# Statistical Methods
# (summer term 2024)

# Monte Carlo Techniques

(based on original lectures by Prof. Dr. N. Christlieb and Dr. Hans-G. Ludwig)

Dr Yiannis Tsapras

ZAH – Heidelberg

# Overview

- Monte Carlo simulation "light": means to test a statistic
  - properties of estimators
- Monte Carlo integration
  - how does it work?
  - why use it?
  - what accuracy can be achieved?
- Random numbers, sampling from arbitrary distributions
  - exact inversion
  - rejection method
  - Metropolis-Hastings algorithm
- Monte Carlo integration based on lecture notes of Volker Springel (see `Notes_MC.pdf` on web)
- Refer to "Numerical Recipes" for comprehensive coverage of MC integration

# What are Monte Carlo (MC) methods?

- Monte Carlo methods are a class of techniques for randomly sampling a probability distribution

- There are many problems in probability, and more broadly in machine learning, where we cannot calculate an analytical solution directly

- The basic idea of MC is to draw samples randomly from the underlying probability distribution and use them to approximate the desired quantity

- Applications of MC methods include:
  - Numerical integration
  - Optimization problems
  - Simulating physical and mathematical systems . . .

- Key components of MC:
  - Random number generation
  - Sampling from probability distributions
  - Statistical analysis of results

# Monte Carlo simulation as means to test a statistic

- Growing computing power made Monte Carlo methods increasingly important in statistics

- Here: testing a statistic (function of the random values of a given sample)
  - basic idea: simply try it out
  - avoids problems of analytical treatment, e.g. sometimes only asymptotic results for large sample sizes can be obtained
  - in principle, it can provide exact properties
  - disadvantage: an MC simulation is a statistical procedure and one needs to control for random errors

- In practice, one needs to work with sufficiently large samples to minimize errors
  - but how large is sufficiently large?
  - simplest approach: repeat MC simulation with different random numbers and see up to which precision results remain the same
  - more rigorously: use the numerical PDF of the statistics one usually obtains in an MC run; from the PDF one can derive estimates of the variance of the quantity of interest

# Properties of a statistic (or estimator)

- Consider a **sample** $x_1, \ldots, x_n$ drawn from a **population** characterized by some parameters (e.g. normal distribution by mean $\mu$ and variance $\sigma^2$)

- A statistic (Schätzfunktion) or estimator (Schätzer) is a function $g(x_1, \ldots, x_n)$ of the sampled values for a parameter $u$ of the population

- An **estimator** $g(x_1, \ldots, x_n)$ of a parameter $u$ is ...
  - *consistent* (dt. konsistent) if $\lim_{n\to\infty} g(x_1, \ldots, x_n) = u$
  - e.g. Suppose $g(x_1, \ldots, x_n)$ is the sample mean $\bar{x}$. As $n \to \infty$, $\bar{x}$ approaches the population mean $\mu$
    For large $n$ the estimate approaches the true value of the parameter
  - *unbiased* (dt. erwartungstreu) if $\mathrm{E}[g(x_1, \ldots, x_n)] = u$
  - e.g. for the sample mean $\bar{x}$, the expectation $E[\bar{x}] = \mu$, making it an unbiased estimator of $\mu$
    Independent of sample size, the estimate corresponds *on average* to the true value

# Properties of a statistic (or estimator)

- An **estimator** $g(x_1, \ldots, x_n)$ of a parameter $u$ is ...
  - (relatively) *efficient* (dt. wirksam) if $\mathrm{E}\big[(g(x_1, \ldots, x_n) - u)^2\big]$ is small/smallest
  - e.g. The **sample** variance $S^2$ is *efficient* if it has a smaller mean square error compared to other estimators of the **population** variance $\sigma^2$

    On average, the mean square deviation between estimated and true value is small. Sometimes, the term "efficient" is reserved for the estimator for the smallest variance as given by the relation above
  - *robust* if the estimate is weakly influenced by outliers, i.e. samples which were in fact *not* drawn from the assumed population
  - e.g. The sample median is *robust* because it is less influenced by extreme values (outliers) compared to the sample mean
- In MC simulations the expectation values can be computed (or rather estimated) as averages over a sufficiently large number of realizations

# Why divide by $n-1$ and not $n$ for the sample variance?

- Having a **sample** $x_1, \ldots, x_n$ drawn from a **population**, one can *estimate* the true mean $\mu$ of the underlying PDF of the whole population by using the sample mean $\bar{x}$

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Furthermore, one can get an estimate of the true variance $\sigma^2$ of the population, by using the sample variance $S^2$

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

- Why do we use $n-1$ here?

- Answer: this makes the estimator *unbiased*!

(proof $\rightarrow$ blackboard)

# Example: comparison of various estimators for the variance of populations with normal or uniform distributions

- Compare the following four estimators for the variance of two populations: the first is normally distributed $N(0,1)$, the other uniformly distributed $U(0,1)$
  ($\rightarrow$ `MCexample4.R`)

$$S^2 = \frac{1}{n+k}\sum_{i=1}^{n}(x_i - \bar{x})^2 \quad \text{with} \quad k = -1, 0, +1, +2$$

  The population mean $\mu$ is estimated from the sample mean $\bar{x}$ via $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

  Use $n = 5$ and 100000 realizations; the R-functions `matrix()`, `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()` are your friends

- How large is the bias of the estimators? Is there an unbiased estimator among them?

- Which estimator is most efficient?

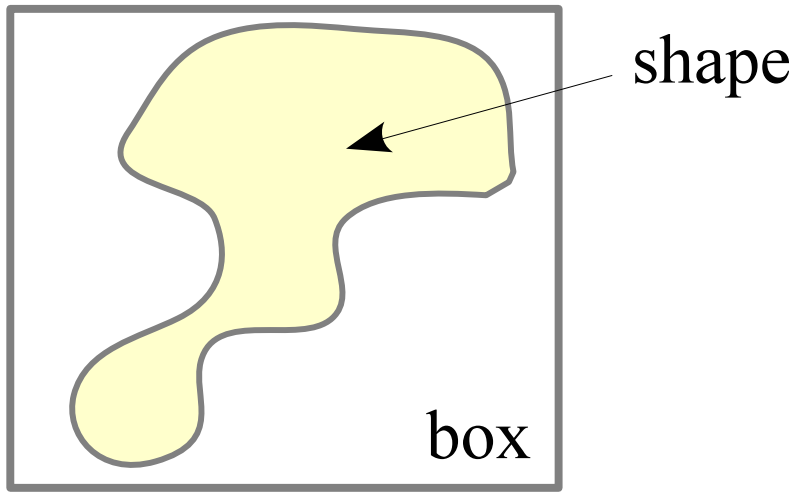
- Which estimator would you choose to estimate the population variance?

