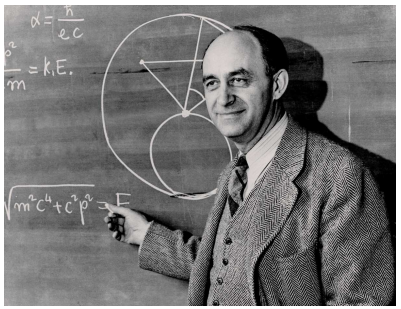
# Importance Sampling

Alessandro Torcini

LPTM - Université de Cergy-Pontoise

UNIVERSITÉ
de Cergy-Pontoise

LPTM
Laboratoire de Physique
Théorique et Modélisation



# Non-Uniform Random Numbers

Suppose I have a generator of random numbers which gives numbers $x$ distributed in $[0,1]$ with a probability distribution function (PDF) $q(x)$, and I want to obtain with a transformation of variable $y = y(x)$ random numbers $y$ which are distributed accordingly to another PDF $p(y)$ defined in a domain $[a,b]$.

The probability is conserved

Therefore

1. $\int_0^1 q(x)dx = \int_a^b p(y)dy = 1$ —- $q(x)$ and $p(y)$ are PDFs
2. The probability to have a random number in $[x, x+dx]$ is given by $q(x)dx$
3. This is conserved if I make a transformation of variables from $x \to y$
4. Therefore $\boxed{q(x)dx = p(y)dy}$ with $y = y(x)$

# Non-Uniform Random Numbers

To simplify our derivation let us suppose that $x$ are random number distributed uniformly in $[0,1]$ then $q(x) \equiv 1$. Thus we have

$$dx = p(y)dy \rightarrow x = \int_a^y p(y')dy' = P(y)$$

where $P(y)$ is the cumulative probability function. We can invert this relationship and get

$$\boxed{y = P^{-1}(x)}$$

which gives the desired random number $y$ with the PDF $p(y)$.

A simple example

I want random numbers $y$ distributed as $p(y) = y/2$ in the interval $[0,2]$, therefore

$$x = P(y) = \int_0^y \frac{y'}{2} dy' = \frac{y^2}{4}$$

and

$$y = P^{-1}(x) = \sqrt{4x}$$

and since $x \in [0,1]$ then $y \in [0,2]$ as desired.

# Exponential Distribution

Let us consider an important distribution the exponential one

$$p(y) = a\mathrm{e}^{-ay} \quad \text{with} \quad y \in [0; \infty)$$

the cumulative distribution is

$$P(y) = \int_0^y p(y')dy' = 1 - \mathrm{e}^{-ay}$$

Therefore

$$x = 1 - \mathrm{e}^{-ay} \rightarrow (1 - x) = \mathrm{e}^{-ay} \rightarrow y = \frac{-\ln(1-x)}{a}$$

since $1 - x$ is a random variable in $[0, 1]$ as $x$, we can finally write

$$\boxed{y = \frac{-\ln(x)}{a}}$$

# Exponential Distribution

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

def ranexp (a):
    x=np.random.random_sample()
    y= -np.log(x)/a
    return y

N=1000000
data=[]
for i in range (1,N):
    z=ranexp(0.5)  #a=0.5
    data.append(z)

