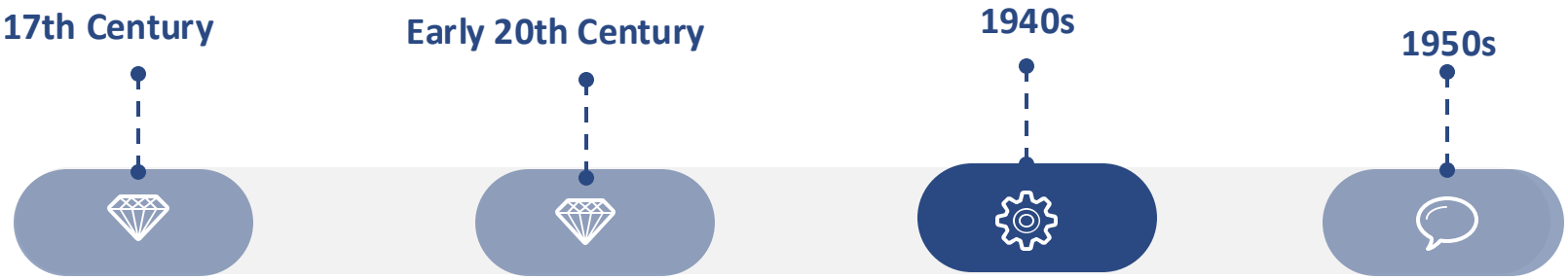
MATH 5010 –Foundations of Statistical Theory and Probability

Section- Sampling Methods: Monte Carlo

- Introduction and Applications
- Standard distributions
- Rejection sampling
- Importance sampling

Instructor: He Wang
Department of Mathematics
Northeastern University

Backgrounds



Simulation

- de Buffon (1733)

Buffon's needle problem

Markov Chain

- **Markov**
-

Monte Carlo algorithms

- The origins of the name: a famous casino town in Monaco

MCMC

- Ulam and von Neumann
- 1940s Manhattan Project: neutron diffusion techniques in a nuclear reactor
- Metropolis-Hastings algorithm
- Gibbs Sampling
- HMC

➢ Overview:

Markov Model:

Describe systems that evolve over time, based on the assumption of memorylessness.

- **Markov Chain:**

  A sequence of events where the probability of each event depends only on the previous event.

Monte Carlo Sampling:

- is used to sample from a probability distribution to obtain the desired quantity

- has limitations when the probability distribution is large.

MCMC

➢ History of Monte Carlo

Monte Carlo Simulation was named after a well-known casino town, Monte Carlo in Monaco, since the element of chance is core to the modeling approach, similar to a game of roulette

Monte Carlo Methods - How does it work?

Monte Carlo techniques are algorithms to approximate numerical result based on random numerical sampling.

- Uses random sampling and statistical modeling to estimate mathematical functions and mimic the operations to complex systems

- How it works:
  1. Uses a probability distribution for any variable that has inherent uncertainty
  2. Recalculates the results many times, using a different set of random numbers within the estimated range each time

- This process generates many probable outcomes, and becomes more accurate as the number of input grows (Law of Large Numbers)

Use of the Monte Carlo Methods-Applications

The underlying concept is to use randomness to solve problems that might be deterministic in principle, but difficult or impossible to solve. Some of the applications:

- **Simulation:** Gather information about a random object by observing many realizations of it. Example: simulation modeling (e.g. risk assessment, the spread of disease)
- **Integration Estimation- Monte Carlo Integration:** Estimate certain numerical quantities related to a simulation model. Evaluate the integral of a complicated function in a given region by writing the integral as the expectation of a random variable
- **Statistical Sampling:** Use Markov Chain Monte Carlo (MCMC) to estimate posterior distributions, Bayesian inference, model fitting, and parameter estimation.
- **Optimization:** Optimize (maximize/minimize) complicated objective functions.

Monte Carlo also plays a fundamental role in the simulation of physical systems. E.g., in nuclear physics and computer graphics, Reinforcement Learning, etc.

Application I Simulation :

Motivation problem:

Easy Question: What is the average height of the students in our class?

Method: measure heights, add them up and divide by n.

Question What is the average height f of people x in Boston $\mathcal{B}$?

Method:

$$E_{x\in\mathcal{B}}[f(x)] \coloneqq \frac{1}{N}\sum_{x\in\mathcal{B}} f(x)$$

Intractable question.

$$\approx \frac{1}{S}\sum_{s=1}^{S} f\left(x^{(s)}\right)$$

Random survey of S people in Boston.

Summary: Use sample mean to approximate the true mean.

Foundation 1-Law of Large Numbers (LLN)

Let $X_1, X_2, \ldots$ be any collection of IID R.V.'s (independent and identically distributed random variables) with common finite **mean** μ and variance σ^2. The **sample mean** of the first n variables is

$$\overline{X} = \frac{1}{n}(X_1 + \ldots + X_n)$$

The weak LNN (**Law of Large Numbers**) says that:

$\overline{X}$ converges (in probability) to the true mean μ as $n \to \infty$.

That is, for any $\epsilon > 0$,

$$\lim_{n \to \infty} P(|\overline{X} - \mu| \geq \epsilon) = 0$$

In other words, the average of the results obtained from a large number of trails should be "close" to the expected value and tends to become closer to the expected values as more trails are performed.

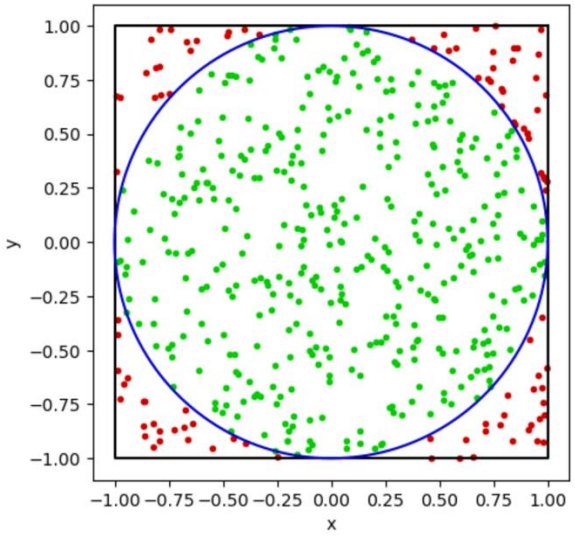
➢ Sampling Example: Uniformly Rain Drop Distribution to Estimate π

- Suppose the rain falls **uniformly** at random over some square region of space, and a circle inscribed within the square
- The probability of a uniform raindrop falling in any region within the square must be proportional to the area of that region and independent of its location
- Suppose the square has side of length $2r$, then the radius of the circle is r.
- Let p be the probability of a raindrop lies within the inscribed circle. Then:

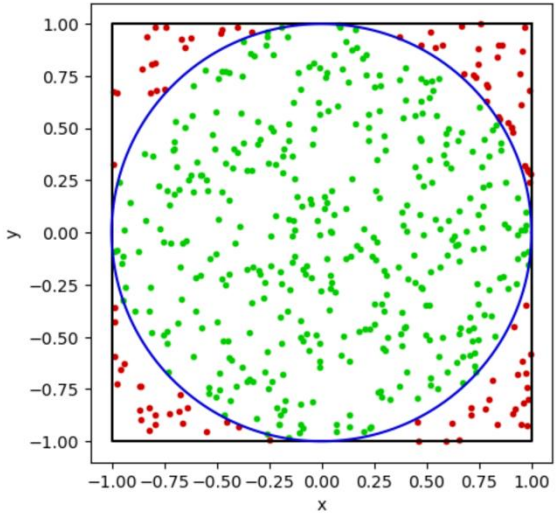
$$p = \frac{\pi r^2}{(2r)^2} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

- Can be used to estimate

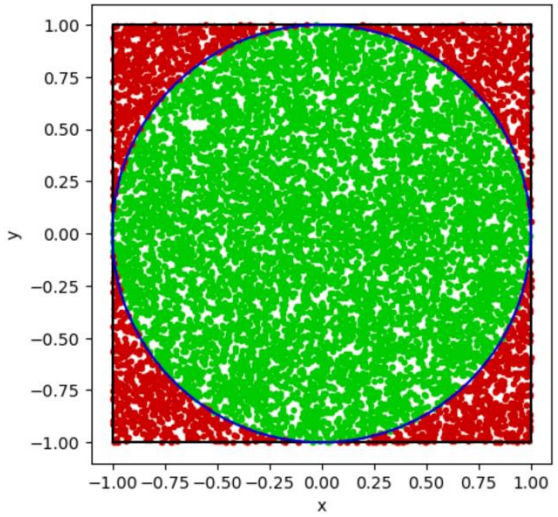
$$\pi \approx 4\hat{p} = \frac{4 * \text{number of raindrops inside the circle}}{\text{number of raindrops inside the square}}$$



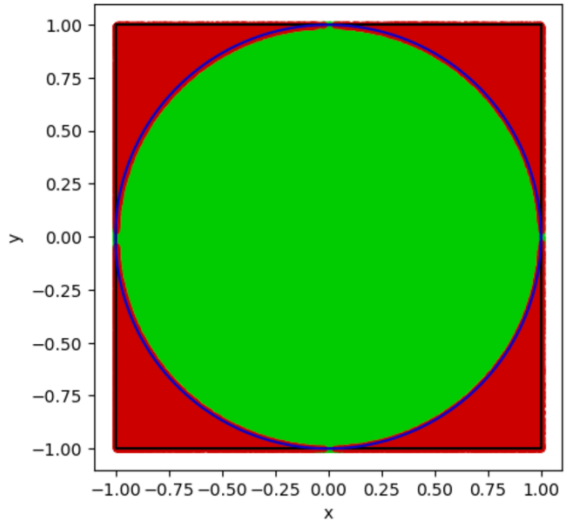
Draw the square over $[-1,1]^2$ with different number of uniformly distributed random samples:



500 samples
$\pi \approx 3.2$
1.8592% off the true value



10,000 samples
$\pi \approx 3.1312$
-0.3308% off the true value



10^6 samples
$\pi \approx 3.142208$
0.0196% off the true value

Python Simulation:

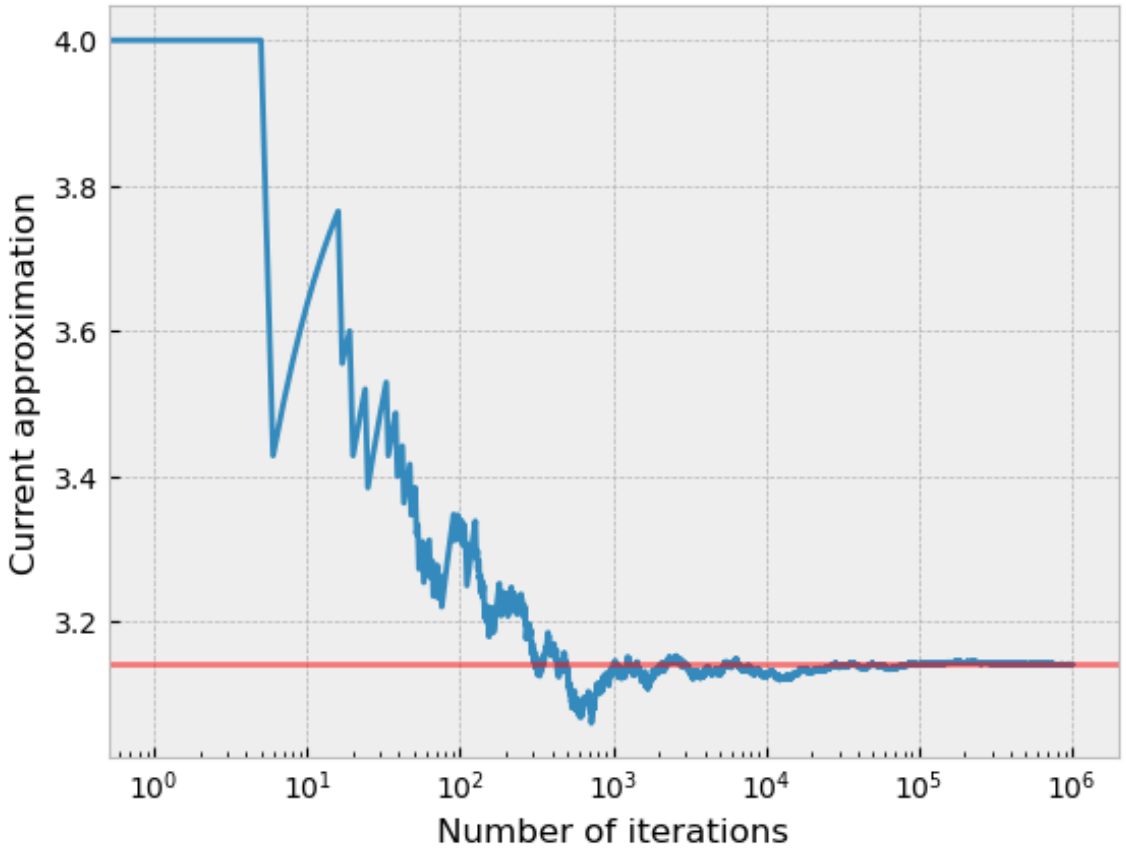
```
import random

def estimate_pi(num_samples):
    inside_circle = 0

    for _ in range(num_ samples):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)

        if x**2 + y**2 <= 1:
            inside_circle += 1

    return 4 * (inside_circle / num_ samples)

num_ samples = 1000000  # Number of samples to take
pi_estimate = estimate_pi(num_ samples)
print("Estimated value of pi:", pi_estimate)
```

