

Lab 4 of Thursday 2nd October 2025

Exercise 1.

Let $X = [X_1, X_2, \dots, X_n] \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}([-1, 1]^n)$ be a random vector uniformly distributed over the n -dimensional square $\Gamma = [-1, 1]^n$, and define the random variable $Z = \mathbb{1}_{\|X\|_2 < 1}$. Observe that

$$I = \mathbb{E}[Z] = \int_{\Gamma} \mathbb{1}_{\|x\|_2 < 1} p(x) dx = \frac{1}{|\Gamma|} |B(0, 1)|,$$

where $p(x)$ is the PDF of $\mathcal{U}([-1, 1]^n)$, and $|B(0, 1)|$ is the volume of the n -dimensional sphere with center 0 and radius 1.

- 1) Let $n = 2$. Use Monte Carlo to approximate the value of I :

$$\bar{I}_N := \frac{1}{N} \sum_{k=1}^N Z_k,$$

For $N = 10, 100, 1000, 10000$, compute \bar{I}_N as well as an approximate confidence interval and compare with the exact value I . In addition, plot the relative error $\frac{|\bar{I}_N - I|}{I}$ versus N in logarithmic scale and verify the convergence rate.

- 2) (On the choice of N). By *a priori* analysis (knowing that $Z \sim \text{Bernoulli}(p)$ with $p = \pi/4$), determine a lower bound for $N(\alpha, \epsilon)$, $\alpha = 0.05$ such that

$$\mathbb{P}(|\bar{I}_N - \pi/4| > \epsilon) < \alpha$$

using the asymptotic normality result (CLT)

$$\frac{\bar{I}_N - \pi/4}{\sqrt{\text{Var}(Z)/N}} \sim N(0, 1).$$

Provide the lower bound formula as a function of ϵ . What is the order of growth of $N(\alpha = 0.05, \epsilon)$ w.r.t ϵ ? For $\epsilon = 0.1$ fixed, compute $N(\alpha = 0.05, \epsilon)$, and generate M independent MC approximations \bar{I}_N^j , $j = 1, 2, \dots, M$ with $M = 1000$. Compute empirically $\mathbb{P}(|\bar{I}_N - \pi/4| > \epsilon)$, and compare with the target $\alpha = 0.05$.

- 3) An important property of the MC method is that, under very weak regularity assumptions, an $O(N^{-1/2})$ convergence rate holds independently of the dimensionality of the underlying problem. To illustrate this, consider approximating $\mathbb{E}[Z]$ as in the first point, for $n = 6$.

Solution

1) A possible python implementation is attached. The results are summarized in Fig. 1