# Monte Carlo integration

- What is Monte Carlo Integration?
  - Monte Carlo integration is a statistical technique used to evaluate the integral of a function over a given domain using random sampling
- How does it work?
  - randomly sample points within the domain and evaluate the function at these points
  - The integral is approximated by the average value of the function at the sampled points, multiplied by the volume of the domain
- Monte Carlo integration is particularly useful for high-dimensional integrals where traditional numerical integration methods become inefficient or impractical
- Random number generation and sampling from probability distributions are crucial components
  - Statistical analysis of the results to estimate accuracy and error

# Basic idea of MC integration

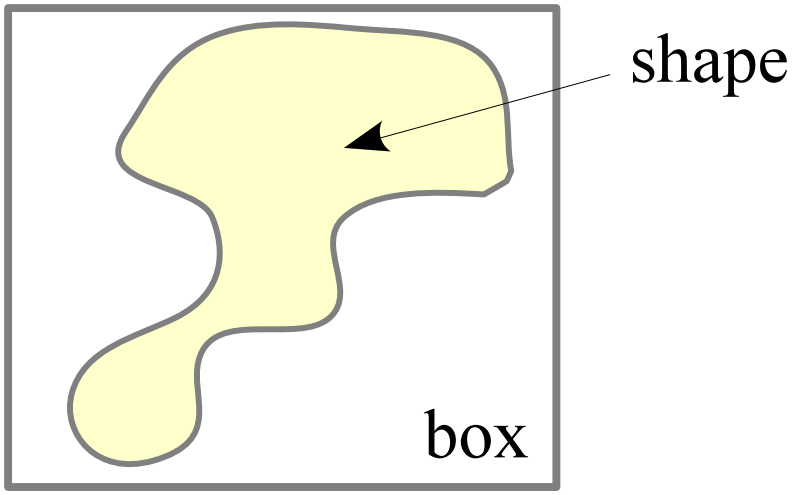
- Intuition: "dartboard method" of integrating the area of an irregular domain




- Choose points randomly (i.e. uniformly) within the box
  - Count the number of points that fall inside the shape
- The probability that a point lands inside the area is given by the ratio of the areas:

$$p(\text{point lands inside area}) = \frac{A_{\text{shape}}}{A_{\text{box}}}$$

# Basic idea of MC integration



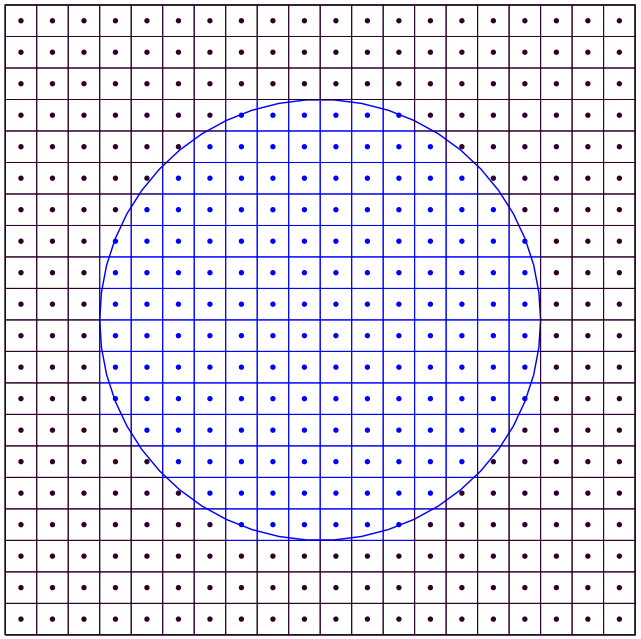

- We can now approximate this probability, and hence the area ratio, through the experimental result:

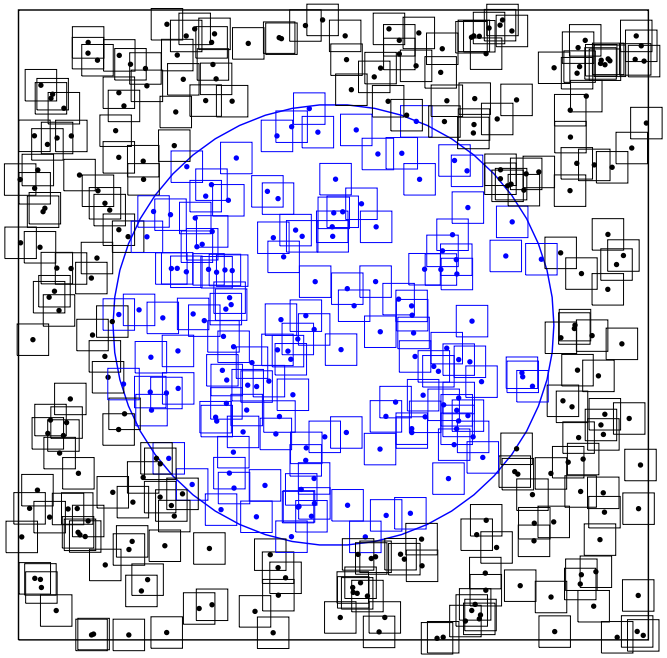
$$\frac{A_{\text{shape}}}{A_{\text{box}}} \simeq \frac{\#\text{points landed in shape}}{\#\text{points landed in box}}$$

- This is expected to become arbitrarily accurate as the number of trials goes to infinity

- This method can be used to integrate functions over more complex domains

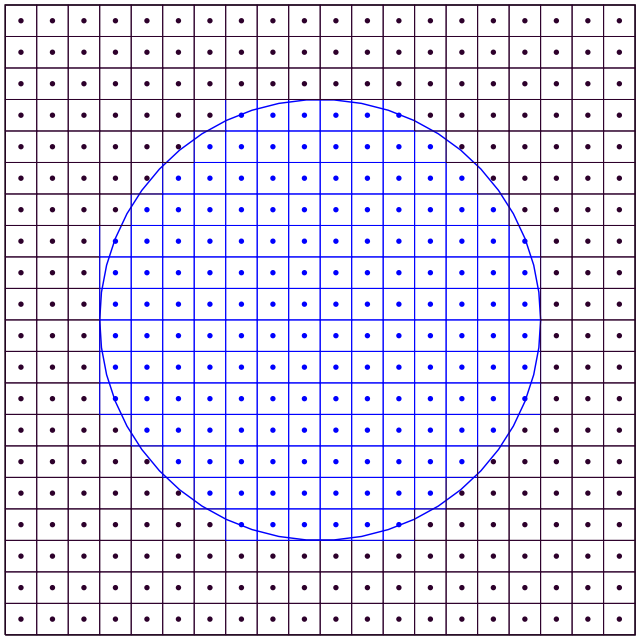
# Why is MC integration useful?