-Modified from initial code by ChatGPT

MATLAB Code:

```
function estimated_pi = estimate_pi(num_samples)
    inside_circle = 0;

    for i = 1:num_samples
        x = rand();
        y = rand();
        distance = x^2 + y^2;

        if distance <= 1
            inside_circle = inside_circle + 1;
        end
    end

    pi_estimate = 4 * inside_circle / num_samples;
    estimated_pi = pi_estimate;
end

% Example usage
num_samples = 1000000;  % Number of samples to take
pi_estimate = estimate_pi(num_samples);
disp(['Estimated value of π: ', num2str(pi_estimate)]);
```

Speed of convergence



The error decreases proportionally to $\frac{1}{\sqrt{N}}$

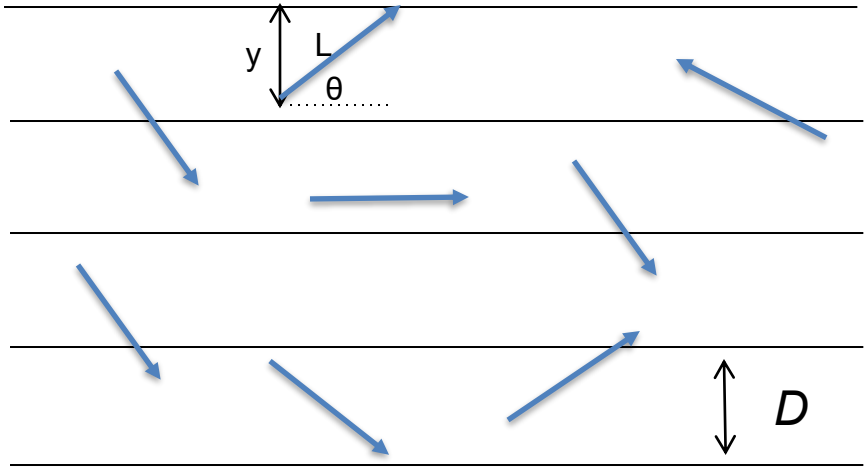
Example: Buffon's needle problem

- This can be used to design a Monte Carlo method for approximating the number π.

$$0 \leq \theta \leq \pi/2$$

A needle intersects a line if:

$$y \leq L \sin\theta$$



The probability of cross a line

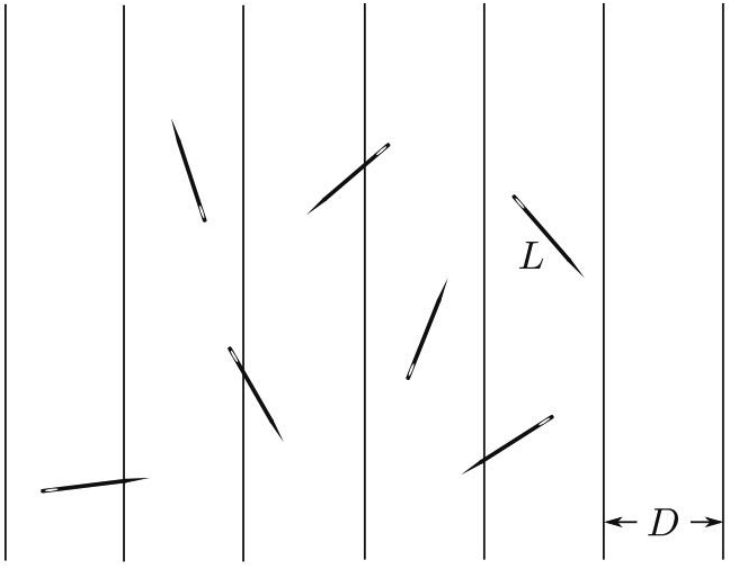
$$P = \frac{2L}{\pi D} \approx \frac{k}{n}$$

So, $\pi = \frac{2L}{DP} \approx \frac{2Ln}{Dk}$

$$0 \leq y \leq D$$

https://demonstrations.wolfram.com/BuffonsNeedleProblem/

History: de Buffon (1733), a French biologist (1707–1788)



- **Convergence**: About N = 100,000 trials are needed for only two digits. Convergence is slow, but foolproof.
- **Transparency**: The method is intuitively understandable, even without any mathematical reasoning.
- **Error estimates, optimization:** Error estimates and optimal choice of L, D are provided by theory of probability. (Binomial distribution, statistical variance as 2nd central moment etc.).

Two dimension version: https://mathworld.wolfram.com/Buffon-LaplaceNeedleProblem.html

Application II: Monte Carlo integration

Foundation 2- Expectation and Integration

Suppose we know the pdf of a random variable X, it is hard to find the pdf of $g(X)$. However, it is easy to calculate the expected value of $g(X)$.

Theorem:

$$E\big(g(X)\big) = \int_{-\infty}^{\infty} g(x)p(x)dx$$

where X is a continuous random variable with density function $p(x)$

Then, by statistical sampling,

$$\int_{-\infty}^{\infty} g(x)p(x)dx \approx \frac{1}{S}\sum_{s=1}^{S} g\big(x^{(s)}\big) \qquad x^{(s)} \sim p(x)$$

Use Monte Carlo, one can approximate the integrals (or very large sums)

Example: Evaluate Integration by Simulating Random Points

- Assume we want to evaluate the integration:

$$\int_0^1 e^{-x^3} dx$$

- An incomplete gamma function, and does not have a closed form solution, so need some techniques to evaluate this number.

The fundamental problem that we therefore wish to address involves finding the expectation of some function $g\ (x)$ (e.g., $g(x) = e^{-x^3}$) with respect to a probability distribution $p(x)$ (e.g., Uniform $p(x) = 1$).

Thus, in the case of continuous variables, we wish to evaluate the expectation.

Rewrite the integration as

$$\int_0^1 e^{-x^3} dx = E\left(e^{-U^3}\right)$$

where U is a **uniform** random variable over the interval $[0, 1]$.

