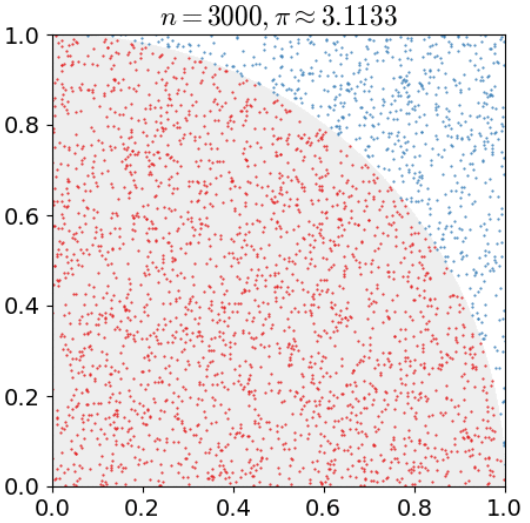
Monte Carlo Methods in Python

Many computer calculations employ Monte Carlo methods that include elements of chance to either:

1) Simulate random physical processes, such as thermal motion or radioactive decay, or
2) Estimate areas, volumes etc. by numerical integration

Monte Carlo Casino, Monaco, Cote-d-Azur, France



Monte Carlo Methods in Python

Stanislaw Ulam

- Polish-American Mathematician and Physicist
- 1909 (Lviv, modern-day Ukraine) – 1984 (Santa Fe, NM, USA)
- Went to the US as refugee in 1939
- Joined the Manhattan Project in Los Alamos while shrouded in secrecy
- Taught at U. Wisc. Mad., U Colorado, U Florida, Rockefeller U, USC (LA), U Calif. (Davis) etc.
- Came up with Monte Carlo while playing solitaire in hospital
- Name coined due to an uncle who frequently gambled at the casino

Monte Carlo Casino, Monaco, Cote-d-Azur, France

Monte Carlo Methods in Python

Randomness in deterministic computers???

Besides, how do even know if anything can be ‘random’??

Monte Carlo Methods in Python

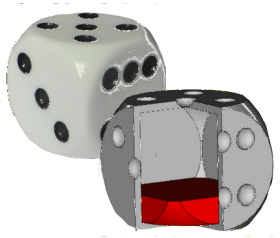
What is ‘randomness’, or a ‘random variable’?

Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers , based on a probability of choice.

- Uniform randomness: When all numbers in the subset are equally likely to be chosen
- Non-uniform Randomness: When the probability of a particular choice depends on the value of the choice.

P(r) dr – Probability of a choice being in [r, r+dr]

Uniform: P(r) constant



Non- Uniform: P(r) depends on r

Monte Carlo Methods in Python

What is ‘randomness’, or a ‘random variable’?

Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers, based on a probability of choice.

- Mathematically, “independent randomness” occurs when there are no correlations between any two sets of choices from the same subset

$$\langle r \rangle \equiv \int_D r\, P(r)\, \mathrm{d}r \text{ , with } \int_D P(r)\, \mathrm{d}r = 1$$

Independently Random:

$$\langle r_1\, r_2 \rangle = \langle r_1 \rangle\, \langle r_2 \rangle \text{ or } P(r_1|r_2) = P(r_1)P(r_2)$$

$$C(r_1, r_2) \equiv \langle r_1\, r_2 \rangle - \langle r_1 \rangle\, \langle r_2 \rangle = 0$$

Monte Carlo Methods in Python

What is ‘randomness’, or a ‘random variable’?

Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers, based on a probability of choice.

- Mathematically, “independent randomness” occurs when there are no correlations between any two sets of choices from the same subset
- Obeys the ‘**Central Limit Theorem’**

Consider an arbitrarily large (infinite) array of independent random variables, called ‘A’.

By the law of large numbers, their average is a fixed constant μ and stdev σ

For a finite (but reasonably large) array of n independent random variables from A , the probability that their sample average (times $\sqrt{n}$) is a value ‘x’ (times $\sqrt{n}$) is

$$P(x) = \frac{1}{\sigma\sqrt{\pi}}\, e^{-(x-\mu)^2/2\sigma^2}$$

Monte Carlo Methods in Python

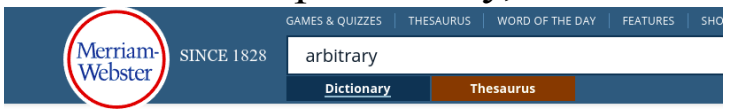
What is ‘randomness’, or a ‘random variable’?

Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers, based on a probability of choice.

- **<u>Note: In science, ‘Random’ and ‘Arbitrary’ are two different things!</u>**

Although they are used interchangeably in common parlance

Arbitrary choices have no well-defined probability, whereas random choices do



arbitrary adjective

Save Word

ar·bi·trary | \ ˈär-bə-ˌtrer-ē, -ˌtre-rē \

Definition of *arbitrary*

1 a : existing or coming about seemingly at random or by chance or as a capricious and unreasonable act of will

// an *arbitrary* choice

// When a task is not seen in a meaningful context it is experienced as being *arbitrary*.

— Nehemiah Jordan

Monte Carlo Methods in Python

What is ‘randomness’, or a ‘random variable’?

Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers, based on a probability of choice.

- **<u>Note: In science, ‘Random’ and ‘Arbitrary’ are two different things!</u>**

  Although they are used interchangeably in common parlance

  Arbitrary choices have no well-defined probability, whereas random choices do

Monte Carlo Methods in Python

What is ‘randomness’, or a ‘random variable’?

Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers, based on a probability of choice.