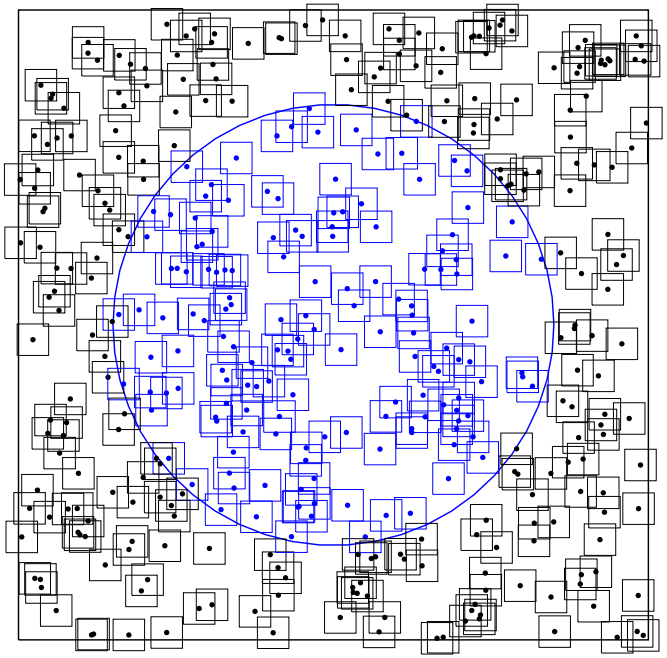




- Comparison of Integration Methods:
  - **Regular Tiling**: Divides the domain into regular, equal-sized surface elements
  - **Monte Carlo Tiling**: Uses random sampling to represent surface elements
  - Each MC sample represents a small surface element (e.g., 400 samples)
  - Smaller surface elements can better follow curved shapes
  - MC samples leave gaps and often overlap →does not provide a nice representation of the Riemann-sum underlying integration

# Why is MC integration useful?





- So why consider using MC Integration at all?
  - Extremely effective for high-dimensional integrals where regular methods become impractical
  - Flexibility in handling complex and irregular domains
  - Statistical properties allow for error estimation and confidence intervals

# Standard Monte Carlo integration

- Formally: Consider the integral in $d$-dimensions:

$$I = \int_V f(\vec{x}) \, \mathrm{d}^d \vec{x}$$

where $V$ is a $d$-dimensional hypercube with dimensions $[0,1]^d$ for simplicity

- Procedure:
  - Generate $N$ random vectors $\vec{x}_i$ with components drawn uniformly from the interval $[0,1]$. Each component of the vector is drawn independently from a uniform distribution
  - Approximate the integral $I_N$ by summing the function values at these random points. Each sample represents a volume element of size $V/N$:

$$I_N = \frac{V}{N} \sum_{i=1}^{N} f(\vec{x}_i)$$

  - As $N \to \infty$, the approximation $I_N$ converges to the true integral $I$

# Standard Monte Carlo integration

- Uncertainty ('error') Estimation:
  - The uncertainty of the result scales as $1/\sqrt{N}$, which is independent of the number of dimensions of the integral
  - This means that even for high-dimensional integrals, the accuracy of Monte Carlo integration improves as the number of samples increases

Let's see how that compares with non-MC integration...

# Comparison with non-MC integration

- Divide each dimension in $n$ *regularly* spaced intervals, hence $N = n^d$
- Depending on the integration rule the error will scale as some power of $1/n$
  - midpoint rule, horizontal line through $f(\frac{a+b}{2})$, error $\propto 1/n^2$
  - trapezoidal rule, secant through $f(a)$ and $f(b)$, error $\propto 1/n^2$
  - Simpson's rule, parabola through $f(a), f(\frac{a+b}{2}), f(b)$, error $\propto 1/n^4$
- $d$ small $\rightarrow$ MC integration has larger errors than standard methods with same $N$
- $d$ large $\rightarrow$ MC integration advantageous, standard methods can spend comparatively fewer regular sampling points per dimension
- Example: at what point is MC as good as midpoint?

$$\frac{C_{\mathrm{midpoint}}}{N^{2/d}} \approx \frac{C_{\mathrm{MC}}}{N^{1/2}}$$

  - starts to decline with $N$ more weakly than MC integration when $d \gtrsim 4$
- MC integration only option for high dimensional problems (e.g. thermodynamics)

# 'Error' estimate in Monte Carlo integration

- Let $y \equiv V f(\vec{x})$ and $y_i = V f(\vec{x}_i)$ is the value of the $i$-th function evaluation in the MC integration; each $y_i$ estimate of integral
  - reminder: $V$ represents the volume of the integration domain
- Taking $N$ uniformly distributed samples, the desired integral is approximated by

$$I_N = \frac{y_1 + y_2 + \ldots + y_N}{N}$$

- Concerning the numerical error, one may ask
  - does $I_N$ provide an accurate approximation to the integral of $f$?
  - what is the difference between $I_N$ and the actual integral of $f$?
- Statistically speaking, if we repeat the MC integration a large number of times ...
  - what is the expectation value of the distribution of $I_N$
  - what is the variance of the distribution of $I_N$ → blackboard

# 'Error' estimate in Monte Carlo integration

- $I_N$ is the sum of identically distributed random variables (divided by $N$)
- For large $N$ the *central-limit-theorem* gives the answer: With

  $$y = V f(\vec{x})$$

  we have

  $$\langle y \rangle = \langle V f \rangle = V \langle f \rangle = I \quad \text{and} \quad \mathrm{Var}[y] = \mathrm{Var}[V f] = V^2 \mathrm{Var}[f] \ ,$$

  so that

  $$I_N \approx \frac{N \langle y \rangle}{N} = I \quad \text{and} \quad \mathrm{Var}[I_N] \approx N \, \mathrm{Var}\Big[\frac{y}{N}\Big] = \frac{\mathrm{Var}[y]}{N} = \frac{V^2}{N} \mathrm{Var}[f]$$

- This involves the exact moments of $f$ $\langle f \rangle$ and $\langle f^2 \rangle$ which can be estimated from the MC samples . . .

