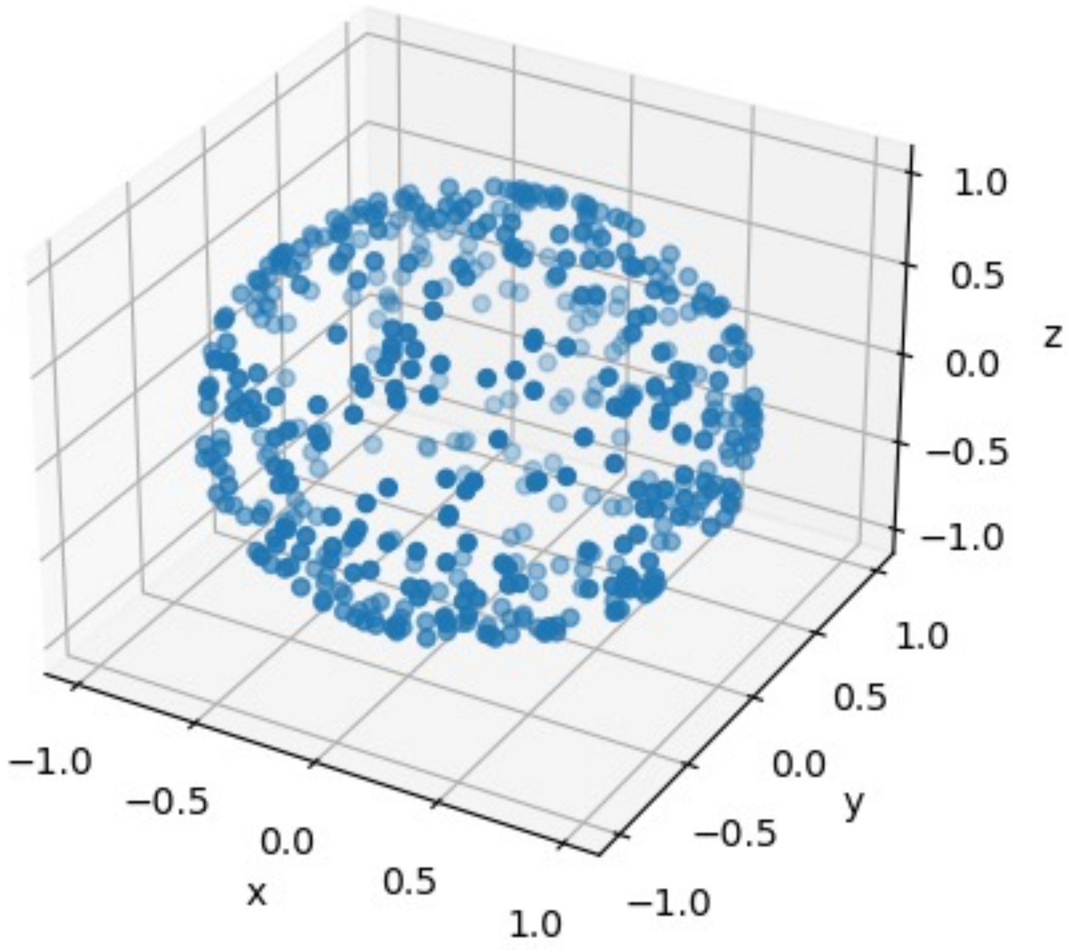
$$\boxed{\theta = \arccos(2\eta - 1).}$$

In practice, work directly with $\cos\theta$ and $\sin\theta$:

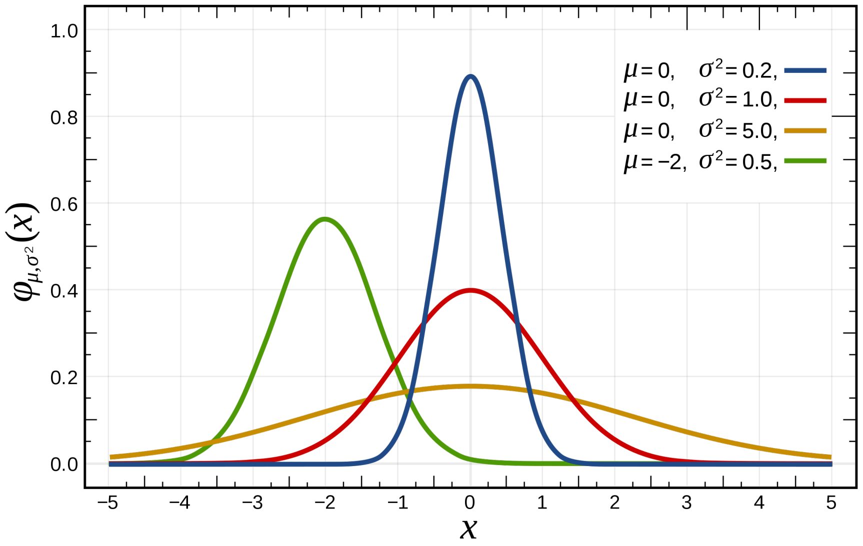
$$\cos(\theta) = 2\eta - 1, \qquad \sin(\theta) = \sqrt{1 - [\cos(\theta)]^2}$$