- **<u>Note: In science, ‘Random’ and ‘Arbitrary’ are two different things!</u>**

  Although they are used interchangeably in common parlance

  Arbitrary choices have no well-defined probability, whereas random choices do

- However, in some cases, depending on context, we have ignored this distinction

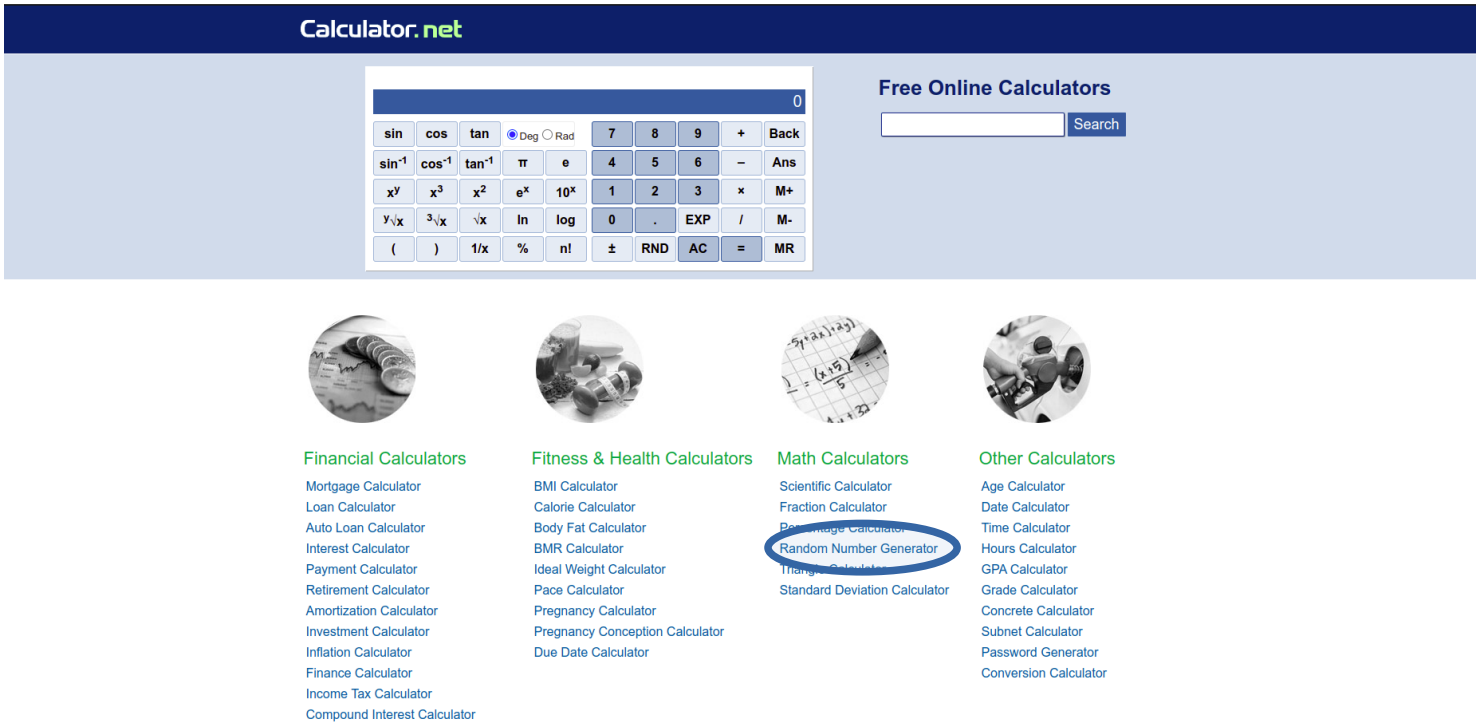
  In Python: “Prove” that a Hermitian matrix always has real eigenvalues:

  1. Build a fairly large random numpy matrix A, do the operation $A \leftarrow A + A.T.conj()$
  2. Get the eigenvalues numerically (usually a matrix tridiagonalization, followed by QR factorizations)
  3. Check to see if they’re all real up to the propagated numerical errors

Monte Carlo Methods in Python

What is ‘randomness’, or a ‘random variable’?

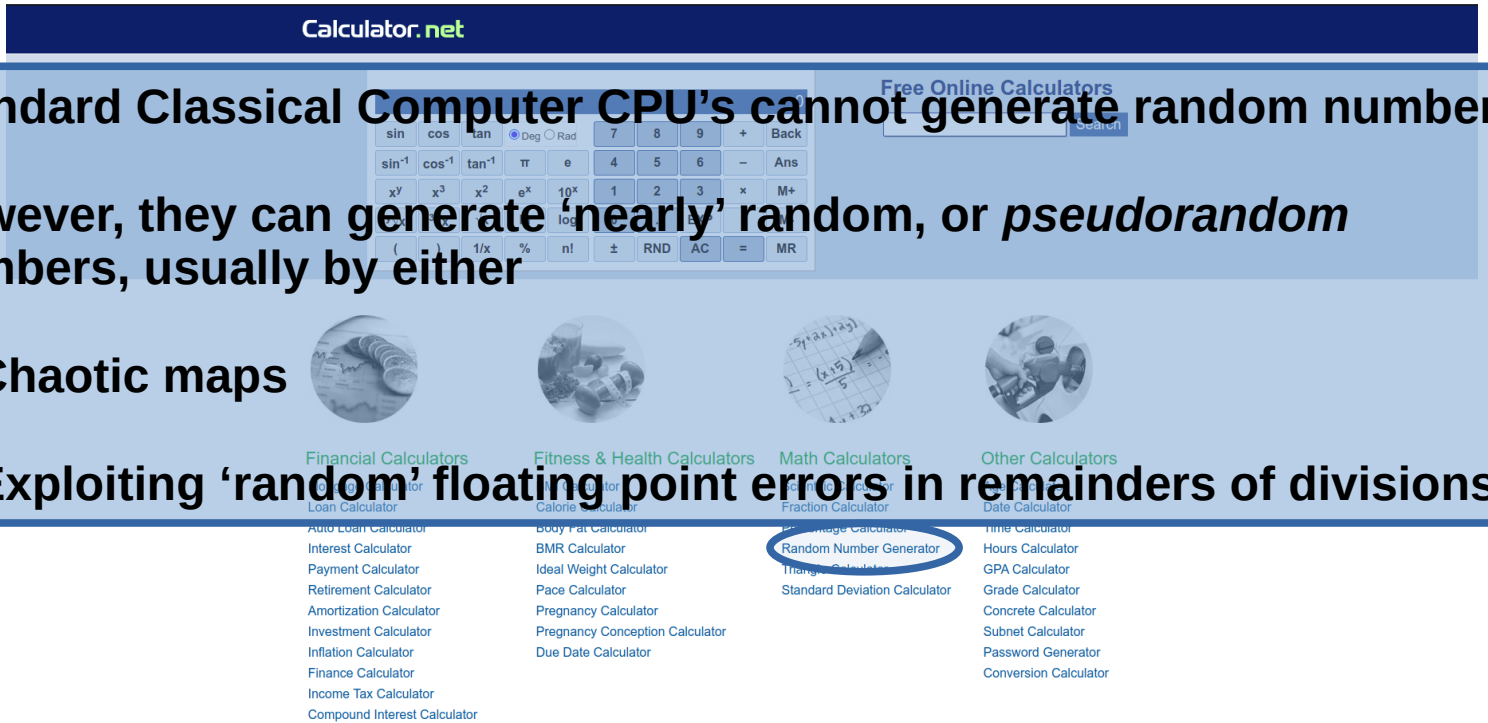
Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers, based on a probability of choice.



