# Importance-Sampling-and-Metropolis

April 9, 2024

## 1 Important Sampling

Suppose that we have to evaluate the integral

$$I = \int_0^1 f(x)dx$$

for some $f$. An alternate method for evaluating the integral would to calculate

$$I \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

Here, the average of $f$ is evaluated by considering its values at $N$ values $\{x_i\}$, chosen at random with equal probability anywhere within the interval $[0, 1]$. The error may be estimated as,

$$\frac{1}{N}\sigma_f^2 = \frac{1}{N}\left[\frac{1}{N}\sum_{i=1}^{N} f^2(x_i) - \left(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right)^2\right]$$

which vanishes for constant $f$.

Let us consider a normalized weight function $w(x)$ given by,

$$\int_0^1 w(x)dx = 1$$

Then the integral can be written as

$$I = \int_0^1 dx\ w(x)\frac{f(x)}{w(x)}$$

Let us make a change of variable

$$y(x) = \int_0^x dx' w(x')$$

such that,

$$\frac{dy}{dx} = w(x)$$

Then the integral becomes,

$$I = \int_0^1 dy \frac{f(x(y))}{w(x(y))}$$

## 1.1 Example

Find numerically

$$4\int_0^1 \frac{dx}{1+x^2} = \pi$$

[19]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[21]:
```
def f(x):
    return 1/(1+x**2)

Ns = [10, 20, 50, 100, 200, 500, 1000, 2000, 50000]

for N in Ns:
    store = 0.0
    for i in range(N):
        xi =  np.random.rand()
        store += f(xi)
    res = 4*store/N
    print(N, "\t", res, "\t", abs(res-np.pi))
```

```
10      3.216340647679113       0.07474799408932009
20      3.1843031940402957      0.0427105404505026
50      3.184601014359154       0.04300836076936099
100     3.178529074146181       0.036936420556387795
200     3.119225068790803       0.022367584798990148
500     3.1387177546184386      0.0028748989713545114
1000    3.1249157302564594      0.016676923333333704
2000    3.135881910892079       0.0057107426977141
50000   3.1432774846909313      0.0016848311011381512
```

### 1.1.1 Find the same integral using the weight function

$$w(x) = \frac{1}{3}(4-2x)$$

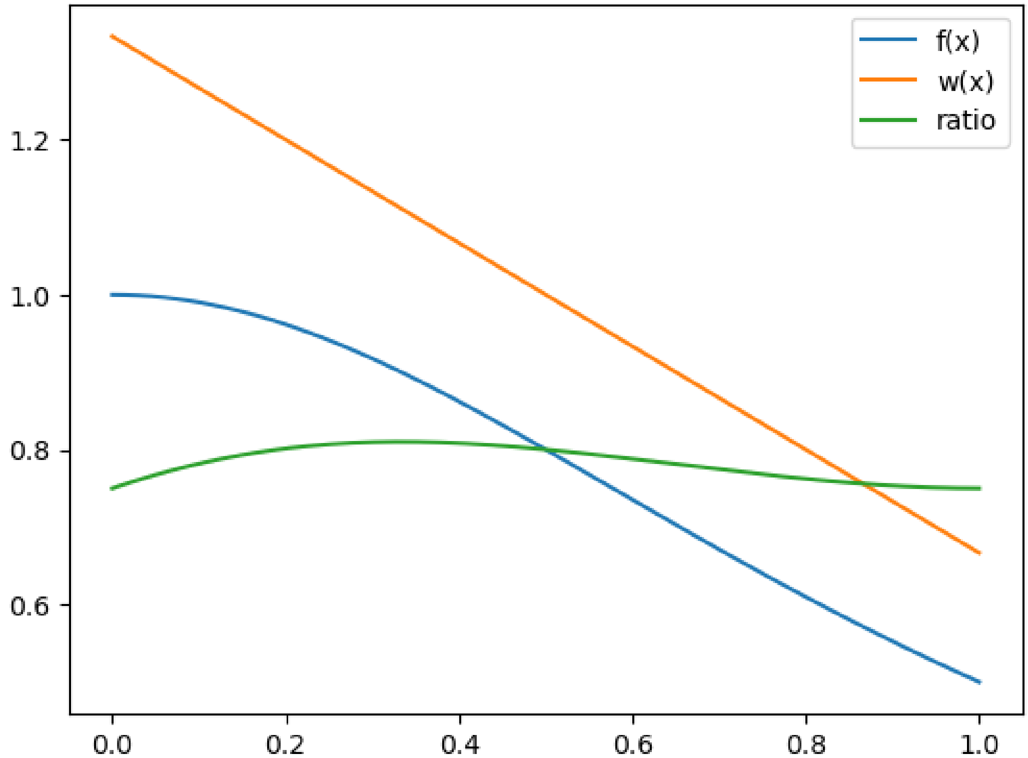
[24]:
```
def f(x):
    return 1/(1+x**2)

def w(x):
    return (4-2*x)/3

def x(y):
    return 2 - (4 - 3*y)**0.5

Ns = [10, 20, 50, 100, 200, 500, 1000, 2000, 50000]

for N in Ns:
```

```
    store = 0.0
    for i in range(N):
        yi = np.random.rand()
        xi = x(yi)
        store += f(xi)/w(xi)
    res = 4*store/N
    print(N, "\t", res, "\t", abs(res-np.pi))
```

```
10       3.1334027124404584      0.008189941149334469
20       3.1279395172271194      0.013653136362673735
50       3.1436390726699686      0.0020464190801754434
100      3.134290037337607       0.007302616252186311
200      3.137761950541716       0.0038307030480773108
500      3.1381019013514893      0.0034907522383038625
1000     3.139808443982005       0.0017842096077882452
2000     3.1402017811949174      0.0013908723948756752
50000    3.142042963230884       0.00045030964109082916
```