# Summary on 'Error' estimate in Monte-Carlo integration

- For standard Monte Carlo integration with $N$ samples, the error is

$$\sigma_{I_N} = V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}$$

where $\langle f\rangle$ and $\left\langle f^2\right\rangle$ are exact moments of the function we integrate, i.e.

$$\langle f\rangle \equiv \frac{1}{V}\int f(x)\mathrm{d}x \qquad\qquad \left\langle f^2\right\rangle \equiv \frac{1}{V}\int f^2(x)\mathrm{d}x$$

- In practice we can estimate these moments from the Monte-Carlo samples themselves, i.e. we estimate

$$\langle f\rangle \approx \frac{1}{N}\sum_i f(x_i) \qquad\qquad \left\langle f^2\right\rangle \approx \frac{1}{N}\sum_i f^2(x_i)$$

and then use these moments to estimate the error $\sigma_{I_N}$

# Interlude: generating random numbers

- Random numbers play a central role in Monte Carlo techniques
- Usually, they are produced by *deterministic algorithms* leading to *pseudo-random numbers*, and are suitable for MC techniques (use same seed value→reproducibility)
- Truly random numbers, on the other hand, are generated by some physical process (like rolling the dice, radioactive decay, quantum transitions, etc.).
- Some CPUs include a hardware random number generator, based on coupled non-linear oscillators and additional sources of entropy
  - These generators are normally $not$ used for MC techniques, because ...
    - ⋆ sequence is not repeatable, making debugging difficult and preventing reproducibility
    - ⋆ they tend to be slow(er)
    - ⋆ the quality of the distribution may not be perfect
    - ⋆ the quality of the distribution may degrade with time, or correlate in subtle ways with environmental factors such as system temperature, etc.

# Interlude: generating random numbers

There is value in having good random numbers. In 1950, the RAND corporation published a book entitled "1 million random digits", whose primary virtue is to contain *no discernable information at all*. This classic is available online (`http://www.rand.org/publications/classics/randomdigits`). While it may not be fun to read, it serves as an important historical reference in the generation of random numbers.

# Interlude: pseudo-random number generators

- Usually create an integer sequence that is then converted to a floating point number in the interval $[0, 1]$

- Essential desirable properties of a good random number generator ...
  - repeatability: for the same seed, we want to obtain the same sequence of random numbers
  - randomness: good random numbers should be...
    - ⋆ uniformly and homogeneously distributed in the interval $[0, 1]$
    - ⋆ independent of each other, i.e. show no correlations whatsoever (this is non-trivial and not exactly true for pseudo-random number generators)
  - speed: sometimes billions of random numbers needed
  - portability: same results on different computer architectures
  - long period: after a finite number of pseudo-random numbers, the sequence repeats. This period should be as large as possible to avoid repeated patterns
  - insensitivity to seed: neither the period nor the quality of the randomness should depend on the value of the seed, i.e. on where the sequence is started

- Moral: random number generation is a non-trivial task. In case of doubt, one needs to test a generator. We won't go deeper into the details here

# Interlude: random sequences and random numbers in R

- `sample()` draws a sample from a population with or without replacement

```
sample(10)                  # sample a permutation of numbers 1 to 10
sample(x=10, size=3)        # draw 3 samples from 1 to 10 (no replacement)
sample(x=10, size=3)        # draw another 3 samples (no replacement)
set.seed(100)               # controls the (pseudo) random number sequence
sample(x=10, size=3)        # draw 3 samples <- uses seed (no replacement)
set.seed(100)               # reset pseudo randon number generator
sample(x=10, size=3)        # draw the same 3 samples (reprodicibility)
sample(c(1,5,8,13,6,27), size=4) # draw 4 from given vector (no repl.)
sample(c(1,5,8,13,6,27), size=4, replace=TRUE) # draw 4 with replace
```

- Using `set.seed()` is crucial for reproducibility in simulations. By setting the same seed, you ensure that the random number sequence is identical each time, which is essential for debugging and replicating results

Try it out!

# Interlude: random sequences and random numbers in R

- Sampling from a probability density distribution $p(x)$
  - pick $x$ with a probability $p(x)\mathrm{d}x$ ($\rightarrow$ drawing)

- R provides random variables from the standard univariate PDFs using functions like `rnorm, runif, rbinom, rpois, , ...`

```
x <- rnorm(1000, mean=2, sd=0.5)n # draw 1000 samples
library(MASS) # load necessary library for histograms
truehist(x, ylim=c(0,1)) # plot histogram of samples
line(x, dnorm(x, mean=2, sd=0.5)) # overplot the true distr.
curve(dnorm(x, mean=2, sd=0.5), add=TRUE, col="blue", lwd=2)
```

- To generate random variables from a multivariate normal distribution, use: `mvrnorm{MASS}` or `rmvnorm{mvtnorm}`

Try it out!

# Example 1: standard Monte Carlo (MC) integration

Compute the integral

$$I = \int_0^1 f(x)\,\mathrm{d}x = \int_0^1 \left(x^{-1/3} + \frac{x}{10}\right)\,\mathrm{d}x$$

by standard MC integration. The integral can be solved analytically and has the value $I = 31/20 \approx 1.550$, so we don't really need MC integration here. However, we can test the numerical result. The variance of the integrand turns out to be $\approx 0.849$ so that we expect for the standard deviation of the width of the distribution of $I$

$$\sigma_I = \sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}} \approx \frac{0.849}{\sqrt{N}}.$$

- Plot the function $f$. Are there any problems?
- Set the number of samples to $N = 10000$ and perform $M = 1000$ Monte Carlo integrations. Plot the histogram of integrals and estimate the dispersion. Does it correspond to the analytical expectations? ($\rightarrow$`MCexample1.ipynb`)
- Does the integral evaluate correctly?