➢ Sampling methods

The general idea behind sampling methods is to obtain a set of samples $z^{(l)}$ (where $l = 1, \ldots, L$) drawn independently from the distribution $g(z)$. **Law of Large Numbers** allows the expectation $E(g(X))$ to be approximated by a finite sum

$$\hat{g} = \frac{1}{L}\sum_{l=1}^{L} g(x^{(l)})$$

- We can generate the IID random variables $U_1, \ldots U_k \sim \text{Uniform}[0,1]$
- Compute $W_1 = e^{-U1^3}$, … $W_k = e^{-Uk^3}$, and finally compute the mean:

$$\overline{W_k} = \frac{1}{k}\sum_{i=1}^{k} W_i = \frac{1}{k}\sum_{i=1}^{k} e^{-Ui^3}$$

- By Law of Large Numbers:

$\overline{W_k}$ converges to $E(W_i) = \mathrm{E}\left(e^{-U^3}\right) = \int_0^1 e^{-x^3} dx$

So this mean can be used as a numerical evaluation of the original integration.

- Generally, the integration can be rewritten as

$$I = \int f(x)p(x)dx$$

Where $f(x)$ is some function, and $p(x)$ is a probability density function.

- Let X be a random variable with density function p. Then

$$I = \int f(x)p(x)dx = \mathrm{E}(\mathrm{f}(\mathrm{X}))$$

- We then can generate IID $X_1, \ldots X_N \sim p$ with N data points, and calculate the sample mean:

$$\bar{I_N} = \frac{1}{N}\sum_{i=1}^{N} f(X_i)$$

- When we change the sampling distribution p, the function f will also change.

Practice Questions: Find the following integrals using MC method:

$$\int_{-1}^{2} \cos^2(x)\sqrt{x^3+1}\,dx \qquad \int_0^1 \frac{e^x-1}{e-1}\,dx$$

Solution:

1. Simulate uniform random variables $X_1, \dots, X_n$ on $[a, b]$
2. Evaluate $g(X_1), \dots, g(X_n)$
3. Take the average of $g(X_1), \dots, g(X_n)$, then

$$(b-a)E\big(g(X)\big) = (b-a)\int_a^b g(x)\frac{1}{b-a}\,dx = \int_a^b g(x)dx$$

is estimated by

$$(b-a)\frac{g(X_1)+\cdots+g(X_n)}{n}$$

Application III: Bayesian inference and learning:

Given some unknown variables $X \in \mathcal{X}$ *and* Y, the following typically intractable integration problems are central to Bayesian statistics

(a) ***Normalization***.

To obtain the posterior $$p(X|Y) = \frac{p(Y|X)p(X)}{p(Y)}$$

given the prior $p(X)$ and likelihood $p(Y \mid X)$, the normalizing factor $p(Y)$ in Bayes' theorem needs to be computed

$$p(Y) = \int_{\mathcal{X}} p(Y|X')p(X')dX'$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} p\left(Y|x^{(s)}\right)$$

(b) **Marginalization**. Compute the marginal posterior

$$p(X|Y) = \int p(X, Z|Y) dZ$$

(c) **Expectation**. The objective of the analysis is often to obtain summary statistics of the form

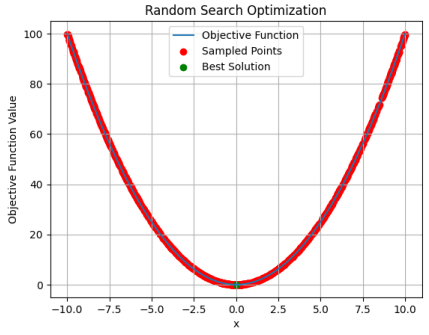
$$E_{p(X|Y)}\big(f(X)\big) = \int_{\mathcal{X}} f(X) p(X|Y) dX$$

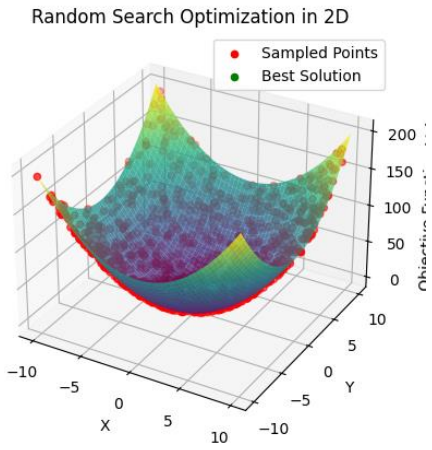
➢ Application IV: Optimization.

Optimization: Minimize (or maximize) functions of some vector that often has many dimensions.

One basic example of applying Monte Carlo methods to optimization is the use of Random Search.

```python
import numpy as np

def objective_function(x):
    return x**2  # Example objective function

def random_search(objective_function, bounds, max_iter=1000):
    best_solution = None
    best_score = float('inf')

    for _ in range(max_iter):
        solution = np.random.uniform(bounds[0], bounds[1])
        score = objective_function(solution)

        if score < best_score:
            best_solution = solution
            best_score = score

    return best_solution, best_score

# Define the bounds of the search space
bounds = (-10, 10)

# Perform random search
best_solution, best_score = random_search(objective_function, bounds)

print("Best solution:", best_solution)
print("Best score:", best_score)
```





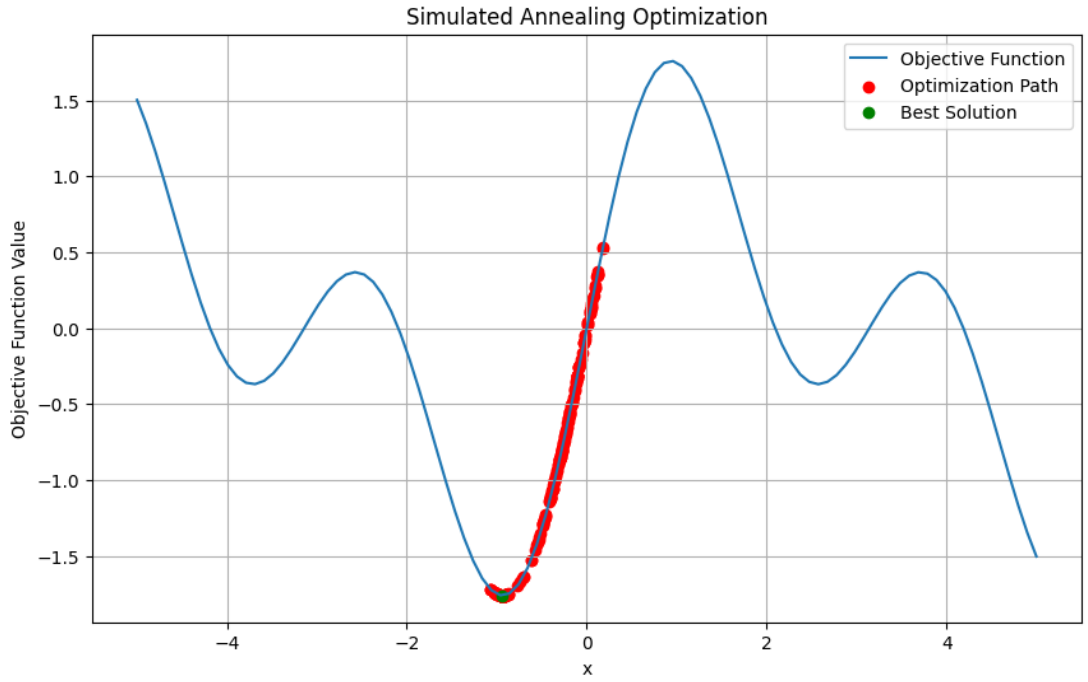
- In Random Search, instead of systematically exploring the entire solution space like traditional optimization algorithms (e.g., gradient descent or Newton's method), random solutions are generated and evaluated iteratively.