[5]:
```
xs = np.linspace(0.0, 1.0, num=200)
y1 = [f(x) for x in xs]
y2 = [w(x) for x in xs]
y3 = [f(x)/w(x) for x in xs]
plt.plot(xs, y1, label="f(x)")
plt.plot(xs, y2, label="w(x)")
plt.plot(xs, y3, label="ratio")
plt.legend()
```

[5]: <matplotlib.legend.Legend at 0x7f1abc253080>

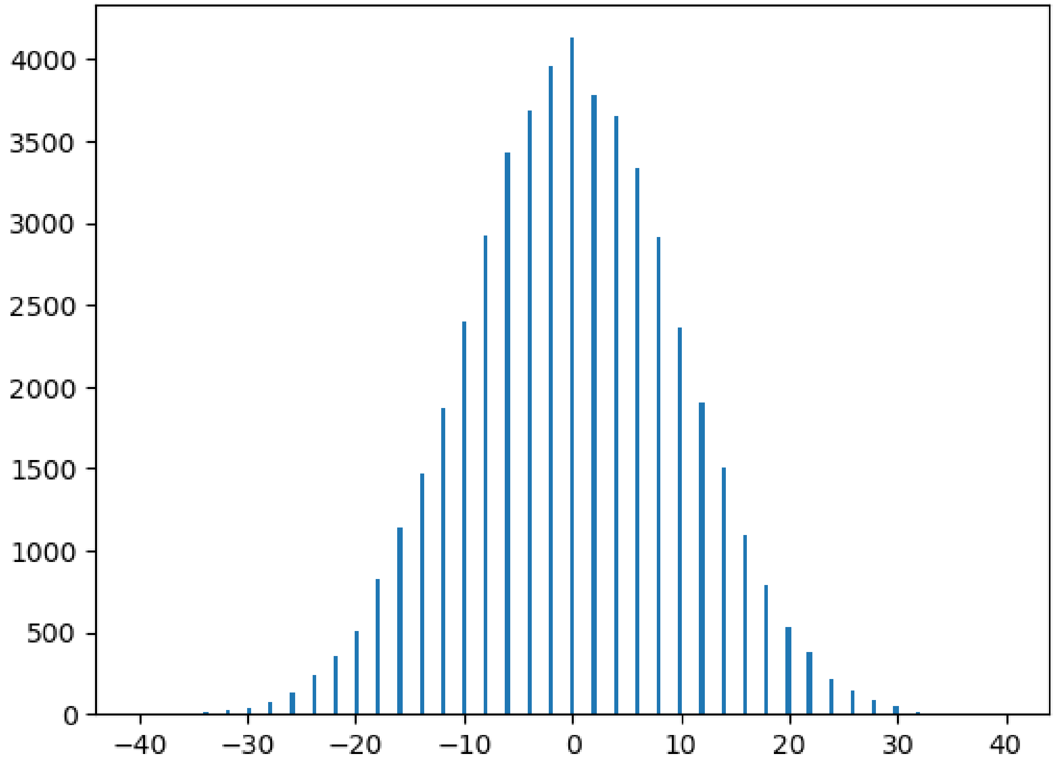



### 1.2 Random walk

Position of a random walker after $N$ steps form a normal distribution.