Sampling an isotropic direction

```
def sample_xyz_isotropic():
    phi = 2 * np.pi * np.random.rand()
    costh = 2 * np.random.rand() - 1
    sinth = np.sqrt(1-costh*costh)
    return sinth * np.cos(phi), sinth * np.sin(phi), costh

xplot = []
yplot = []
zplot = []
N = 500
for i in range(N):
    x, y, z = sample_xyz_isotropic()
    xplot.append(x)
    yplot.append(y)
    zplot.append(z)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')

ax.scatter(xplot,yplot,zplot)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.show()
```



Sampling normally distributed variables



One of the most common probability distributions is the **normal (or Gaussian) distribution**, given by:

$$\rho(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} .$$

There are many standard methods for sampling from this distribution. One common approach is to **standardize the variable** by making the transformation $x \rightarrow \mu + \sigma x$

After this transformation, the new variable follows a **standard normal distribution** with **zero mean** and **unit standard deviation**:

$$\rho(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} .$$

Calculating the cumulative distribution function $F(x) = \int_{-\infty}^{x} \rho(x')dx'$ is **not straightforward**, use numerical methods.

Sampling normally distributed variables

Instead of one variable, we can consider a pair of independent normally distributed variables x, y:

$$\rho(x, y) = \frac{1}{2\pi} e^{-\frac{x^2}{2}} e^{-\frac{y^2}{2}},$$

Making a change of variables to polar coordinates

$$x = r\cos(\phi), \qquad y = r\sin(\phi),$$

and taking into account

$$dxdy = rdrd\phi$$

we get

$$\rho(r, \phi) = \frac{1}{2\pi} r e^{-r^2/2}.$$

Therefore, we can sample x and y by sampling two independent random variables r and ϕ. ϕ is uniformly distributed in $[0, 2\pi)$. For r we have the following probability density

$$\rho_r(r) = re^{-r^2/2},$$

and the cumulative distribution function

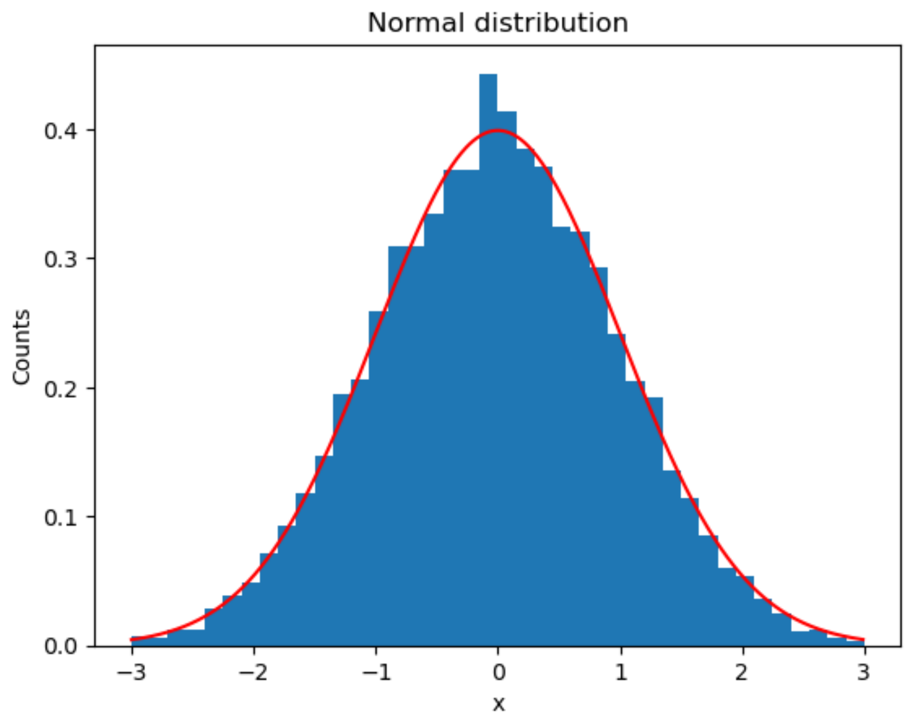
$$F_r(r) = \int_0^r r' e^{-r'^2/2} dr' = 1 - e^{-r^2/2},$$