Hint: the functions `hist()`,`mean()`,`sd()`,`runif()` may help

# Interlude: Drawing from arbitrary distributions

- As we shall see, we sometimes need to draw from arbitrary distributions
- While programming environments offer a limited number of well-known distributions (e.g., normal, uniform, binomial), they might not be sufficient for specific applications
- There are various methods for generating samples from a given distribution:
  - Inverse Transform Sampling: Uses the cumulative distribution function (CDF) to generate samples
  - Rejection Sampling: Uses a proposal distribution and accepts/rejects samples based on a criterion
  - Importance Sampling: Weights samples according to their importance in the target distribution
  - Markov Chain Monte Carlo (MCMC): Generates samples by constructing a Markov chain that has the desired distribution as its equilibrium distribution
- Each method has its own advantages and use cases depending on the properties of the distribution and the computational resources available

# Exact Inversion of Cumulative PDF

- Available: samples $x_i$ from distribution $p_1(x)$
- Needed: samples $y_i$ from distribution $p_2(y)$
- Looking for function $y(x)$ that transforms $x_i$ into $y_i$, such that $p_2(y) = p_1(x(y))$
- PDFs satisfy conditions $\int p(x)\,\mathrm{d}x = 1$ and $p(x) \geq 0$ for all $x$. Conservation of probability (or the value of the integral) can be stated as

  $$|p_1(x)\,\mathrm{d}x| = |p_2(y)\,\mathrm{d}y| \quad \rightarrow \quad \text{functional determinant}$$

  (here a modulus has been added to neutralize a possible sign change due to the mapping)
- The cumulative distribution functions (CDFs) for $p_1(x)$ and $p_2(y)$ are given by:

  $$F_1(x) = \int_{-\infty}^{x} p_1(u)\,\mathrm{d}u \qquad\qquad F_2(y) = \int_{-\infty}^{y} p_2(v)\,\mathrm{d}v$$

  with $u, v$ as dummy variables of integration

# Exact Inversion of Cumulative PDF

- Matching CDFs: Since $F_1(x)$ and $F_2(y)$ are the cumulative probability distribution functions of $p_1$ and $p_2$, respectively, they must be equal for corresponding $x$ and $y$ values: $F_1(x) = F_2(y)$
  - The key idea here is that the CDFs of the initial and target distributions are related

- To find $y$ corresponding to $x$, we use the inverse CDF (quantile function) of the target distribution, so that

$$y = F_2^{-1}\left[F_1(x)\right]$$

- If $p_1(x)$ is a uniform distribution in $[0, 1]$, then $F_1(x) = x$, so the transformation simplifies to:

$$y = F_2^{-1}(x)$$

- Unfortunately, the inversion of $P_2$ cannot always be carried out algebraically, but if this is possible, this is the method of choice

# Exact Inversion of Cumulative PDF (in simple terms)

- **Start with Samples**:
  - Begin with samples from a known distribution (e.g., uniform distribution)
- **Calculate CDF**:
  - Compute the CDF of the known distribution
- **Find Target CDF**:
  - Use the target distribution's CDF and find its inverse
- **Transform Samples**:
  - For each sample from the initial distribution, use the inverse CDF of the target distribution to get the corresponding sample