plt.hist(data, bins=200, range=(0,10),normed=1)
# data contains the number of times you have the random number
# bins is the number of bins you want
# range fix the extrema
# normed tells that you want a histogram with area one
plt.show()
```

# Importance Sampling

Instead of employing an uniform sampling of random points it would be clever to choose the random points accordingly to a probability distribution function (PDF) $p(x)$ that favours the converge of the integral

$$I = \int_a^b \mathrm{d}x\, f(x)$$

In particular we can rewrite the integral as

$$I = \int_a^b \mathrm{d}x\, \frac{f(x)}{p(x)} p(x) = \langle \frac{f(x)}{p(x)} \rangle_p \quad \text{where} \quad \int_a^b \mathrm{d}x\, p(x) = 1$$

therefore $p(x)$ is a PDF defined in $[a, b]$.

# Importance Sampling

An estimate of $I$ can be obtained by generating $N$ random points $x_i$ which follows a distribution $p(x_i)$, then

$$I_{estimate} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

obvously $I_{estimate} \to I$ for $N \to \infty$.
Which is the variance of the estimate of $I$ ?

$$\sigma_I^2 = \left\langle \left( \frac{f(x)}{p(x)} \right)^2 \right\rangle_p - \left\langle \frac{f(x)}{p(x)} \right\rangle_p^2$$

For which choice of $p(x)$ the error is minimal ?

1. If we choose $p(x) = Cf(x)$ clearly $\sigma_I^2$ is zero ! ! !
2. Is this magic or should we pay a price for this optimal choice ?
3. $p(x)$ is a PDF, therefore $\int_a^b p(x)dx = C \int_a^b f(x)dx = CI = 1$

# Importance Sampling

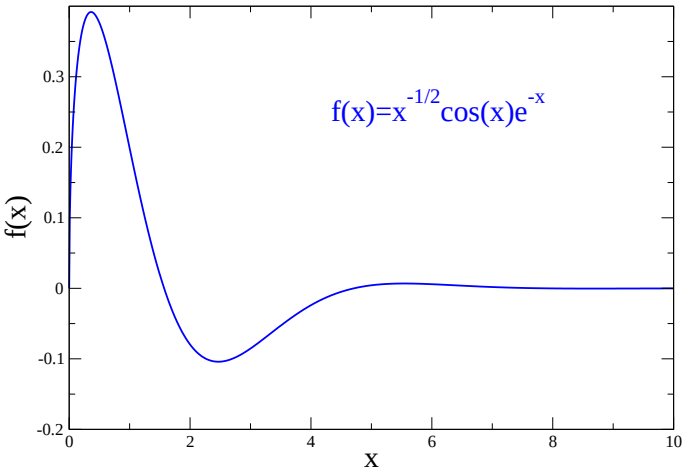
No lunch for free

To define $p(x) = Cf(x)$ we should know the normalization constant $C = 1/I$, therefore we should know the integral $I$ that we want to calculate ! ! !

How to proceed

1. We should choose $p(x)$ "close" to $f(x)$, such that the points where $f(x)$ is large are more frequently selected
2. Ensure that we never have $p(x) << f(x)$ or we risk to have a very large error (variance $\sigma_I^2$
3. The method is very effective : individual steps are very simple (random number $x_i$ according to the law $p(x)$ to calculate $f(x_i)/p(x_i)$)
4. The error is $1/\sqrt{N}$ in any spatial dimension
5. Problems :
   (a) It is not always easy to select a "good" $p(x)$
   (b) We need to construct a specific random number generator for each function

# An example




**Exercise**

Estimate the integral

$$I = \int_0^\infty \sqrt{x}\cos(x)\mathrm{e}^{-x}dx$$

by employing the importance sampling method with PDF $p(x) = \mathrm{e}^{-x}$

First all the integral can be rewritten as

$$I = \int_0^\infty G(x)p(x)dx \qquad G(x) = \sqrt{x}\cos(x)$$

which can be estimated by a MC method with a sequence of random numbers $\{y_i\}$ obtained from the PDF $p(y)$ as

$$I_{estimate} = \frac{1}{N}\sum_{i=1}^{N} G(y_i)$$

where $y_i = -\ln(x_i)$ with $x_i$ random variable uniform distributed in $[0, 1]$.

# An example

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

def ranexp (a):
    x=np.random.random_sample()
    y= -np.log(x)/a
    return y

def f1(x):          # the function
    return np.sqrt(x)*np.cos(x)
```

# An example

```
def HM(f):
  somme, somme2 = 0. , 0.
  i = 0
  while i<HM.N:
    fi=f(ranexp(1.0)) # function evaluated in random values
    somme += fi
    somme2 += fi*fi
    i += 1
  somme /= float(HM.N)                    # the integral
  somme2 /= float(HM.N)
  HM.erreur = np.sqrt((somme2-somme*somme)/float(HM.N))
  # standard deviation of the average
  return somme

# we can estimate the integral over many realizations N
for N in [1e2, 1e3, 1e4, 1e5,1e6]:
  print ("N =", N)
  HM.N = N
  print ("I1 = ", HM(f1), "+/-", HM.erreur, "(exact : 0.201656)")
```