therefore

$$r = \sqrt{-2\ln(1-\eta)}.$$

Sampling normally distributed variables

```python
def sample_xy_normal():
    phi = 2 * np.pi * np.random.rand()
    eta = np.random.rand()
    r = np.sqrt(-2*np.log(1-eta))
    return r * np.cos(phi), r * np.sin(phi)

N = 10000
samples = []
for i in np.arange(0,N,2):
    x, y = sample_xy_normal()
    samples.append(x)
    samples.append(y)
```



Rejection sampling

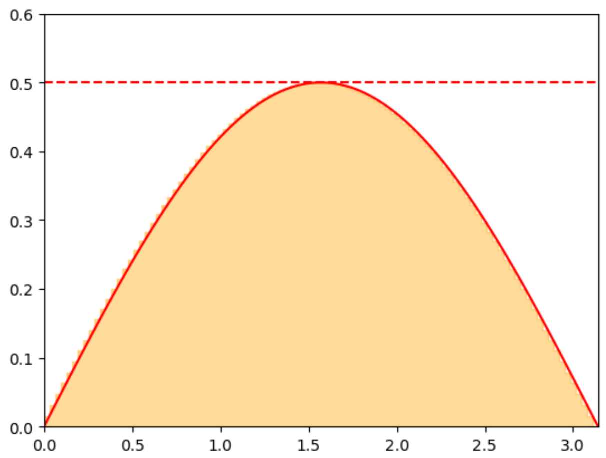
The **rejection sampling method** allows one to sample a variable ξ from an **envelope distribution** and accept or reject it with a certain probability.

Consider again the probability density function for the polar angle:

$$\rho_\theta(\theta) = \frac{\sin(\theta)}{2}.$$

Since ρ_θ is bounded from above, we define: $\rho_\theta^{\max} = 1/2.$

1. Sample a candidate value θ_{cand} from a uniform distribution over $(0, \pi)$.
2. Accept θ_{cand} with probability: $p = \rho_\theta(\theta_{\text{cand}})/\rho_\theta^{\max}.$
3. This step can be performed by sampling y **from a uniform distribution** over $(0, \rho_\theta^{max})$ and accepting θ_{cand} if $y < \rho_\theta(\theta_{cand})$



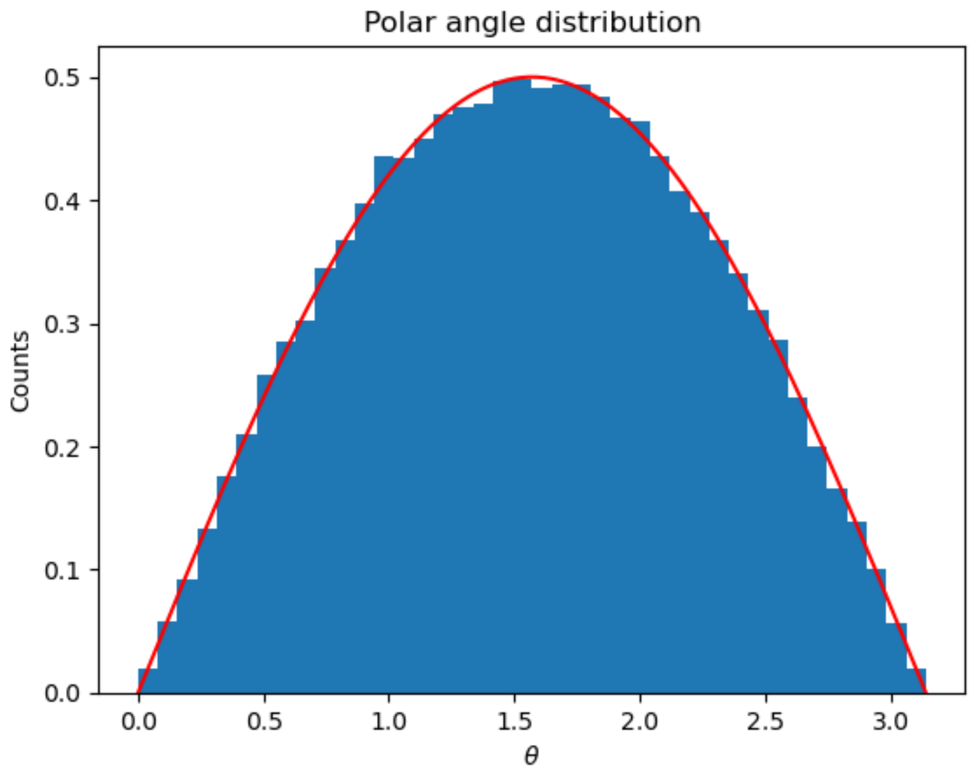
Geometric Interpretation: If we consider $\theta_{cand} = x$ and y as the **coordinates of a point** in a plane, we accept θ_{cand} **if it lies below the curve** defined by $\rho_\theta(\theta)$. This ensures that θ_{cand} values are accepted at a rate proportional to $\rho_\theta(\theta)$, as desired.