# Exact inversion of cumulative PDF, now graphically

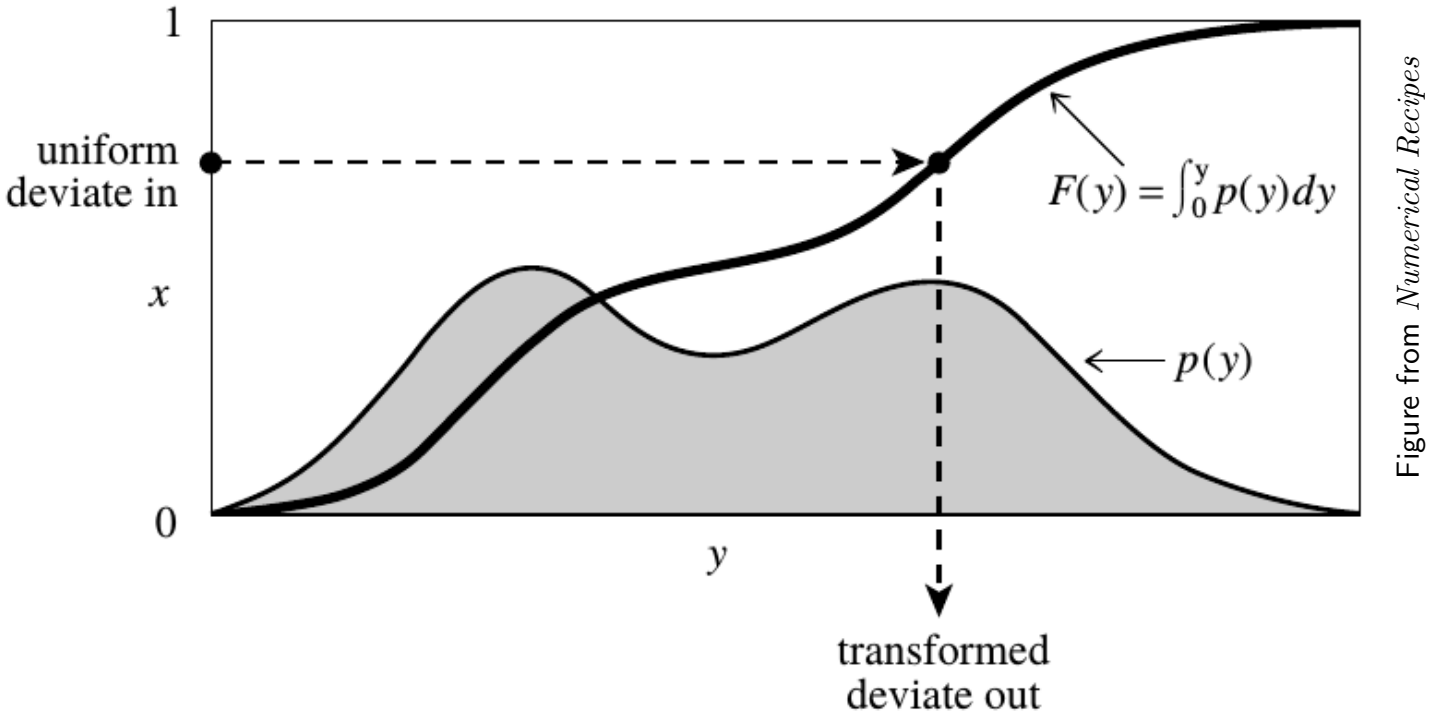



Figure from *Numerical Recipes*

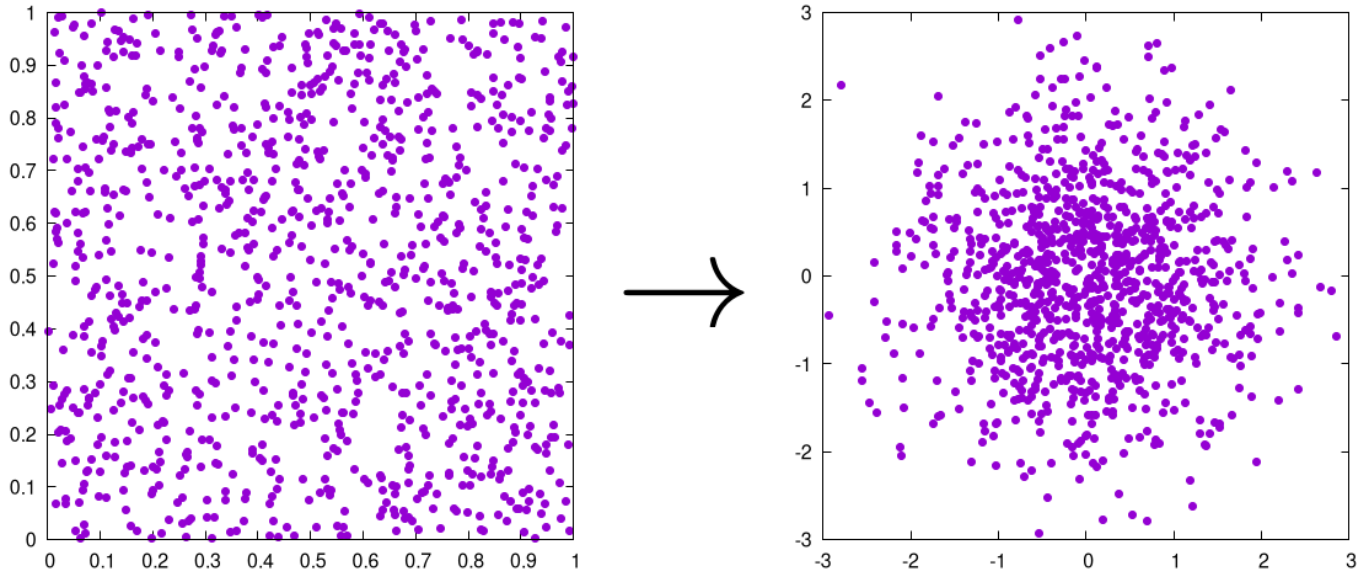
- We want: p(y)
  - calculate $F(y) = \int_0^y p(v)\mathrm{d}v$
  - uniformly sampled $x_i$ are transformed to

$$y_i = F^{-1}(x_i)$$

- Note, how gradient $\mathrm{d}F/\mathrm{d}x = p$ controls density of samples on $y$-axis

# Exact inversion of cumulative PDF

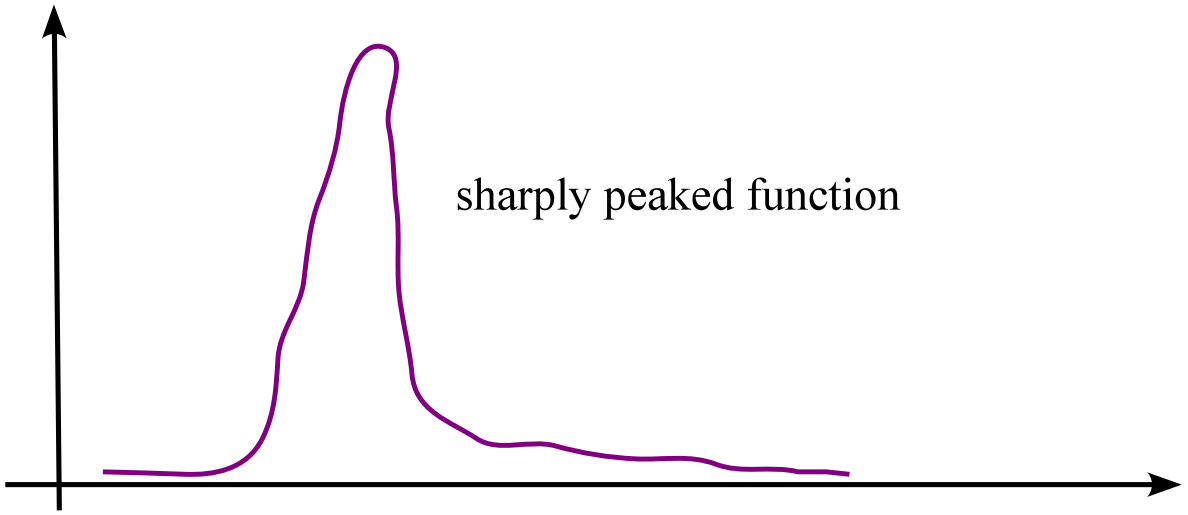
- Further tricks possible
  - Example in R: →`inversion_of_cdf_demo.ipynb`
  - classical example: usage of 2D polar coordinates in the so-called Box-Muller procedure for Gaussian samples




Box-Muller transformation →Blackboard

# Importance sampling

- Common problem of MC integration: The integrand is very small over most of the integration volume, e.g., if the integrand is sharply peaked




- Idea of *importance sampling*: choose random points preferentially around the peak, selecting fewer points where the integrand is small
  - should be more efficient and help to reduce the error for a given number of points

# Importance sampling

Assume we want to integrate (here in 1D for simplicity)

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

- Suppose we choose a *probability distribution* $p(x)$ that is:
  - Close to the function $f(x)$
  - Simple enough to generate $x$-values from this distribution
  - Has its domain of definition $[a, b]$

- We can change variables as $y = P(x)$ and $\mathrm{d}y = p(x)\mathrm{d}x$:

$$I = \int_{x=a}^b \frac{f(x)}{p(x)}\,p(x)\mathrm{d}x = \int_{y=0}^1 \frac{f(P^{-1}(y))}{p(P^{-1}(y))}\,\mathrm{d}y$$

# Importance sampling

- Approximating with Monte Carlo integration:

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

- Here, $y \sim U(0,1)$ as in standard MC integration. Note the change of the interval boundaries. The last equality comes from transforming between random samples so that $x \sim p(x)$, i.e., samples drawn from the distribution $p(x)$

  Because $f/p$ is flatter than $f$ if the shape of $p$ is similar to that of $f$, the variance of $f/p$ will be smaller than the variance of $f$, i.e. we obtain a smaller error for a given number of samples $N$

$$I_N = \frac{1}{N} \sum_{i} \frac{f(x_i)}{p(x_i)}.$$

# Importance sampling

- The error estimate from before carries over to the function $\frac{f}{p}$:

$$\sigma_{I_N} = \sqrt{\frac{\langle f^2/p^2 \rangle - \langle f/p \rangle^2}{N}}$$

  - reasoning as before: appeal to the Central Limit Theorem

- Importance sampling is an example from the family of methods related to the *reduction of variance*

# Importance sampling

- Ideal choice: $p(x) \propto f(x)$
  (in fact $p(x) \propto |f(x)|$) but let's assume $f$ is positive everywhere

What would be the integration error in this case?