[6]:
```
def take_a_step():
    if np.random.rand()<0.5:
        return -1
    return 1

steps = 100
walkers = 100000
finpos = np.zeros(walkers,)
#finpos = np.zeros((walkers,),dtype=int)
k = 0
for w in range(walkers):
    pos = 0
    for s in range(steps):
        pos += take_a_step()
    finpos[k] = pos
    k += 1
```

[7]:
```
plt.hist(finpos[0::2],bins=2*steps+1);
```




What if we need a ***random walk*** with a desired distribution? Markov chain.

## 2 Metropolis-Hastings

Generate a set of random numbers with a desired distribution $w(x)$. Consider a point in the sequence $x_i$ and a new ***trial*** point $x_{i+1}$. The trial point can be generated by taking a ***random*** step from $x$. The trial point is ***accepted*** or ***rejected*** depending on the ratio

$$r = \frac{w(x_{i+1})}{w(x_i)}.$$

If $r$ is 1 or larger, then the step is accepted, else it is ***accepted with a probability*** $r$.

So the acceptance probability is

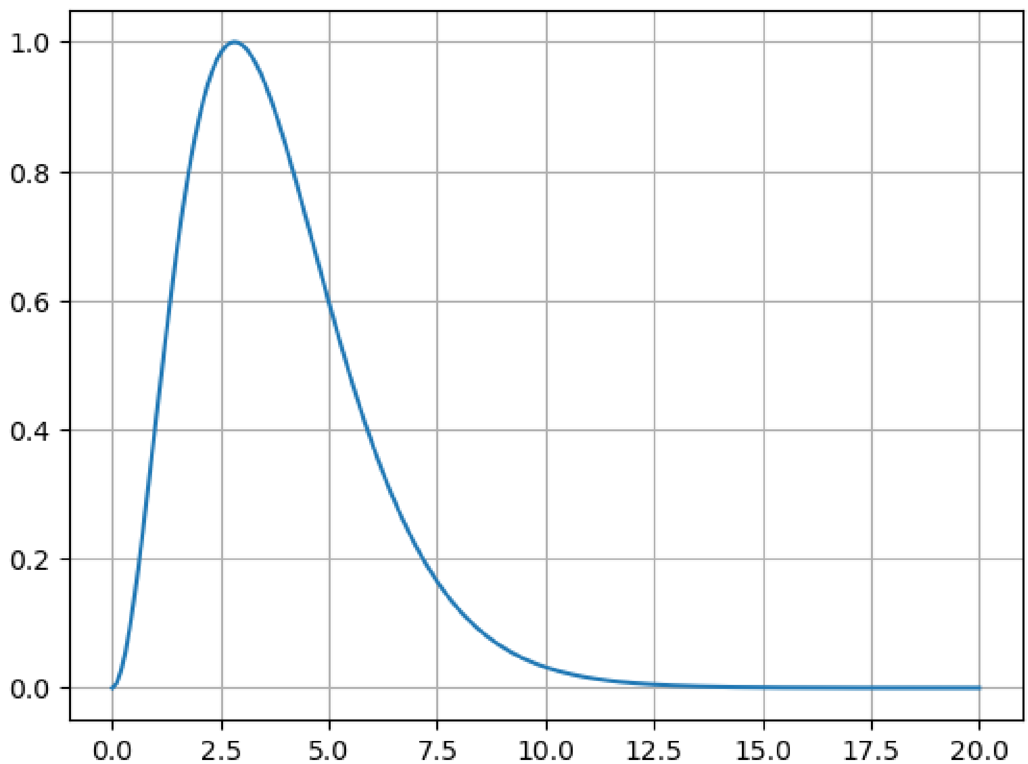
$$a = \min\left[1, \frac{w(x_{i+1})}{w(x_i)}\right]$$

#### 2.0.1 Example 1

You have access to a uniform sampler. Generate a set of random numbers having a distribution