# Markov Chains

A more general method to obtain sequences of random numbers $\{x_i\}$ distributed according to a generic PDF $p(x)$ is the so called method of Markov Chains
The random sequence can be generated as a random walk :

1. Given a value $x_i$ at time $t$ the next value $x_j \in [x_i - \lambda, x_i + \lambda]$ at time $t+1$ can be generated by a transition probability density $T_\lambda(x_i \to x_j)$
2. if $P_t(x_i)$ is the probability to get the variable in $x_i$ at time $t$ then

$$P_{t+1}(x_j) = \int P_t(x_i) T_\lambda(x_i \to x_j) dx_i$$

3. since $T_\lambda$ is a PDF we have $\boxed{\int T_\lambda(x_i \to x_j) dx_j = 1}$ the point $x_i$ should end in a point within the interval $[x_i - \lambda, x_i + \lambda]$
4. therefore we can formally write

$$P_{t+1}(x_j) - P_t(x_j) = \int P_t(x_i) T_\lambda(x_i \to x_j) dx_i - \int P_t(x_j) T_\lambda(x_j \to x_i) dx_i$$

# Markov Chains

The transition PDF $T_\lambda$ should be selected in a way that the chosen $p(x)$ is stationary solution (independent of $t$) of the recurrence equation for the PDF $P_t(x)$, namely

$$p(x) \equiv P^*(x) = \int P^*(y) T_\lambda(y \to x) dy$$

by employing the fact that $T_\lambda$ is normalized to one we can write the following

$$\int P^*(x) T_\lambda(x \to y) dy = \int P^*(y) T_\lambda(y \to x) dy$$

or equivalently

$$\int [P^*(x) T_\lambda(x \to y) - P^*(y) T_\lambda(y \to x)] dy = 0$$

A necessary (but not sufficient) condition ensuring that the above equation is satisfied is the so-called detailed balance condition

$$P^*(x) T_\lambda(x \to y) = P^*(y) T_\lambda(y \to x)$$

# Markov Chains

What happens when we start with a distribution $P_t(x) \neq P^*(x)$ and we apply the recurrence iteration

$$P_{t+1}(x_j) = \int P_t(x_i) T_\lambda(x_i \to x_j) dx_i$$

the PDF $P_t(x) \to P^*(x)$ for $t \to \infty$ or not ?

Yes if the Markov chain is ergodic

1. ergodic means that, for any couple $x_i, x_j$, there is a possibility to go from $x_i$ to $x_j$ with a finite number of consecutive individual jumps.
2. independently from the initial condition $x_i$ the recurrence relation can bring you in any final value $x_j$

# Metropolis' Algorithm

A simple choice to select $T_\lambda$ that fulfills the detailed balance has been proposed by Metropolis (1953)

$$T_\lambda(x_i \to x_j) = min\left[1, \frac{p(x_j)}{p(x_i)}\right]$$

The Algorithm

1. $x_i$ is the value of the variable at time $t$
2. For time $t+1$ we select a tentative value $x_{tr} = x_i + \lambda_i$, where $\lambda_i \in [-\lambda, \lambda]$ is a random number
3. we calculate the transition probability $T_\lambda(x_i \to x_{tr}) = min\left[1, \frac{p(x_{tr})}{p(x_i)}\right]$
4. the new point is selected $x_{i+1} = x_{tr}$ with probability $T_\lambda(x_i \to x_{tr})$
5. we restart from step one

The step 4 can be implemented by choosing a random number $q \in [0, 1]$ :

1. if $q < T_\lambda(x_i \to x_{tr})$ the trial point $x_{tr}$ is selected
2. otherwise we select $x_i$ another time

# Remarks on the Metropolis' Algorithm

1. We do not need to know the normalization constant $p(x) = Cf(x)$, because the algorithm is defined only in term of the ratio of the functions

$$\frac{p(x_{tr})}{p(x_i)} = \frac{f(x_{tr})}{f(x_i)}$$

   However, the algorithm will generate a distribution of random points accordingly to a normalized PDF $p(x)$. i.e. for which $\int p(x)dx = 1$.