- By randomly sampling points from the solution space, the algorithm attempts to find the optimal solution without being restricted to specific search directions.

- While it may seem inefficient, random search can be surprisingly effective, especially in high-dimensional spaces or when the objective function is non-convex and contains many local optima.

Simulated Annealing algorithm

Another application of Monte Carlo methods in optimization is in the context of global optimization, particularly when dealing with complex, multi-modal, or non-convex objective functions.

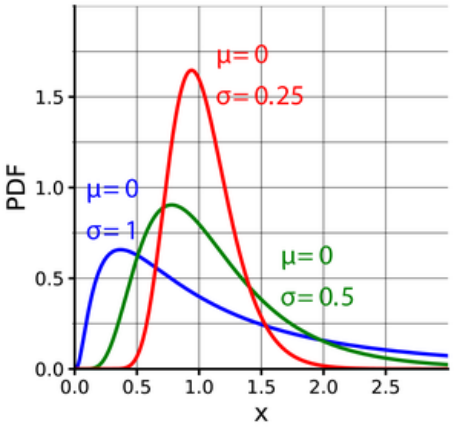


```python
def objective_function(x):
    return np.sin(x) + np.sin(2*x)  # Example objective function

def simulated_annealing(objective_function, bounds, initial_temperature=100, cooling_rate=0.95, num_iterations=1000):
    current_solution = np.random.uniform(bounds[0], bounds[1])
    current_score = objective_function(current_solution)
    best_solution = current_solution
    best_score = current_score
    temperature = initial_temperature
    solution_history = [(current_solution, current_score)]

    for _ in range(num_iterations):
        neighbor_solution = current_solution + np.random.uniform(-0.1, 0.1)  # Generate neighbor solution
        neighbor_score = objective_function(neighbor_solution)

        # Accept the neighbor solution if it improves the objective function or with a certain probability
        if neighbor_score < current_score or np.random.rand() < np.exp((current_score - neighbor_score) / temperature):
            current_solution = neighbor_solution
            current_score = neighbor_score

            # Update the best solution if needed
            if current_score < best_score:
                best_solution = current_solution
                best_score = current_score

        # Cool down the temperature
        temperature *= cooling_rate

        solution_history.append((current_solution, current_score))

    return best_solution, best_score, solution_history
```

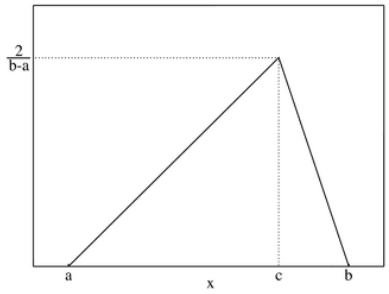
Common Probability Distribution Used in Monte Carlo

- Normal Distribution
- Uniform Distribution
- Exponential Distribution
- Poisson Distribution
- Discrete Distribution

- Lognormal Distribution: continuous probability distribution of a r.v. whose logarithm is normally distributed



- Triangular Distribution: continuous probability distribution with lower limit a, upper limit b and mode c, where $a < b \; and \; a \leq c \leq b$.



➢ Basic Sampling Algorithms

- Standard distributions
- Rejection sampling
- Importance sampling
- Sampling-importance-resampling

Random Number Generation

The Monte Carlo method rests on our ability to produce **random numbers** drawn from any particular probability distribution $F(x)$, or, if it exists, from the probability density function (pdf) $f(x)$ with

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

Step 1. We assume that we can generate iid samples from the uniform distribution on [0, 1].

Step 2. Transform the uniform random numbers into anything else one might want.

Pseudorandom number generator

(Pseudo-random numbers, outside the scope of this course)

https://en.wikipedia.org/wiki/Pseudorandom_number_generator

https://artowen.su.domains/mc/Ch-unifrng.pdf

https://people.smp.uq.edu.au/DirkKroese/ps/montecarlo.pdf

https://www.cs.tufts.edu/~nr/cs257/archive/martin-haugh/generating-random-variables.pdf

Example: Linear Congruential Generator

Set a starting *seed* X_0, and then we generate pseudo-random numbers via a recurrence relation:

$$X_{n+1} = (aX_n + c) \bmod m$$

Take $U_n = \frac{X_n}{m}$, we could generate $\text{Uniform}\,(0,1)$.

➢ Standard distributions (Inverse-Transform Method)

- Generate random numbers from simple **nonuniform** distributions,
- Suppose that Z is **uniformly** distributed over the interval [0, 1],
- The distribution of Y will be governed by

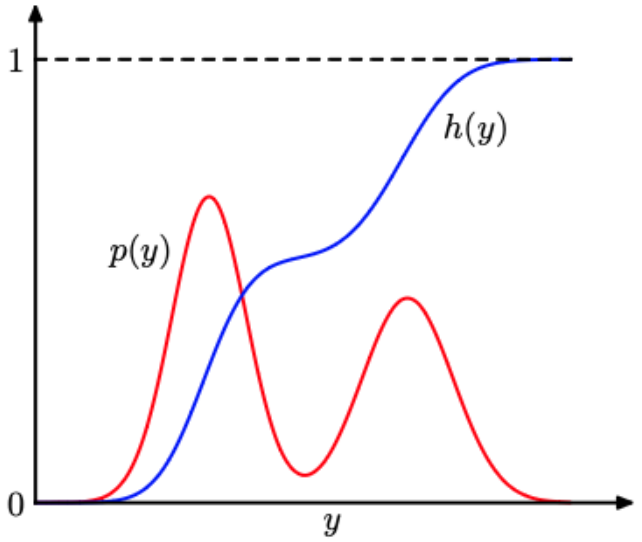
$$p(y) = p(z)\left|\frac{dz}{dy}\right|$$

where in this case $p(z) = 1$.