$$\frac{x^3}{e^x - 1},\ \forall\ x \geq 0$$

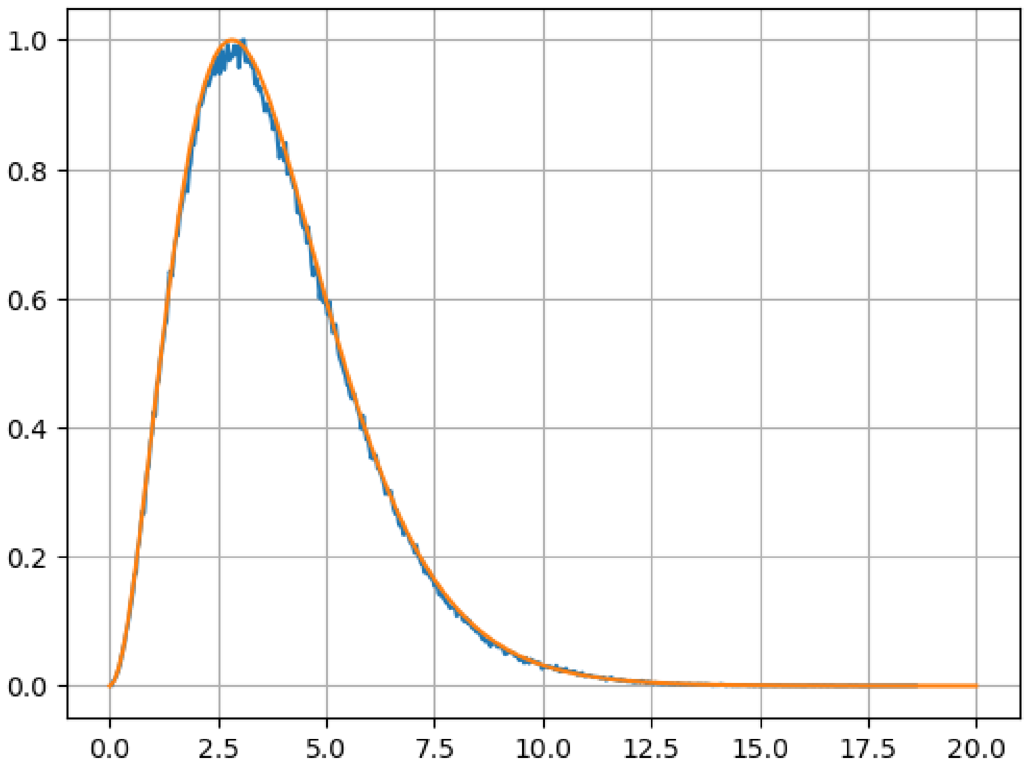
[25]:
```
def target_dist(x):
    if x<=0.0 :
        return 0.0
    return x**3/(np.exp(x)-1)

xs = np.linspace(0.,20.,num=200)
ys = [target_dist(x) for x in xs]

ys = ys/np.max(ys)

plt.plot(xs, ys);
plt.grid()
```




[26]:
```
x = 2.5
N = 1000000
sequence = np.zeros(N)
for i in range(N):
    xt = x + 2.0*(np.random.rand()-0.5)
    a = min([1.0, target_dist(xt)/target_dist(x)])
    if np.random.rand() < a:
        x = xt
    sequence[i] = x
```

[27]:
```
freq, edges = np.histogram(sequence,bins=500);
freq = freq/np.max(freq);
strip = edges[1]-edges[0];
```