Advantages of Rejection Sampling:

$\rho_\theta(\theta)$ **does not need to be a normalized distribution** for the method to work.

Rejection sampling

```
def sample_rejection(rho, a, b, rhomax):
    while True:
        x_cand = a + (b-a)*np.random.rand()
        y = rhomax * np.random.rand()
        if (y < rho(x_cand)):
            return x_cand
    return 0.

def rho_theta(theta):
    return np.sin(theta) / 2.

N = 100000
samples = []
for i in np.arange(0,N,1):
    theta = sample_rejection(rho_theta, 0., np.pi, 0.5)
    samples.append(theta)
```



Pros and Cons of Rejection Sampling

Pros:

- Does not require the distribution to be normalized.
- Works even if y_{max} is **larger than the true maximum** of $\rho(x)$
- Applicable to **generic distributions** and does not require the evaluation of the cumulative distribution function.

Cons:

- Can be **inefficient** if the rejection rate is high (e.g., for highly peaked distributions).
- Not directly applicable to distributions **over infinite ranges**.

Generalizations of Rejection Sampling

To address some of its limitations, several generalizations of rejection sampling can be used, including:

- **Adaptive rejection sampling** by considering multiple **enveloping rectangles**.
- **Variable transformation** to map an **infinite interval** into a finite one.
- **Sampling from a non-uniform enveloping distribution** for better efficiency.

Importance sampling

Recall the calculation of an integral as statistical average

$$I = \int_a^b f(x)dx = (b-a)\langle f \rangle, \qquad \text{where} \qquad \langle f \rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i), \qquad x_i \in U(a,b)$$

Some issues with the method:

- Sample unimportant regions (e.g. f is highly peaked)
- Integrable singularities

Importance sampling:

Sample x_i from a *non-uniform* distribution w(x) that resembles f(x).

The integrand is then calculated as

$$I = \int_a^b \frac{f(x)}{w(x)} w(x)dx = \left\langle \frac{f(x)}{w(x)} \right\rangle_w$$

Normalization:

$$\int_a^b w(x)dx = 1.$$

Error:

$$\delta I = \frac{\sqrt{\left\langle \left[\frac{f(x)}{w(x)}\right]^2 \right\rangle_w - \left\langle \frac{f(x)}{w(x)} \right\rangle_w^2}}{\sqrt{N}}.$$

Importance sampling

$$I = \int_a^b \frac{f(x)}{w(x)} w(x) dx = \left\langle \frac{f(x)}{w(x)} \right\rangle_w \qquad \delta I = \frac{\sqrt{\left\langle \left[\frac{f(x)}{w(x)}\right]^2 \right\rangle_w - \left\langle \frac{f(x)}{w(x)} \right\rangle_w^2}}{\sqrt{N}}.$$