Monte Carlo Methods in Python

What is ‘randomness’, or a ‘random variable’?

Basically, it’s a special way of choosing the elements of a set. In our case, it involves choosing a single number from a particular subset of complex numbers, based on a probability of choice.



Standard Classical Computer CPU’s cannot generate random numbers

However, they can generate ‘nearly’ random, or *pseudorandom* numbers, usually by either

1) Chaotic maps

2) Exploiting ‘random’ floating point errors in remainders of divisions

Monte Carlo Methods in Python

Random numbers in NumPy: Uniform beteween 0 and 1 by default

```
Python 3.8.8 (default, Apr 13 2021, 19:58:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> print(np.random.random())
0.6226775696151192
>>> print(np.random.random())
0.792263425922030 5
>>> print(np.random.random())
0.14630228848931848
>>> a = np.random.random((3,2)); print(a)
[[0.69190953 0.36821051]
 [0.36047735 0.17258306]
 [0.00406954 0.26549197]]
```

Monte Carlo Methods in Python

Random numbers in NumPy: Uniform beteween 0 and 1 by default

How to make the probability of a number r be biased by P(r)?

```
import numpy as np

def weighted_choice(objects, weights):
    """ returns randomly an element from the sequence of 'objects',
        the likelihood of the objects is weighted according
        to the sequence of 'weights', i.e. percentages."""

    weights = np.array(weights, dtype=np.float64)
    # Normalize the weights:
    weights /= weights.sum()
    # "Integrate" the probabilities to find the cumulative probability
    weights = weights.cumsum()
    r = np.random.random()
    for i in range(len(weights)):
        if r < weights[i]:
            return objects[i]
```

$$\lambda(x_i) = \int_0^{x_i} P(r)\,\mathrm{d}r$$

Cumulative "Weights"

$$x_i = \{x_0, x_1, x_2, \ldots x_n\}$$

Objects to choose from

Choose r uniform-randomly

If $r < \lambda(x_i)$, then

return x_i

Monte Carlo Methods in Python

Random numbers in NumPy: Uniform beteween 0 and 1 by default

How to make the probability of a number r be biased by P(r)?

```
import numpy as np

def weighted_choice(objects, weights):
    """ returns randomly an element from the sequence of 'objects',
        the likelihood of the objects is weighted according
        to the sequence of 'weights', i.e. percentages."""

    weights = np.array(weights, dtype=np.float64)
    # Normalize the weights:
    weights /= weights.sum()
    # “Integrate” the probabilities to find the cumulative probability
    weights = weights.cumsum()
    r = np.random.random()
    for i in range(len(weights)):
        if r < weights[i]:
            return objects[i]
```

Note that:

$$\lambda(x_i) = \int_0^{x_i} P(r)\,\mathrm{d}r$$

$$\Longrightarrow \quad P(r) = \frac{d}{dr}\lambda(r)$$

Monte Carlo Methods in Python

https://bit.ly/monte_carlo_lab

Random numbers in NumPy: Uniform beteween 0 and 1 by default

How to make the probability of a number r be biased by P(r)?

$$\lambda(x_i) = \int_0^{x_i} P(r)\,\mathrm{d}r$$

$$\Longrightarrow\ P(r) = \frac{d}{dr}\lambda(r)$$

Example problems: See Jupyter Notebook

- **Radioactive Decay (exact)**
- **Random Walks**
- **Metropolis Algorithm and the Ising Model (1d and 2d)**

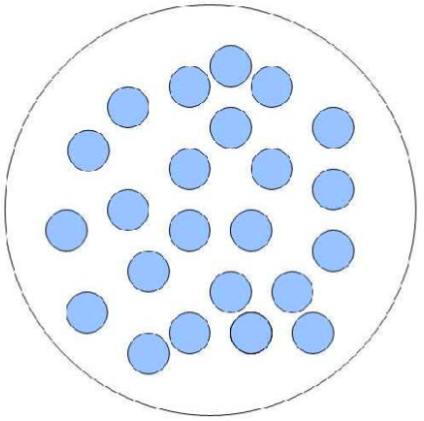
Monte Carlo Methods in Python

Example problems: See Jupyter Notebook

https://bit.ly/monte_carlo_lab

Radioactive Decay:

- **A sample of N radioactive atoms undergoing spontaneous decay**
- **The probability P of decay within a given time interval is proportional to the interval**



$$N(t) \xrightarrow{\Delta t} N - |\Delta N(t)|$$

$$P = -\lambda \Delta t = \frac{\Delta N}{N}$$

$$\Delta N \to dN \implies \frac{dN}{dt} = -\lambda N \implies N = N_0 e^{-\lambda t}$$

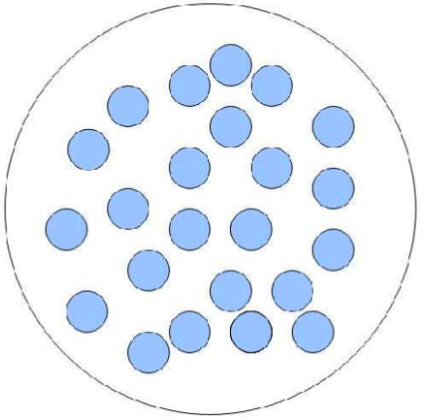
Monte Carlo Methods in Python

Example problems: See Jupyter Notebook

https://bit.ly/monte_carlo_lab

Radioactive Decay:

- **A sample of N radioactive atoms undergoing spontaneous decay**
- **The probability P of decay within a given time interval is proportional to the interval**



$$N(t) \xrightarrow{\Delta t} N - |\Delta N(t)|$$

$$P = -\lambda \Delta t = \frac{\Delta N}{N}$$

$$\Delta N \to dN \xrightarrow[\text{Continuum Approximation}]{} \frac{dN}{dt} = -\lambda N \Rightarrow N = N_0 e^{-\lambda t}$$