2. the value of $\lambda$ (maximal step) should be selected by trials and errors, searching for a $\lambda$ values giving acceptance rate around $30 - 50\%$, i.e. on average every two/three trials $x_{tr}$ will be accepted

   (a) if $\lambda$ is too small the acceptance rate is high, but the random points are quite similar $x_i$ and the estimation of the integral is not well done, since the same region is always explored ;

   (b) if $\lambda$ is too large the acceptance rate is quite low, and $x_i$ does not change in time, also not good for the integral

# Remarks on the Metropolis' Algorithm

1. Thermalization Phase : a transient period is needed to the Markov chain to converge towards the asymptotic PDF $p(x)$, therefore the first $M$ steps should be discarded and not used for the estimation of the integral. $M$ will depend on the $\lambda$ value.
2. How often ? The points generated by a Markov chain are always correlated, and the estimaton of the ntegral is better performed with uncorrelated random numbers. A method to increase the decorrelation is to use one random point every $K$ generated points in the Markov chain.
3. A good starting point $x_0$ for the sequence of random numbers is around the maximum of $p(x)$ because this is the point of maximal probability that contributes more to the integral and it should be visited as long as possible by the Markov chain ;

# Integration with the Metropolis' Algorithm

I want to estimate the variance of a Gaussian distribution, namely

$$I = \int_{-\infty}^{\infty} \frac{x^2}{\sqrt{2\pi}\sigma} \mathrm{e}^{-x^2/(2\sigma^2)} dx = \sigma^2$$

since in this case the average is exactly zero.

How do I proceed with the Metropoli's algorithm ?

1. I rewrite the integral as $\int_{-\infty}^{\infty} f(x)p(x)dx$, where $f(x) = x^2$ and $p(x) = \mathrm{e}^{-x^2/(2\sigma^2)}$, I do not need to normalize the PDF ! ! !
2. I generate a Markov chain $\{x_i\}$ of lenght $N$ accordingly to the chosen $p(x)$
3. I use the $N$ random numbers $\{x_i\}$ to estimate the integral

$$I_{estimate} = \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

# Integration with the Metropolis' Algorithm

## Selecting the parameters

1. It can be demonstrated that the best choice for a Gaussian distribution is $\lambda = 3.70$, for other distribution it can be different ;
2. For the thermalization phase we choose $M = 10000$ , this seems a good choice ;
3. For this integration we choose $K = 1$ in other cases can be clever to choose $K > 1$

# Variance of a Gaussian

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

def met(y):
    rr=np.random.random_sample()
    x=y+rr*met.lam*2. - met.lam
    ratio = PDF(x)/PDF(y)
    if ratio > 1.:
        return x
    rr=np.random.random_sample()
    if rr > ratio:
        return y
    return x

def PDF(x):
    return np.exp(-x*x/2.) # sigma = 1

def f1(x):          # the function
    return x*x
```

# Variance of a Gaussian

```
def HM(f):
  somme, somme2 = 0. , 0.
  i = 0
  yi= 0.
  while i<10000:     # thermalization
    yi =  met(yi)
    i += 1

  i=0
  while i<HM.N:
    yi =  met(yi)
    fi=f(yi) # function evaluated in random values
    somme += fi
    somme2 += fi*fi
    i += 1
  somme /= float(HM.N)                    # the integral
  somme2 /= float(HM.N)
  HM.erreur = np.sqrt((somme2-somme*somme)/float(HM.N))
  # standard deviation of the average
  return somme
```

# Variance of a a Gaussian

```
# we can estimate the integral over many realizations N
met.lam=3.70
for N in [1e3,1e4,1e5,1e6]:
  print ("N =", N)
  HM.N = N
  print ("I1 = ", HM(f1), "+/-", HM.erreur, "(exact : 1.00)")
```