Take the integral of both sides $dz = p(y)dy$:

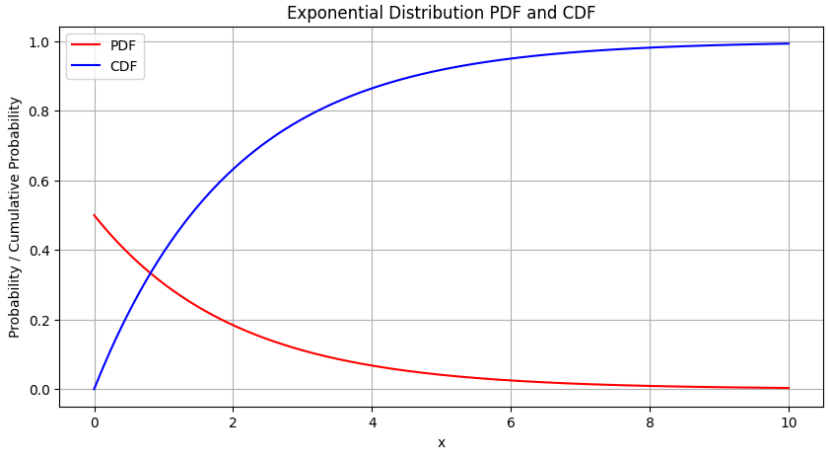
$$z = h(y) \equiv \int_{-\infty}^{y} p(\hat{y})\mathrm{d}\hat{y}$$

- Thus, $\boldsymbol{y = h^{-1}(z)}$, the **inverse of the cdf** of Y.



[Bishop]

Example: Exponential distribution



pdf $p(y) = \lambda \exp(-\lambda y)$

cdf $h(y) = 1 - \exp(-\lambda y)$

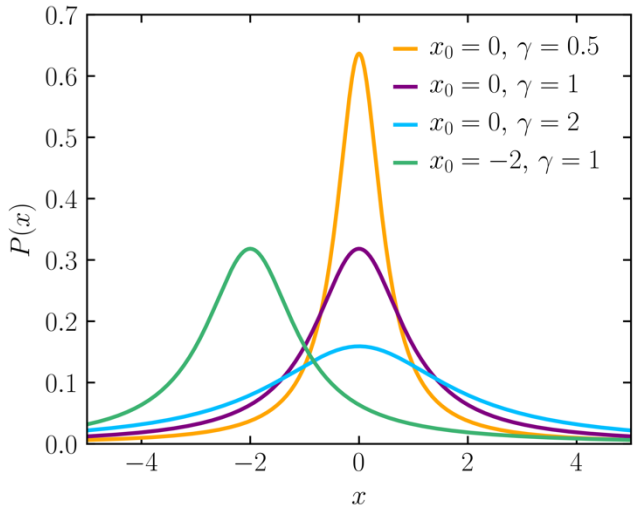
Then, transform our uniformly distributed variable using

$$y = h^{-1}(z) = -\lambda^{-1} \ln(1 - z),$$

where $z \sim \text{Uniform}[0, 1]$,

Example: Cauchy distribution

The **Cauchy distribution** is a continuous probability distribution that has heavy tails and undefined moments beyond the first percentile.



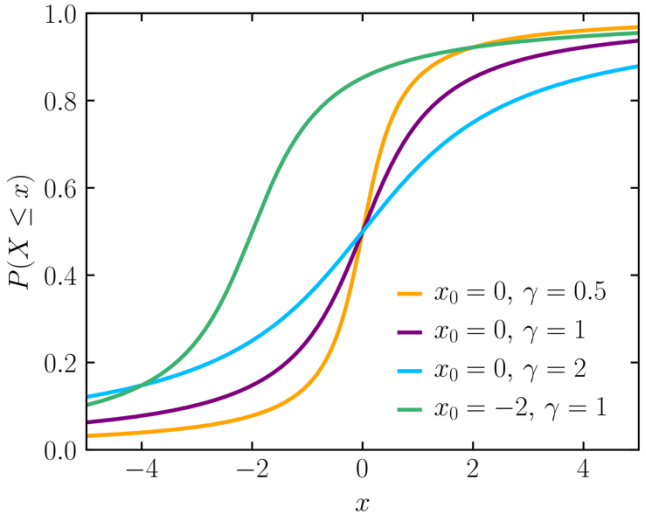
The **pdf** of Cauchy distribution:

$$p(x;\ x_0, \gamma) = \frac{1}{\pi\gamma} \frac{1}{1 + \left(\frac{x - x_0}{\gamma}\right)^2}$$

x_0: the location parameter (median),
γ : the scale parameter (half-width at half-maximum).

The **cdf** of Cauchy distribution:

$$F(x; x_0, \gamma) = \frac{1}{\pi} \tan^{-1}\left(\frac{x - x_0}{\gamma}\right) + \frac{1}{2}$$



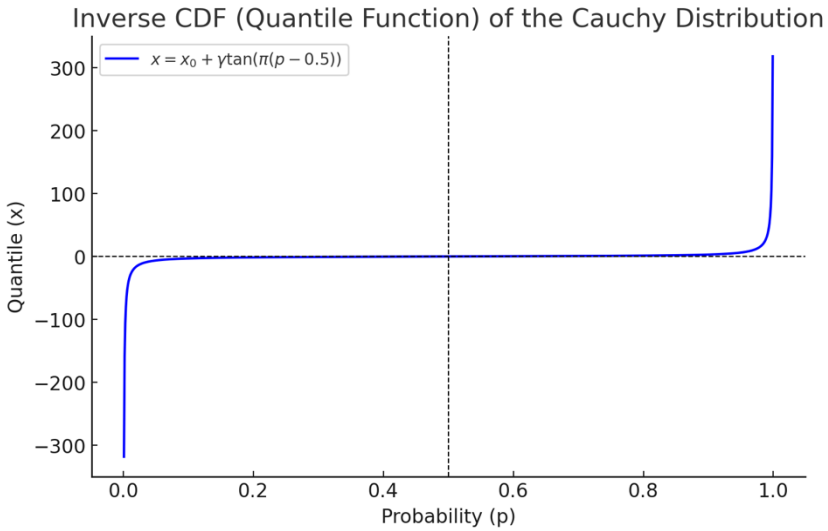
The **inverse CDF** allows us to generate Cauchy-distributed random variables.

$$x = F^{-1}(z) = x_0 + \gamma \tan\left[\pi\left(z - \frac{1}{2}\right)\right]$$

If $Z \sim \text{Uniform}[0, 1]$, then the transform

$$X = x_0 + \gamma \tan\left[\pi\left(Z - \frac{1}{2}\right)\right]$$

follows a Cauchy(x_0, γ) distribution.