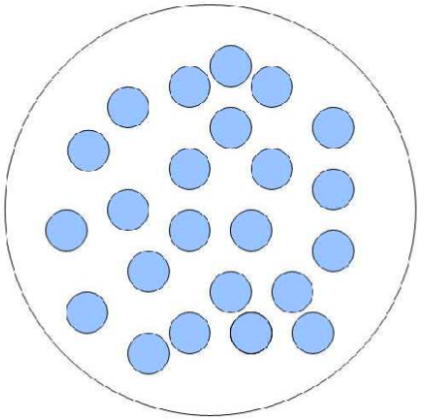
Monte Carlo Methods in Python

Example problems: See Jupyter Notebook

https://bit.ly/monte_carlo_lab

Radioactive Decay:

- **A sample of N radioactive atoms undergoing spontaneous decay**
- **The probability P of decay within a given time interval is proportional to the interval**



$$N(t) \xrightarrow{\Delta t} N + \Delta N(t)$$

$$P = -\lambda \Delta t = \frac{\Delta N}{N}$$

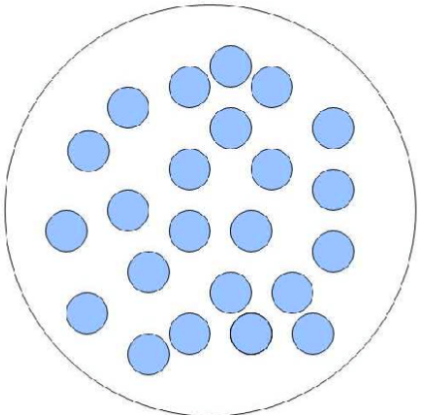
- **Exact Simulation: At each time, for each atom, choose a random number 'r' and decrement N only if**

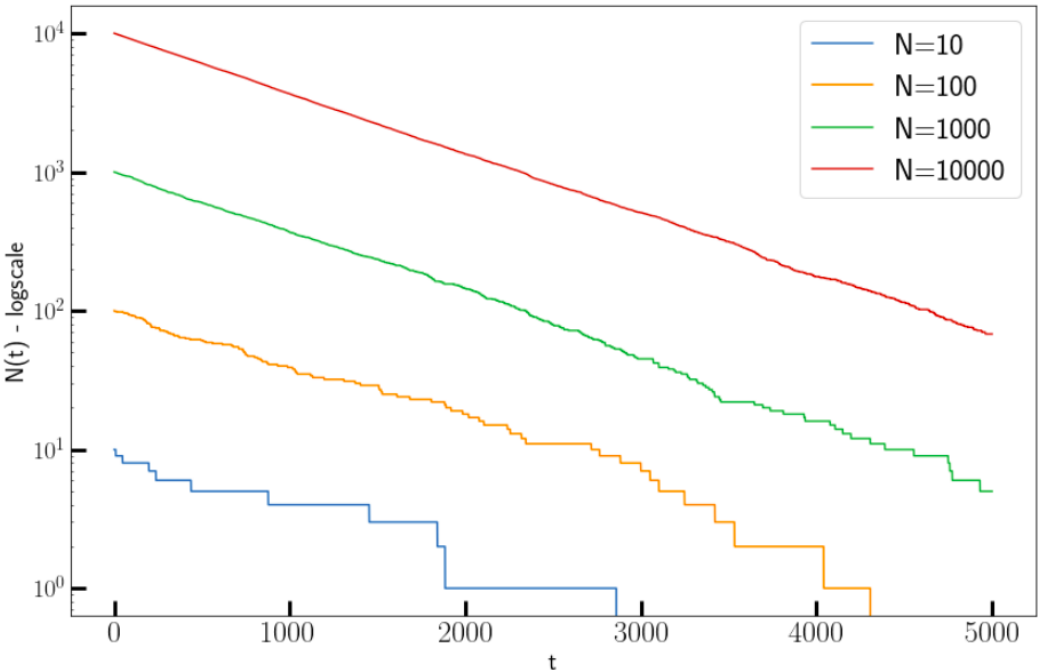
$$r < \lambda$$

Monte Carlo Methods in Python

Adaptable to solving other ODE's

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12,8)
plt.rcParams['font.size'] = 20

# Decay constant
lambda1 = 1e-3

natoms_arr = [10, 100, 1000, 10000]
maxtime = 5000

timeline = np.arange(0, maxtime  + 1)
for natoms in natoms_arr:
    natoms_begin = natoms
    natoms_time = []
    for time in timeline:
        decays = np.random.random(natoms)
        natoms -= np.count_nonzero(decays < lambda1)
        natoms_time.append(natoms)

    plt.plot(timeline,
natoms_time,label=f'N={natoms_begin}')