[29]:
```
plt.plot(edges[0:-1]+strip/2, freq)
plt.plot(xs, ys)
plt.grid()
```



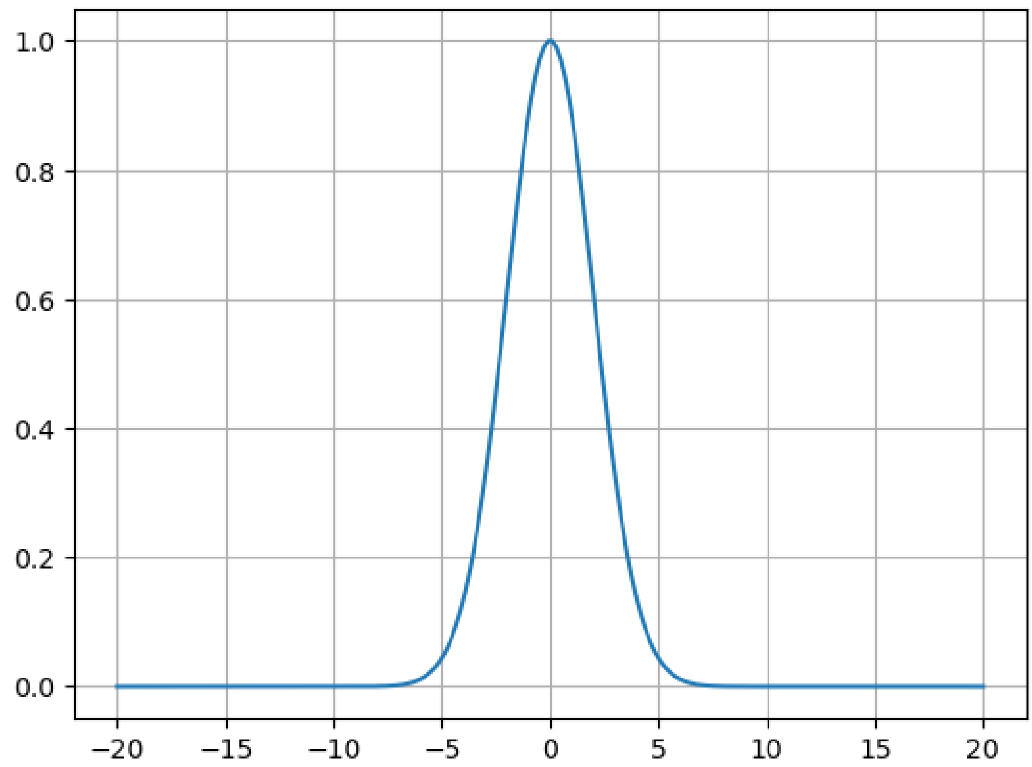

# 3 Example 2

Generate a sequence of random numbers that has a normal distribution.

$$e^{-(x-x_\circ)^2/2\sigma^2} \qquad \forall x \in \mathbb{R}$$

[12]:
```
def target_dist(x):
    return np.exp(-(x-xo)**2/2/sigma)

xs = np.linspace(-20.0,20.0,num=200)

xo = 0.0
sigma = 4.0
```

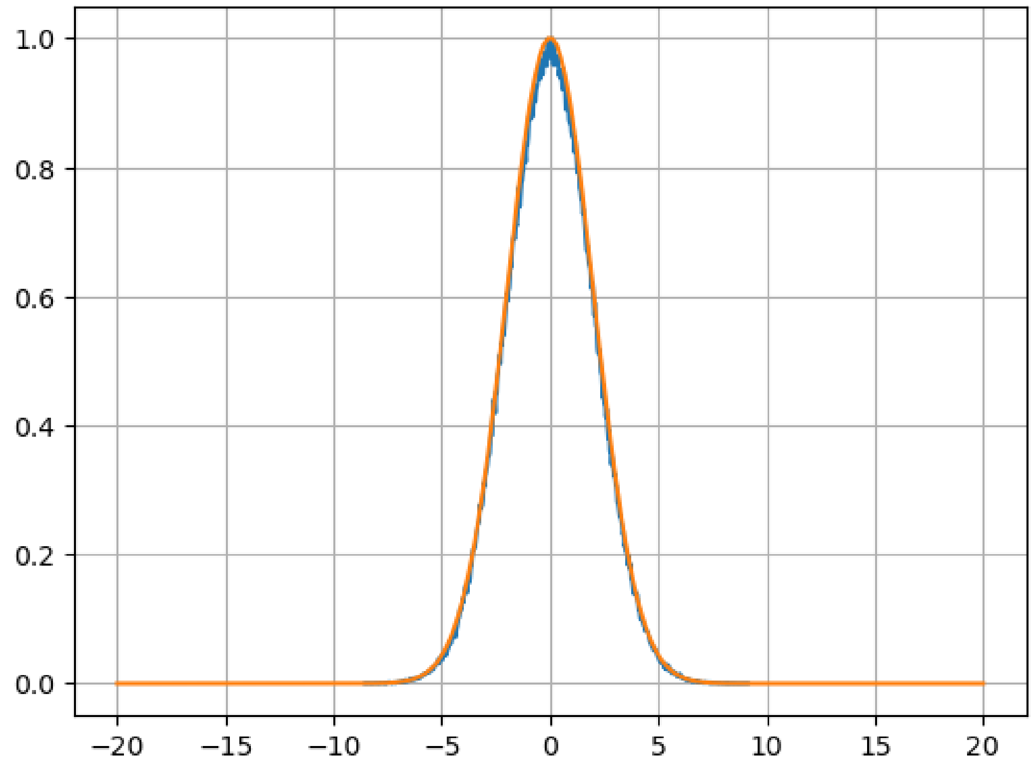
```
ys = [target_dist(x) for x in xs]

ys = ys/np.max(ys)

plt.plot(xs, ys);
plt.grid()
```




[16]:
```
x = 0.5
N = 1000000
sequence = np.zeros(N)
for i in range(N):
    xt = x + 2.0*(np.random.rand()-0.5)
    r = min([1.0, target_dist(xt)/target_dist(x)])
    if np.random.rand() < r:
        x = xt
    sequence[i] = x
```

[17]:
```
freq, edges = np.histogram(sequence,bins=500);
freq = freq/np.max(freq);
strip = edges[1]-edges[0];
```

[18]:
```
plt.plot(edges[0:-1]+strip/2, freq)
plt.plot(xs, ys)
plt.grid()
```




[ ]:
```
```