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

# Define the inverse CDF (quantile function) for the Cauchy distribution
def cauchy_inverse_cdf(p, x0=0, gamma=1):
    return x0 + gamma * np.tan(np.pi * (p - 0.5))

# Generate probability values between 0 and 1 (excluding 0 and 1)
p_values = np.linspace(0.001, 0.999, 1000)
x_values = cauchy_inverse_cdf(p_values)

# Plot the inverse CDF
plt.figure(figsize=(8, 5))
plt.plot(p_values, x_values, label=r"$x = x_0 + \gamma \tan(\pi (p - 0.5))$", color='b')
plt.axhline(0, color='k', linestyle='--', linewidth=0.8)
plt.axvline(0.5, color='k', linestyle='--', linewidth=0.8)
plt.xlabel("Probability (p)")
plt.ylabel("Quantile (x)")
plt.title("Inverse CDF (Quantile Function) of the Cauchy Distribution")
plt.legend()
plt.grid()

# Show the plot
plt.show()
```

➢ Multiple Variables

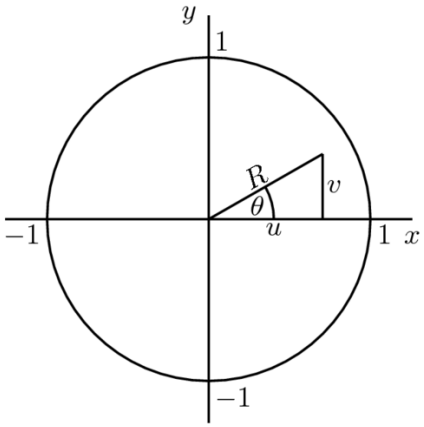
Jacobian of the change of variables:

$$p(y_1, \dots, y_M) = p(u_1, \dots, u_M) \left| \frac{\partial(u_1, \dots, u_M)}{\partial(y_1, \dots, y_M)} \right|$$

where $u_i \sim \text{Uniform}[0, 1]$.

Example: The joint pdf of two independent normal variables (Z_1, Z_2) is

$$f(z_1, z_2) = \frac{1}{2\pi} e^{-\frac{z_1^2 + z_2^2}{2}}$$



$$R^2 = u^2 + v^2$$
$$\cos\theta = \frac{u}{R}$$
$$\sin\theta = \frac{v}{R}$$

Transform the problem into **polar coordinates**:

$$Z_1 = R\cos\Theta \qquad Z_2 = R\sin\Theta$$

where R is the **radius** and Θ is the **angle** in a 2D plane.

Then $R^2 = Z_1^2 + Z_2^2$, then

$$f(R, \Theta) = \frac{1}{2\pi} e^{-\frac{R^2}{2}} R$$

- Θ is **uniformly** distributed between 0 and 2π.
- R follows the Rayleigh distribution **pdf**:

$$f(R) = Re^{-\frac{R^2}{2}}$$

with cdf $F(R) = 1 - e^{-R^2/2}$ and inverse transform

- $R = \sqrt{-2 \ln U_1}$, where $U_1 \sim \text{Uniform}[0, 1]$
- $\Theta = 2\pi U_2$, where $U_2 \sim \text{Uniform}[0, 1]$

Example: Multi-Normal distribution (Box-Muller method**)**

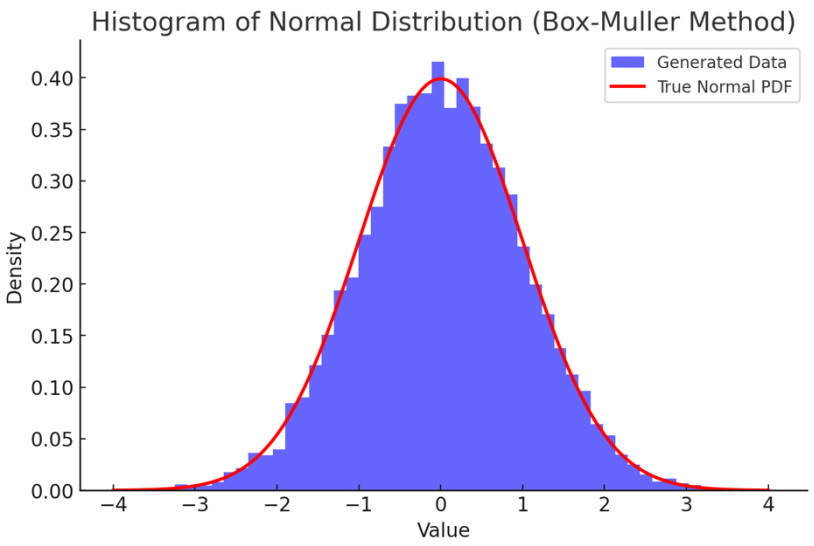
The Box-Muller method converts two independent uniform random variables into two **independent** standard normal random variables.

$U_1 \sim \text{Uniform}[0,1]$ and $U_2 \sim \text{Uniform}[0,1]$ are independent.

Let $Z_1 = R\cos\Theta = \sqrt{-2\ln U_1}\cos(2\pi U_2)$

$Z_2 = R\sin\Theta = \sqrt{-2\ln U_1}\sin(2\pi U_2)$

Then, Z_1 *and* Z_2 are independent standard normal random variables.

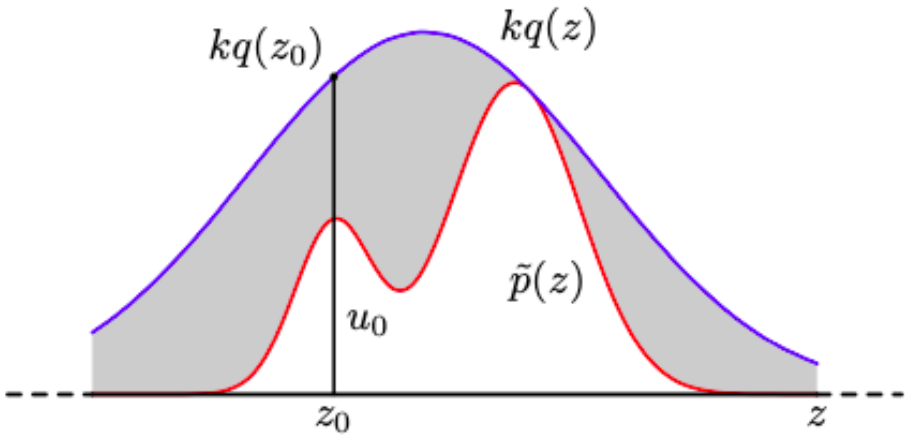


❑ Rejection sampling

- The rejection sampling framework allows us to sample from relatively complex distributions, subject to certain constraints.
- To apply rejection sampling, some simpler distribution $q(z)$, sometimes called a ***proposal*** *distribution*.
- It satisfies

$$p(z) \leq kq(z), \qquad \text{for } k < \infty$$

for all values of z



The Algorithm

1. Sample $u \in \text{Uniform}(0,1)$

2. Sample a candidate $z \in q(z)$ from the simpler distribution

3. If $u \leq \frac{p(z)}{kq(z)}$, then accept the candidate z.

   Otherwise, reject z and go back to step one.

The algorithm can be repeated until the desired number of samples from the target density.

Limitations:

$$P(all\ z \text{ is accepted}) = P\left(u \le \frac{p(z)}{kq(z)}\right) = \int P\left(u \le \frac{p(z)}{kq(z)} | X = z\right) q(z)dz$$

$$= \int \frac{p(z)}{kq(z)} q(z)dz$$

$$= \frac{1}{k}$$

Here, $P(\text{accept} | \text{X} = \text{z}) = \frac{p(z)}{kq(z)}$

- It is not always possible to bound $p(z)/q(z)$ with a reasonable constant k over the whole space Z.
- If k is too large, the acceptance probability is too small.
- In high dimensional spaces it can be exponentially slow to sample points. (The points usually will be rejected)

❑ Importance Sampling

- One of the principal reasons for wishing to sample from complicated probability distributions is to be able to evaluate expectations. The technique of importance sampling provides a framework for approximating expectations directly but does not itself provide a mechanism for drawing samples from distribution $p(z)$.
- As in the case of rejection sampling, importance sampling is based on the use of a proposal distribution $q(z)$ from which it is easy to draw samples. We can then express the expectation in the form of a finite sum over samples $\{z^{(l)}\}$ drawn from $q(z)$

$$E[f] = \int f(z)p(z)dz$$