# Importance sampling

- Ideal choice: $p(x) \propto f(x)$
  (in fact $p(x) \propto |f(x)|$) but let's assume $f$ is positive everywhere)

What would be the integration error in this case?

And why doesn't this work?

# Importance sampling

- Moral: pick a distribution $p(x)$ that you can normalize and sample from!

# Importance sampling and estimating moments of distributions

- Consider a sample $\{x_i\}$ of $N$ values drawn from a population with probability distribution $p(x)$. The expectation value $\mathrm{E}[x]$ of the distribution is usually estimated by the sample mean as

$$\mathrm{E}[x] \equiv \int_{-\infty}^{+\infty} x\, p(x) \mathrm{d}x \approx \frac{1}{N} \sum_{i=1}^{N} x_i$$

- Formula can be interpreted as a MC integration with importance sampling

- This carries over to higher moments

- MC error formula applies here for the estimation of the population moments.
  - The standard error of the mean, for instance, is given by: $\sigma_{\bar{x}} = \sigma/\sqrt{N}$, where $\sigma$ is the standard deviation of the population

# Example 2: Monte Carlo integration with importance sampling

Repeat Example 1 with importance sampling, using the sampling probability

$$p(x) = \frac{2}{3} x^{-1/3}$$

over the interval $0 < x \leq 1$.
Note that $p(x)$ captures part of the shape of $f(x)$, while being simple enough to allow a creation of properly sampled points by direct inversion
For the standard deviation of $g \equiv f/p$ one gets

$$\sigma_I = \sqrt{\frac{\langle g^2 \rangle - \langle g \rangle^2}{N}} \approx \frac{0.0448}{\sqrt{N}}.$$

This is nearly 20 times better than obtained with plain sampling in Exercise 1.

- Can you confirm this expectation numerically?
- Plot the function $g$!
- Does the integral come out correctly? ($\rightarrow$ `MCexample2.ipynb`)

# Rejection sampling (von Neumann 1951)

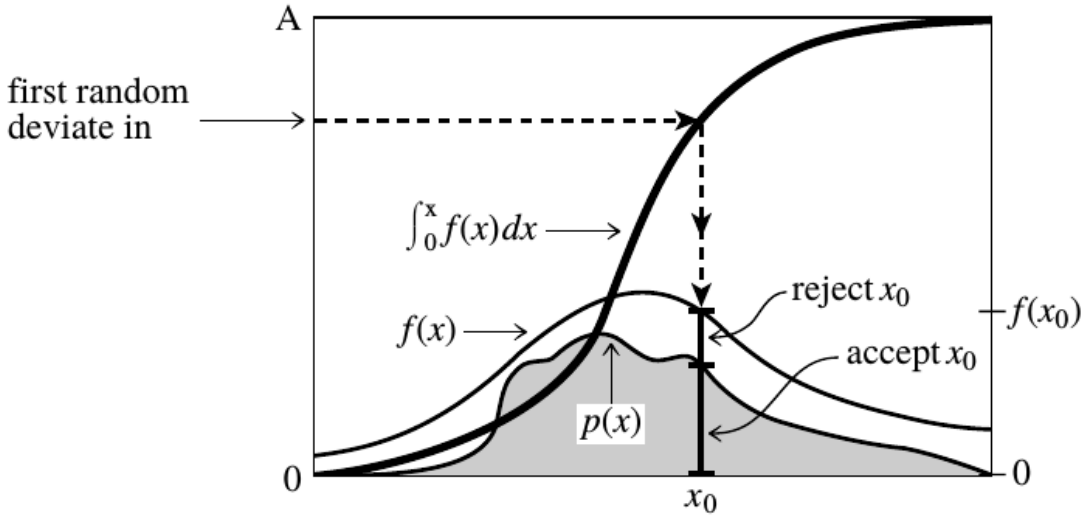



Figure from *Numerical Recipes*

$p(x)$ is the desired random number distribution, and $f(x)$ is a distribution that we can create. $f$ should obey

$$p(x) \leq f(x) \quad \forall x$$

Then the method proceeds as follows:

1. Generate trial random point $x_0$ drawn from distribution $f(x)$ (see inversion method)
2. Generate a sample $y$ from a uniform distribution with the bounds $0 \leq y < f(x_0)$
3. Acceptance step: If $y \leq p(x_0)$ return $x_0$ as a sample value
4. Otherwise, i.e. for $y > p(x_0)$ reject the trial value for $x_0$ and repeat from step 1

# Rejection sampling

- The basic idea is to define a simpler function $f(x)$ that we $can$ draw samples from, but we only accept a sample to our set with a certain probability $p(x)/f(x)$, otherwise we reject it

- Advantages: works in any dimension, $p(x)$ need not be normalized

- Disadvantages: potentially low efficiency if many samples are rejected (actually gets worse for higher dimensions of p(x))

# Sampling with a stochastic process

- Sometimes neither direct inversion nor the rejection method can be readily used to sample from a given distribution function $p(x)$
- Another idea: construct $p(x)$ through a $stochastic\ process$ that has $p(x)$ as its equilibrium distribution
- This involves a random walk in parameter space, tailored so that the probability to be at $x$ is proportional to $p(x)$ (so that regions of higher probability density are preferentially explored)
- This is known as a $Markov\ process$, generating a **Markov Chain**
- A Markov Chain is a discrete sequence of states

$$x_1 \xrightarrow{f} x_2 \xrightarrow{f} x_3 \xrightarrow{f} \ldots \xrightarrow{f} x_n$$

- The function $f$ is a random update operator
  - e.g., in 50 % of the updates,the process moves left, otherwise it moves right

# Sampling with a stochastic process

- Characterizing property of a Markov process: The defining property of a Markov process is that the transition probability $W_f$ from one state to the next state in the chain

$$W_f(x \to x') = W_f(x'|x),$$

depends *only* on the current state

  - the history of the process does not influence the transition probabilities
  - this makes Markov chains particularly simple and efficient
  - the function $f$ can mediate small or large updates

- The transition probability must satisfy

$$\int W_f(x \to x')\,\mathrm{d}x' = 1 \quad \text{and} \quad W_f(x \to x') \geq 0$$

  - this ensures the "walking entity" does not disappear, maintaining the conservation of probability