plt.ylabel("N(t) - logscale")
plt.xlabel("t")
plt.yscale("log")
plt.legend()
plt.show()
```





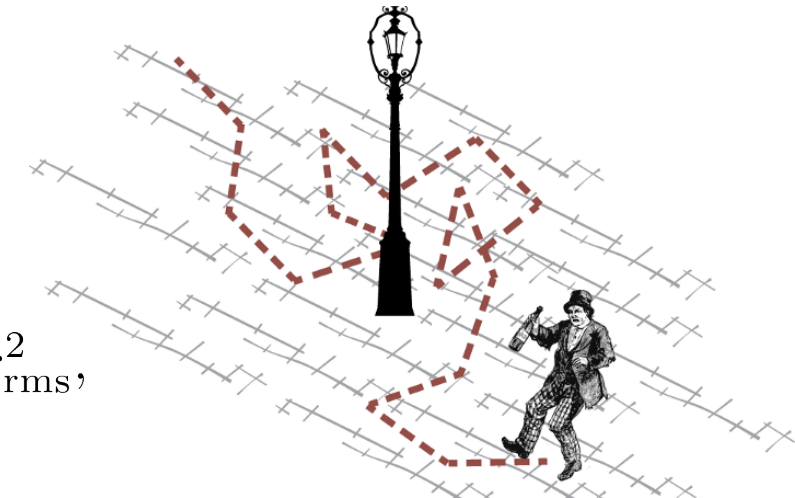
Monte Carlo Methods in Python

Example problems: See Jupyter Notebook

https://bit.ly/monte_carlo_lab

Random Walk Problem:

- **Monte Carlo-like model of Diffusion, Brownian motion etc.**
- **A digital simulation of a 'drunk man', where a point advances along a graph in discrete steps**
- **Each step can be of a random stepsize in a random direction, or fixed stepsize in a random direction.**



$$R^2 = (\Delta x_1 + \Delta x_2 + \cdots + \Delta x_N)^2 + (\Delta y_1 + \Delta y_2 + \cdots + \Delta y_N)^2$$

Average over a large number of trials:

$$R^2_{\mathrm{rms}} \simeq \left(\Delta x_1^2 + \Delta y_1^2\right)_{\mathrm{rms}} + \left(\Delta x_2^2 + \Delta y_2^2\right)_{\mathrm{rms}} + \cdots = N r^2_{\mathrm{rms}},$$

$$R_{\mathrm{rms}} \simeq \sqrt{N} r_{rms},$$

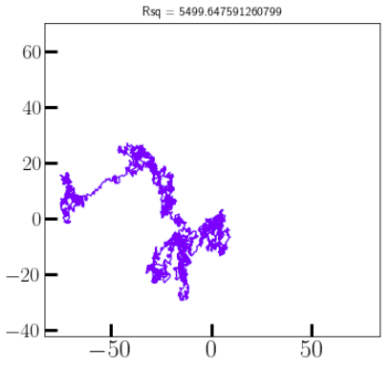
Monte Carlo Methods in Python

Example problems: See Jupyter Notebook

https://bit.ly/monte_carlo_lab

Random Walk Problem: Given the mean free path of air at room temperature (68 nm), how many collisions will it take for a single molecule suspended in air to travel 20 metres?

Molecule = Random Walker

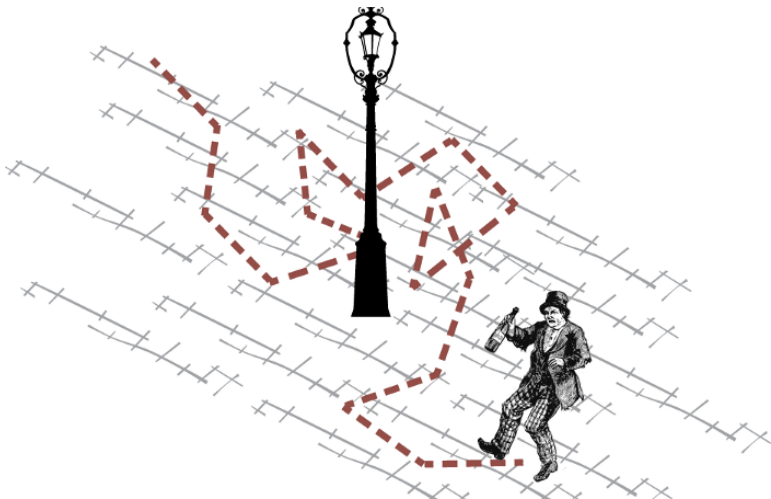


$$R^2 = (\Delta x_1 + \Delta x_2 + \cdots + \Delta x_N)^2 + (\Delta y_1 + \Delta y_2 + \cdots + \Delta y_N)^2$$

Average over a large number of trials:

$$R_{rms}^2 \simeq \langle \Delta x_1^2 + \Delta y_1^2 \rangle + \langle \Delta x_2^2 + \Delta y_2^2 \rangle + \cdots = N\langle r^2 \rangle = N r_{rms}^2,$$

$$R_{rms} \simeq \sqrt{N} r_{rms},$$



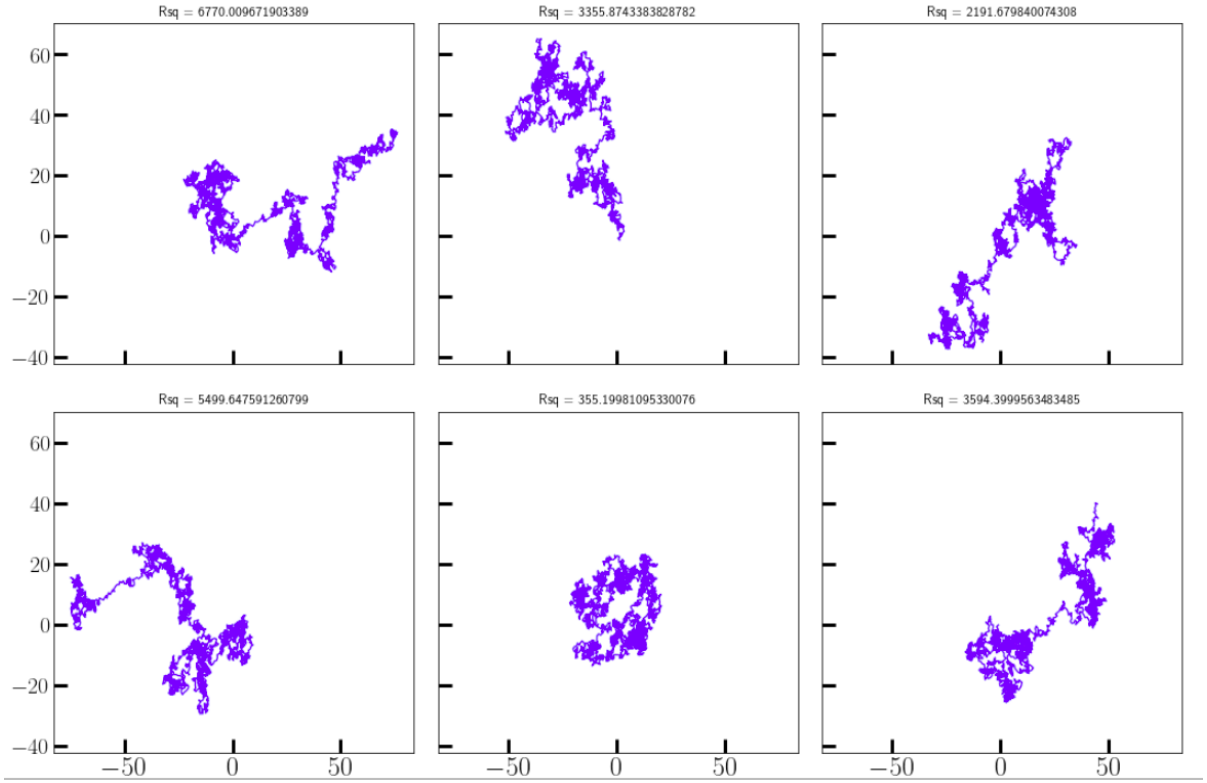
Monte Carlo Methods in Python

Example problems: See Jupyter Notebook

https://bit.ly/monte_carlo_lab

Random Walk Problem: Given the mean free path of air at room temperature (68 nm), how many collisions will it take for a single molecule suspended in air to travel 20 metres?

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (15,10)
plt.rcParams['font.size'] = 20

fig, axs = plt.subplots(2,3, sharex=True, sharey=True)
axs = axs.flatten()

jmax = 2500
for ax in axs:
    dr_vals = 2.0 * np.random.random((jmax,2)) -1
    Lvals = np.sum(dr_vals**2, axis=1)
    dr_vals /= np.sqrt(Lvals[:, None])

    points = np.cumsum(dr_vals, axis=0)
    Rsq = np.linalg.norm(points[-1])**2
    ax.set_title(f"Rsq = {Rsq}", fontsize=12)
    ax.plot(points[:,0], points[:,1], color='blue', alpha=0.6)
plt.show()
```