- For w(x)=1/(b-a) we recover the mean value method

$$I = \int_a^b f(x) dx = (b-a)\langle f \rangle$$

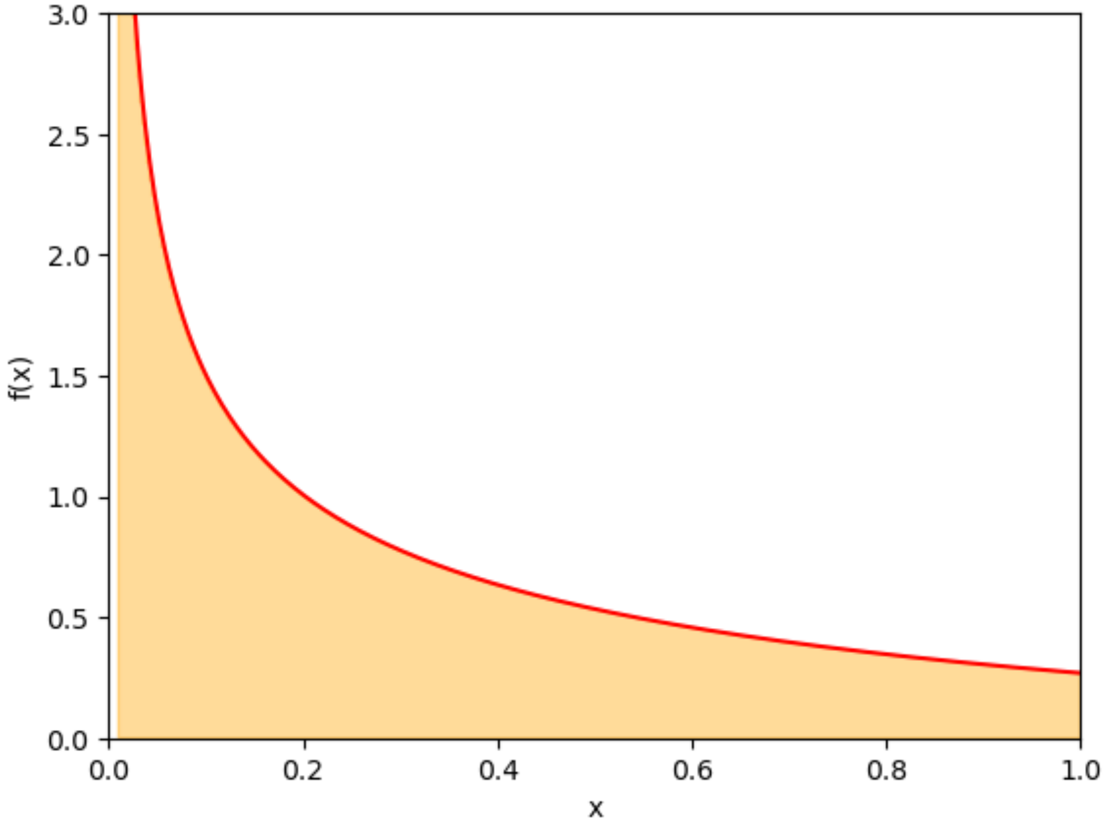
- For w(x) ∝ f(x) one has $\left\langle \frac{f(x)}{w(x)} \right\rangle = \text{const} = \text{I}$ and $\delta I = 0$

Importance sampling

```python
# Calculate integral \int_a^b f(x) dx using importance sampling
# f = f(x) is the integrand
# N is the number of random samples
# wx = w(x) is the normalized probability density from which
# the sampling takes place
# sampler is a function which samples a random number from w(x)
def intMC_weighted(f, N, wx, sampler):
    total = 0
    total_sq = 0
    for i in range(N):
        x = sampler()
        fval = f(x)
        total += fval / wx(x)
        total_sq += (fval / wx(x))**2
    fw_av = total / N
    fwsq_av = total_sq / N
    return fw_av, np.sqrt((fwsq_av - fw_av*fw_av)/N)
```

Importance sampling: Example

$$\int_0^1 \frac{x^{-1/2}}{e^x + 1} dx$$



Integrable singularity at x=0

Importance sampling: Example

$$\int_0^1 \frac{x^{-1/2}}{e^x + 1} dx$$

Mean value method

$$w(x) = \frac{1}{b-a}$$

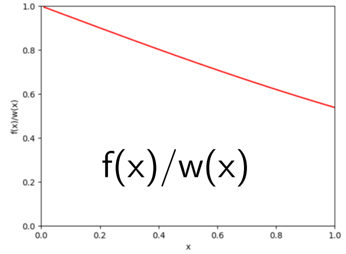
```
def uniform_sample():
    eta = np.random.rand()
    return eta

def uniform_w(x):
    return 1.

np.random.seed(1)
N = 1000000
I, err = intMC_weighted(f, N, uniform_w, uniform_sample)
print("I = ",I," +- ",err)
```

```
I =  0.8374063441946126  +-  0.0017772180714415427
```

Importance sampling

$$w(x) = \frac{1}{2\sqrt{x}} \qquad I = \left\langle \frac{2}{e^x + 1} \right\rangle_w \qquad x = \eta^2$$

```
def rsqrt_sample():
    eta = np.random.rand()
    return eta * eta

def rsqrt_w(x):
    return 1. / (2. * np.sqrt(x))
```




```
N = 1000000
I, err = intMC_weighted(f, N, rsqrt_w, rsqrt_sample)
print("I = ",I," +- ",err)
```

```
I =  0.839014917136739  +-  0.0001409071521618816
```

Statistical error is more than x10 smaller than in the mean value method.
We would need more than x100 samples in the mean value method to reach the same accuracy as importance sampling in this case.