# Sampling with a stochastic process

- Applying the transition probability to an entire probability distribution (or ensemble of walkers), the new probability distribution after one transition is:

$$p(x) \xrightarrow{f} p'(x') = \int p(x) W_f(x \to x') \, \mathrm{d}x.$$

- Two essential properties make the Markov process powerful:

  1. $f$ must preserve $p(x)$ as an equilibrium distribution, meaning $p(x)$ is a fixed point of $f$. This requires

$$p(x') = \int p(x) W_f(x \to x') \, \mathrm{d}x.$$

  2. Starting from any state $x$, *repeated applications* of $f$ must be able to approach any other state $x'$. This is known as the *ergodic property*

# Sampling with a stochastic process

- Two important results follow from ergodicity and existence of an equilibrium distribution:
  - Any ensemble of states approaches the equilibrium distribution if $f$ is applied sufficiently often
  - The collection of states in a single Markov chain under the action of $f$ approaches $p(x)$ as the number of steps goes to infinity

- Obvious question: how often is "sufficiently often"?
  - In practice, this depends on the specific problem and the properties of $f$
  - Techniques such as monitoring the convergence of sample statistics and diagnostic tools like trace plots can be helpful

# Condition of detailed balance

- Almost all of the commonly used update steps follow the *detailed balance condition* at equilibrium

$$p(x) \cdot W_f(x \to x') = p(x') \cdot W_f(x' \to x)$$

What does this mean?

- Detailed balance ensures that the probability flow from $x$ to $x'$ is balanced by the flow from $x'$ to $x$, maintaining equilibrium

- If detailed balance holds, it is easy to show that $p(x)$ is a fixed point under $f$. For other choices of $f$, this might still be true but could be more challenging to prove

- Detailed balance and ergodicity are already sufficient conditions for a Markov chain to sample from $p_{\text{eq}}(x)$. However, we need to find a concrete realization of $W_f$.

- Key question now:
  How do we set the transition probability $W_f$ to fulfil the detailed balance condition?

# The Metropolis-Hastings algorithm

- The Metropolis-Hastings algorithm, developed by N. Metropolis (1953) and updated by W.K. Hastings (1970), provides a generic way to construct a suitable transition operation for sampling from complex probability distributions

- Steps of the Metropolis-Hastings algorithm:

1. **Propose new state**: When the current state is $x$, propose a new state $x'$ with a proposal probability $q(x \to x')$
2. **Calculate the Hasting's ratio** (the $acceptance\ probability$):

$$r = \min\left(1, \frac{p(x')\, q(x' \to x)}{p(x)\, q(x \to x')}\right),$$

   where the $\min$-operation is used to restrict the value of $r$ to the range $[0, 1]$
3. **Accept or Reject the proposed state:** draw a random number $u \in [0, 1)$
   - If $u < r$, then $x_{n+1} = x'$ and $x'$ is $accepted$ as a new element of the Markov chain
   - If $u \geq r$, then $x_{n+1} = x$ and $x'$ is $rejected$ as a new element of the Markov chain

# Does this procedure fulfil the condition of detailed balance?

- To verify, work out transition probability $W(x \to x')$ which is the product of the proposal probability of the new state and acceptance probability:

$$W(x \to x') = q(x \to x') \frac{p(x')q(x' \to x)}{p(x)q(x \to x')} = \frac{p(x')}{p(x)} q(x' \to x)$$

  - assuming here without loss of generality that the Hastings ratio is $\leq 1$

- In this case, the inverse transition probability is given as (Hastings ratio is the inverse of the one above but limited to be $\leq 1$)

$$W(x' \to x) = q(x' \to x) \cdot 1$$

- Combining both equations above give the condition of detailed balance

$$p(x) \cdot W(x \to x') = p(x') \cdot W(x' \to x)$$

- If the Hastings ratio is in fact $> 1$ one interchanges $x$ and $x'$ in the reasoning

# The Metropolis-Hastings algorithm

- The proposal probability $q(x \to x')$ can be fairly arbitrary – it only needs to be $ergodic$.
  - **Ergodic:** All states must be reachable through successive applications of $q$. This ensures that the Markov Chain can explore the entire state space given enough time

- The Markov Chain Monte Carlo (MCMC) created by the algorithm will eventually produce a fair sample of the target distribution $p(x)$
  - As the number of iterations goes to infinity, the distribution of the states in the Markov Chain approaches the target distribution $p(x)$

- This result is remarkable and worth contemplating. For further details, see lecture notes by Volker Springel or $Numerical\ Recipes$

# The Metropolis-Hastings algorithm

- Unlike independent samples from a distribution, the sequence of states in a Markov Chain is *correlated*
  - This means that each state depends on the previous state, introducing correlations between successive samples
- Consequently, traditional error estimates for MC integration do not apply to MCMC
  - alternative methods are used that compute, for example, the covariance matrix and the autocorrelation function
- Nevertheless, Markov Chains are the workhorses of many Monte Carlo codes today due to their flexibility and ability to sample from complex distributions
  - They are extensively used in statistical physics, Bayesian statistics, and machine learning

# Metropolis update

- This is the special case known as the **Metropolis algorithm**, in which the stochastic proposal operator is symmetric, i.e. $q(x \to x') = q(x' \to x)$

- The acceptance probability simplifies to the Metropolis ratio:

$$r = \min\left(1, \frac{p(x')}{p(x)}\right) \tag{1}$$

  - A proposed move to a state of higher probability is always accepted
  - Sometimes, a move to a proposed state of lower probability is also accepted

- The simplest form of such a symmetric update would be:

$$q(x \to x') : \quad x' = x + e,$$

  where $e$ is distributed symmetrically around zero and is independent of $x$

- For example, $e$ could be drawn from a normal or uniform distribution of some prescribed width

# Example: Samples from a bivariate normal distribution generated with the Metropolis-Hastings algorithm

Generate samples of a PDF with the Metropolis-Hastings algorithm. Use

- 2D Gaussian distribution, zero mean, standard deviation 1, covariance 0.6
- Update $q(x \to x')$ with uniform distribution with variable "reach"
- Start at unlikely location
- Monitor the acceptance rate of the MH update step

Hint: Use the partial R code `MCexample3_fragment.ipynb` as a starting point. You need to provide the Metropolis-Hastings part yourselves.

- What do you conclude about the step size in the random walk?
- What is your assessment of the initial samples in the beginning of the chain?
- What is the correlation among your samples?

# Example: Samples from a bivariate normal distribution generated with the Metropolis-Hastings algorithm