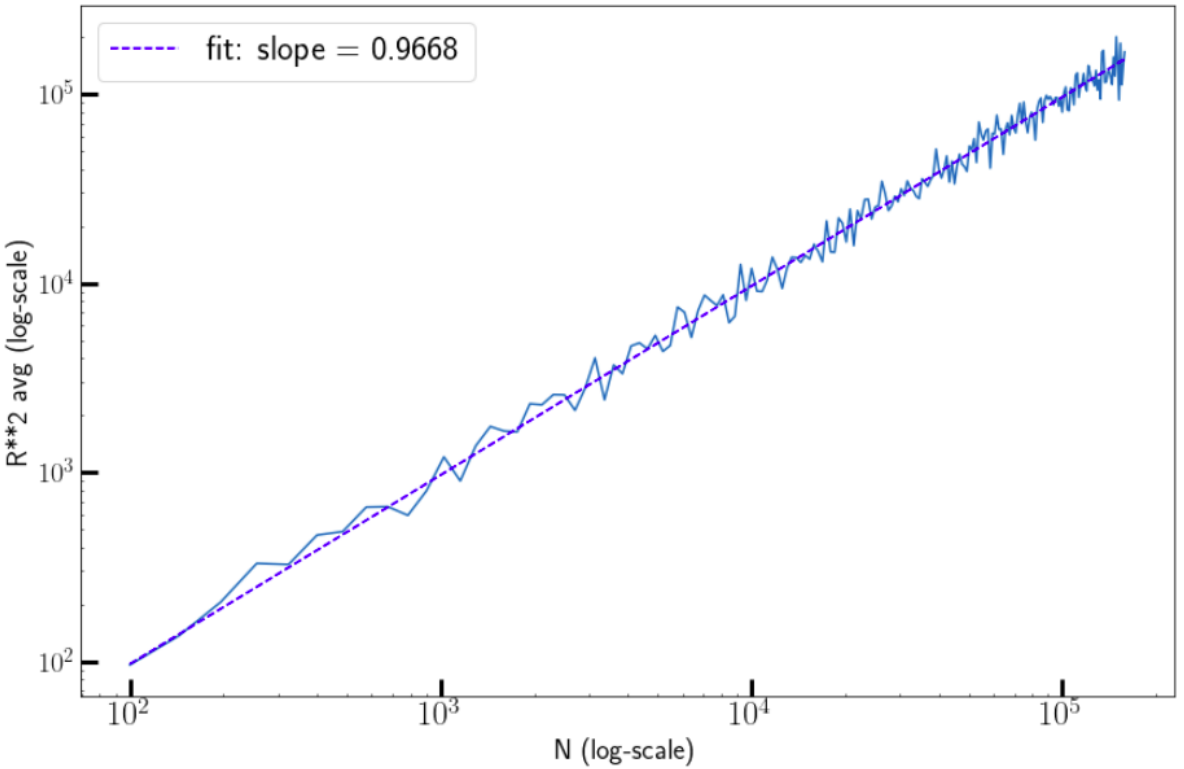
Monte Carlo Methods in Python

Example problems: See Jupyter Notebook

https://bit.ly/monte_carlo_lab

Random Walk Problem: Given the mean free path of air at room temperature (68 nm), how many collisions will it take for a single molecule suspended in air to travel 20 metres?

```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy as np
plt.rcParams['figure.figsize'] = (12,8)
plt.rcParams['font.size'] = 20

def randwalk2d(nsteps, repeat=6):
    points = []
    for i in range(repeat):
        dr_vals = 2.0 * np.random.random((nsteps,2)) -1
        Lvals = np.sum(dr_vals**2, axis=1)
        dr_vals /= np.sqrt(Lvals[:, None])
        points.append(np.cumsum(dr_vals, axis=0))
    return points

nsteps = np.arange(10,400,2)**2
Rsq_avs = []
for nstp in nsteps:
    walks = randwalk2d(nstp, repeat=50)
    Rsq_avs.append(np.average([np.linalg.norm(w[-1])**2 for w in walks]))
plt.loglog(nsteps, Rsq_avs)
popt, pcov = curve_fit(lambda x,a: a * x, nsteps, Rsq_avs)
plt.loglog(nsteps, popt[0] * nsteps, 'b--', label=f'fit: slope = {popt[0]:2.4f}')
plt.xlabel("N (log-scale)")
plt.ylabel("R**2 avg (log-scale)")
plt.legend(); plt.show()
```



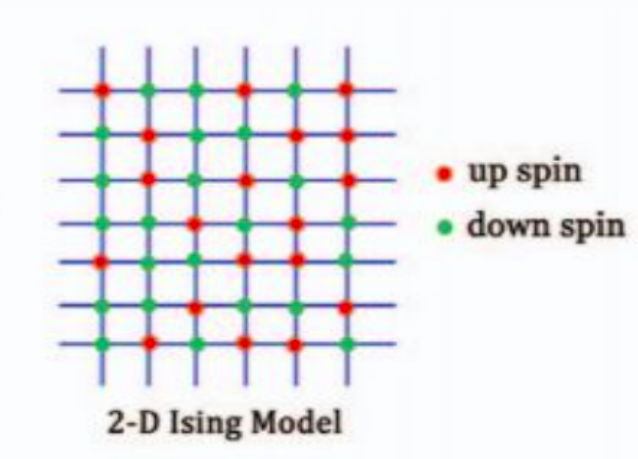
Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

Cannot get the exact partition function numerically
Too many combinations of spins for a large size

$$N \to 2^N$$



Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

The **Metropolis Algorithm** is a technique for computing the Monte Carlo calculation of averages that accurately simulates the fluctuations occurring during thermal equilibrium.