$$= \int f(z)\frac{p(z)}{q(z)}q(z)dz$$

$$E(f) \approx \frac{1}{L}\sum_{i=1}^{L}\frac{p(z^{(l)})}{q(z^{(l)})}f(z^{(l)})$$

The quantities $r_l = \frac{p(z^{(l)})}{q(z^{(l)})}$ are known as importance weights, and they correct the bias introduced by sampling from the wrong distribution.

Note that, unlike rejection sampling, all of the samples generated are retained.

❑ Importance Sampling (further examples)

It will often be the case (e.g., Bayesian logistics regression) that the distribution $p(z)$ can only be evaluated up to a normalization constant, so that

$$p(z) = \tilde{p}(z)/Z_p$$

where $\tilde{p}(z)$ can be evaluated easily, whereas Z_p is unknown.

Similarly, we may wish to use an importance sampling distribution

$$q(z) = \tilde{q}(z)/Z_q,$$

which has the same property. We then have

$$E[f] = \int f(z)p(z)\,dz$$

$$= \frac{Z_q}{Z_p}\int f(z)\frac{\tilde{p}(z)}{\tilde{q}(z)}q(z)dz$$

$$= \frac{Z_q}{Z_p}\frac{1}{L}\sum_{i=1}^{L}\tilde{r}_l f\left(z^{(l)}\right)$$

- where $\tilde{r}_l = \left(\tilde{p}(z)\right)/\left(\tilde{q}(z)\right)$
- We can use the same sample set to evaluate the ratio Z_p/Z_q with the result

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q}\int \tilde{p}(z)\,dz = \int \frac{\tilde{p}(z)}{\tilde{q}(z)}q(z)dz$$

$$= \frac{1}{L}\sum_{i=1}^{L}\tilde{r}_l$$

Hence

$$E[f] = \sum_{i=1}^{L} w_l f\left(z^{(l)}\right)$$

where we have defined

$$w_l = \frac{\tilde{r}_l}{\sum_m \tilde{r}_m} = \frac{\tilde{p}\left(z^{(l)}\right)/q\left(z^{(l)}\right)}{\sum_m \tilde{p}(z^{(m)})/q(z^{(m)})}$$

Remarks:

- The rejection sampling method discussed depends in part for its success on the determination of a suitable value for the constant k. For many pairs of distributions $p(z)$ and $q(z)$, it will be impractical to determine a suitable value for k in that any value that is sufficiently large to guarantee a bound on the desired distribution will lead to impractically small acceptance rates.

- As in the case of rejection sampling, the sampling-importance-resampling (SIR) approach also makes use of a sampling distribution $q(z)$ but avoids having to determine the constant k. There are two stages to the scheme. In the first stage, L samples $z^{(1)}, \ldots z^{(L)}$ are drawn from q(z). Then in the second stage, weights $w_1, \ldots, w_L$ are constructed. Finally, a second set of L samples is drawn from the discrete distribution $(z^{(1)}, \ldots z^{(L)})$ with probabilities given by the weights $(w_1, \ldots, w_L)$.

Integrals, often expectations, occur frequently in statistics:

- **Monte Carlo sampling** approximates expectations with a sample average.
- **Rejection sampling** draws samples from complex distributions.
- **Importance sampling** applies Monte Carlo to `any' sum/integral.

Monte Carlo Simulation- Advantages & Limitations

- **Advantages:**
  - Easy and efficient
  - Solve complex problems simply
  - Randomness as a strength

- **Limitations:**
  - Processing power
  - Sensitivity to the chosen probability distribution
  - Not to work on high-dimensional integration or more complex problems

"Monte Carlo is an extremely bad method; it should be used only when all alternative methods are worse." -Alan Sokal, 1996 MC in Statistical Mechanics

❖References:

Pattern Recognition And Machine Learning – Chris Bishop - Springer 2006

An interactive visualization of many of these algorithsm in 2d, see
https://github.com/chi-feng/mcmc-demo?tab=readme-ov-file

UCLA Stat 202C: Monte Carlo Methods for Optimization
http://www.stat.ucla.edu/~sczhu/Courses/UCLA/Stat_202C/Stat_202C.html (Theory)
Book: Monte Carlo Methods
https://doi.org/10.1007/978-981-13-2971-5 (Free download through Northeastern Library website.)

Dongarra J, Sullivan F (2000) Guest editors introduction: the top 10 algorithms. Comput Sci Eng 2(1):22–23
https://www.cs.fsu.edu/~lacher/courses/COT4401/notes/cise_v2_i1/index.html

Andrieu, C.(2003). An Introduction to MCMC for Machine Learning: https://www.cs.ubc.ca/~arnaud/andrieu_defreitas_doucet_jordan_intromontecarlomachinelearning.pdf

Neal, R. (1995). Suppressing Random Walks in Markov Chain Monte Carlo Using Overrelaxation: https://arxiv.org/pdf/bayes-an/9506004.pdf

Tang, Y. (2022). A Note on Monte Carlo Integration in High-Dimensions: https://arxiv.org/pdf/2206.09036.pdf