It changes individual spins randomly, but weights the changes such that on the average a Boltzmann distribution results (Canonical Ensemble)

$$\mathcal{P}(E_{\alpha_j}, T) = \frac{e^{-E_{\alpha_j}/k_B T}}{Z(T)}$$

Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

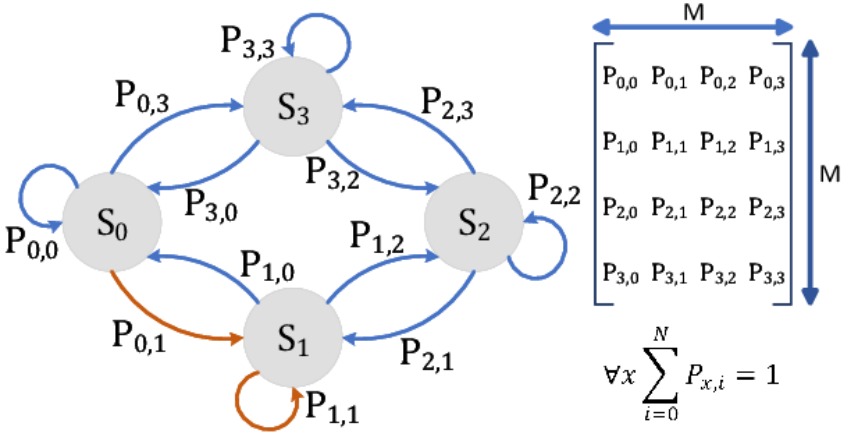
1. Start with an arbitrary spin configuration $\alpha_k = \{s_1, s_2, \ldots, s_N\}$.
2. Generate a *trial* configuration α_{k+1}:
   a. Pick a particle i randomly and flip its spin.
   b. Calculate the energy $E_{\alpha_{\text{tr}}}$ of the trial configuration.
   c. If $E_{\alpha_{\text{tr}}} \leq E_{\alpha_k}$, accept the trial by setting $\alpha_{k+1} = \alpha_{\text{tr}}$.
   d. If $E_{\alpha_{\text{tr}}} > E_{\alpha_k}$, accept with relative probability $\mathcal{R} = \exp(-\Delta E/k_B T)$:
      a. Choose a uniform random number $0 \leq r_i \leq 1$.
      b. Set $\alpha_{k+1} = \begin{cases} \alpha_{\text{tr}}, & \text{if } \mathcal{R} \geq r_j \quad \text{(accept)}, \\ \alpha_k, & \text{if } \mathcal{R} < r_j \quad \text{(reject)}. \end{cases}$

Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

Will this equilibriate into a Canonical Ensemble?



Markov Chain:
Each AB transition depends only on

$$P_{AB}$$

Detailed Balance:
Happens when all transition probs balance out

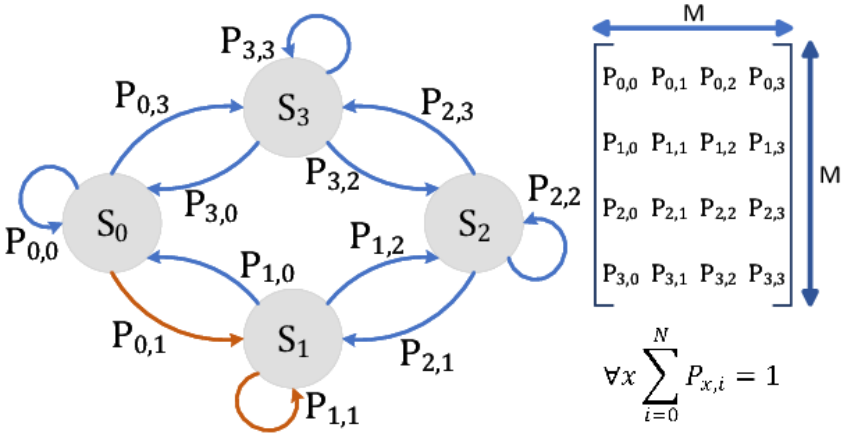
$$p_{A\to B} = p_A P_{AB} = p_{B\to A} = p_B P_{BA},$$

Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

Will this equilibriate into a Canonical Ensemble?



Detailed Balance:
Happens when all transition probs balance out

$$p_{A\to B} = p_A P_{AB} = p_{B\to A} = p_B P_{BA},$$

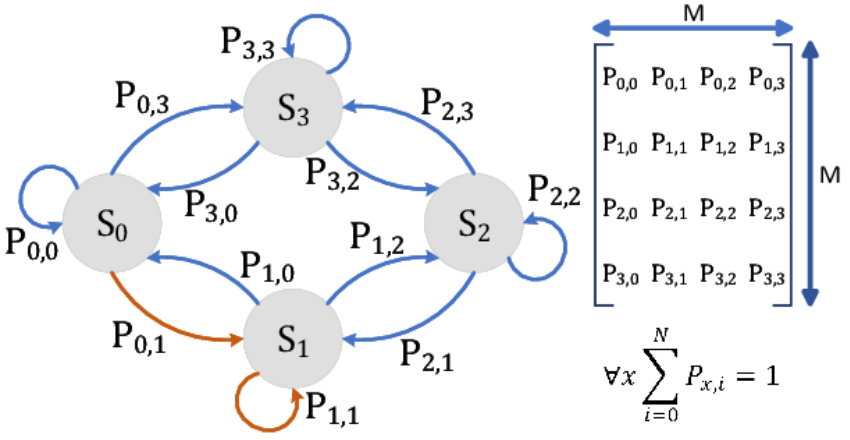
$$\frac{p_A}{p_B} = \frac{\pi_{BA}}{\pi_{AB}} = \frac{p_{BA}^{acc}}{p_{AB}^{acc}}.$$

Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

Will this equilibriate into a Canonical Ensemble?



Detailed Balance:
Happens when all transition probs balance out

$$\frac{p_A}{p_B} = \frac{\pi_{BA}}{\pi_{AB}} = \frac{p_{BA}^{acc}}{p_{AB}^{acc}}.$$

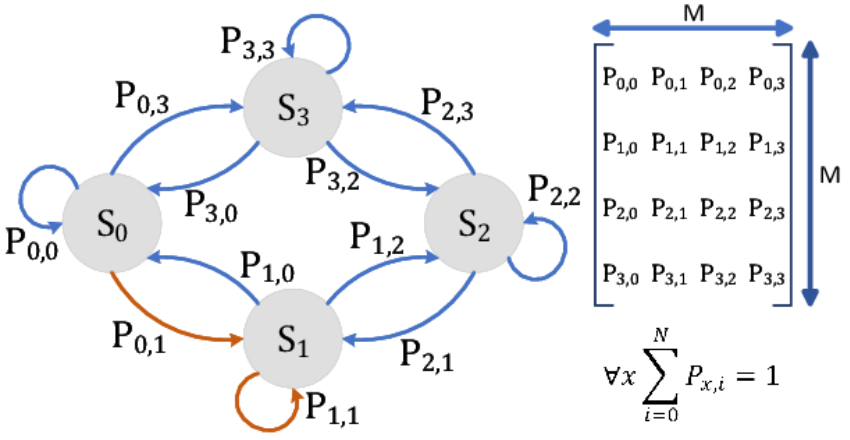
$$p_{AB}^{acc} \sim e^{(E_A - E_B)/k_B T}, \; p_{BA}^{acc} = 1$$

Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

Will this equilibriate into a Canonical Ensemble?



Detailed Balance:
Happens when all transition probs balance out

$$p_r \sim \exp\left\{-\frac{E_r}{k_B T}\right\}$$

Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

$$H = \begin{cases} -\sum_i s_i s_{i+1} - h\sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j}\left(s_{i,j+1} + s_{i+1,j}\right) - h\sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

Videos: 2D without field

$$N = 150 \times 150,$$
$$T_H = 10J/k_B,$$

Monte Carlo Methods in Python

Demo: Ising Model and the Metropolis Algorithm

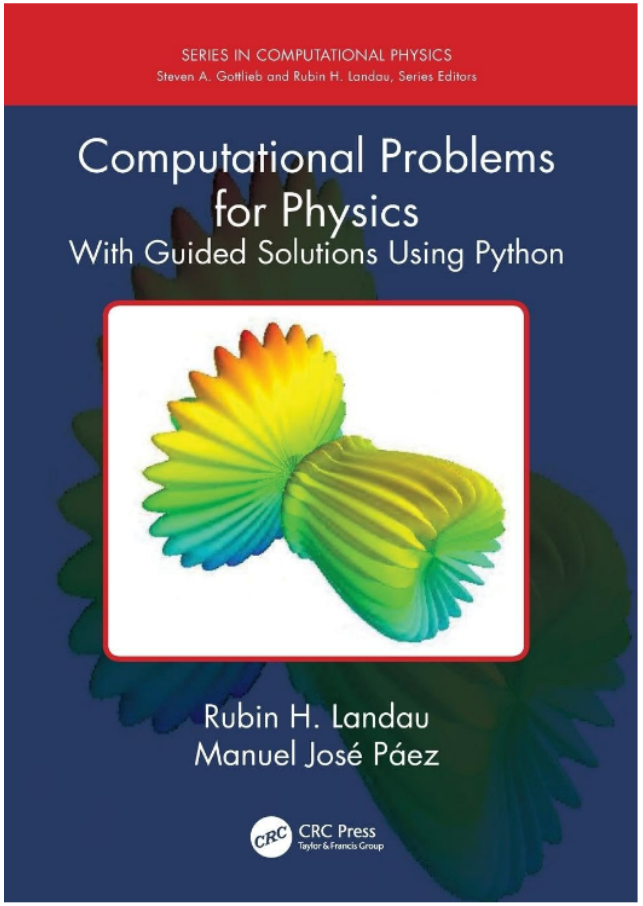
$$H = \begin{cases} -\sum_i s_i s_{i+1} - h \sum_i s_i & 1-D \\ -\sum_{ij} s_{i,j} \left(s_{i,j+1} + s_{i+1,j} \right) - h \sum_{i,j} s_{i,j} & 2-D \text{ square} \end{cases}$$

Videos: 2D without field

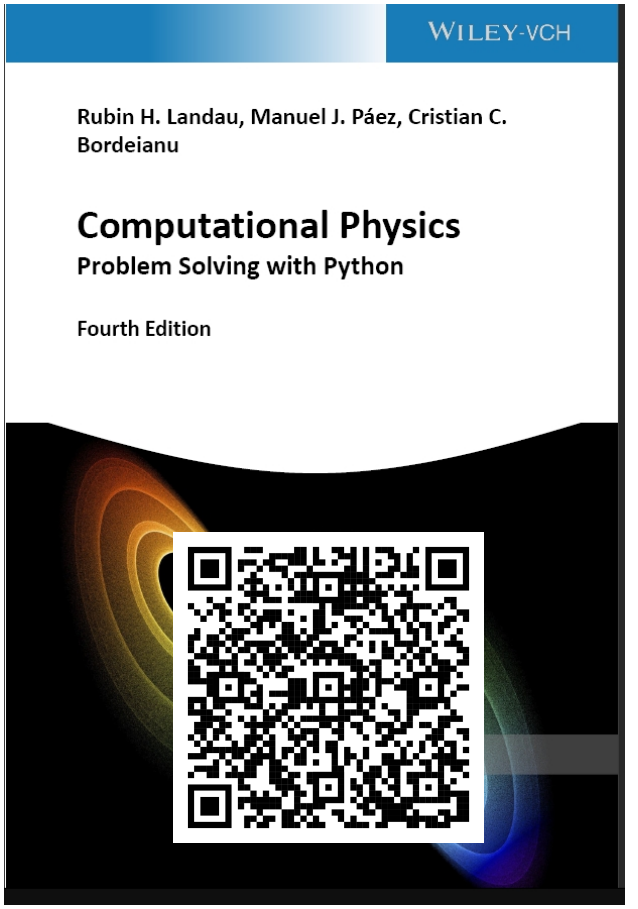
$$N = 150 \times 150,$$

$$T_L = 0.1 J / k_B$$

References



ISBN: 978-1138705418



https://tinyurl.com/landau-paez-bordeianu