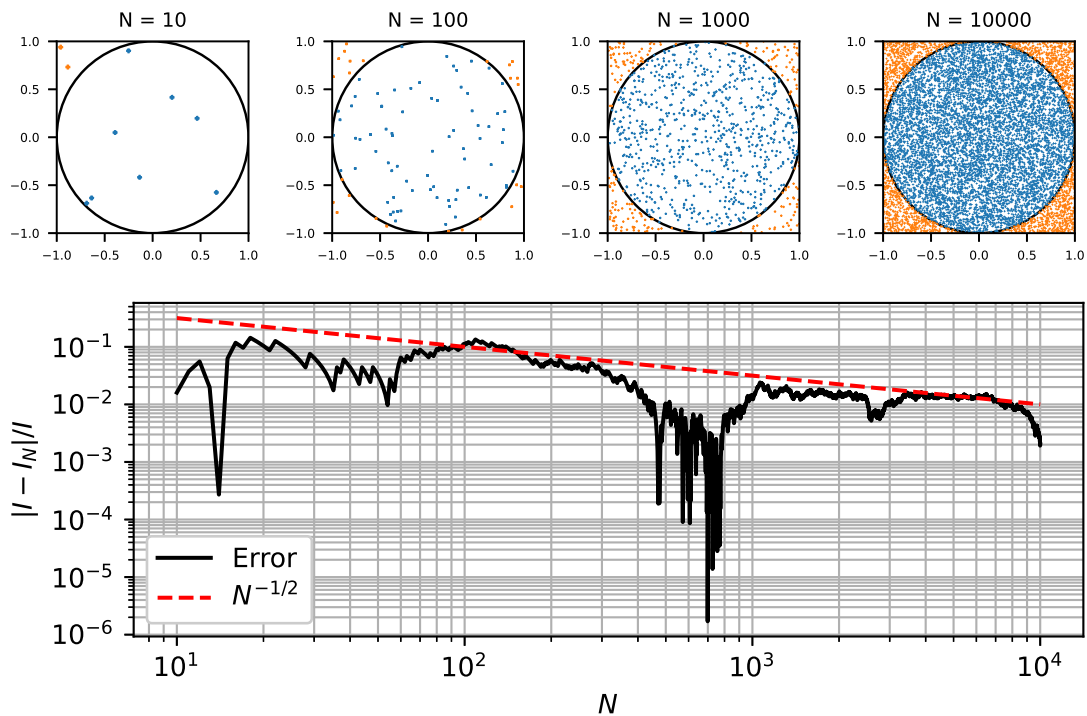


Figure 1: **Top:** Computing the area of a circle using Monte Carlo. **Bottom:** Monte Carlo error as a function of N , for N ranging from 10 to 10000. As we can see, even though there is some *randomness* associated to the rate (which is to be expected), on average, the rate decreases as $N^{-1/2}$

In addition, Fig. 2 shows the correct value of I compared with its 95% confidence interval for different values of N .

Python code:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

np.random.seed(42)

##### Generate circle plots to illustrate Monte Carlo
Ns = [10,100,1000,10000]
markersizes=[2,1,0.5,0.25]
n = 2
I = np.pi/4 # Area of circle divided by area of square
fig, ax = plt.subplots(1,len(Ns))

for i in range(len(Ns)):
    circle = plt.Circle((0,0),1,color='black',fill=False)
    N = Ns[i]
```

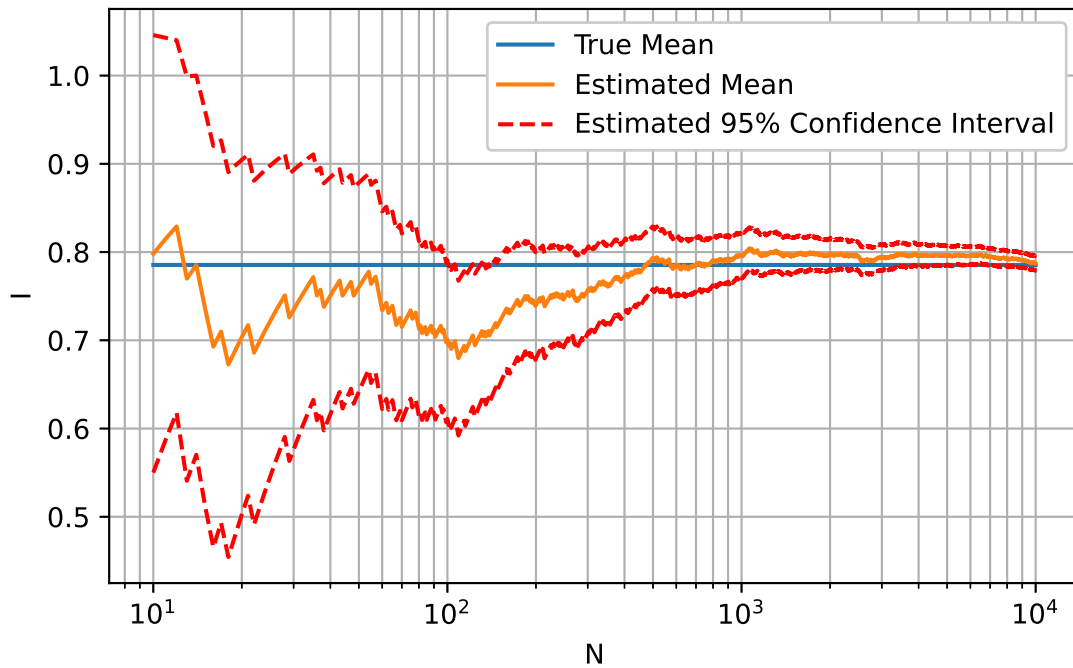


Figure 2: Estimated mean and confidence interval for different sample sizes

```

ms = markersizes[i]
print('Running for N =',N)

# Generate N samples of n-dimension vectors
X = st.uniform.rvs(loc=-1,scale=2,size=(N,n))

# Compute Z, I_N and relative error
Z = [int(np.linalg.norm(x)<1) for x in X]
I_N = np.mean(Z)

# Collect points that are inside and outside
Xin = [x for (x,z) in zip(X,Z) if z==1]
Xout = [x for (x,z) in zip(X,Z) if z==0]

# Plotting
ax[i].plot([x[0] for x in Xin], [x[1] for x in Xin],'+',markersize=ms)
ax[i].plot([x[0] for x in Xout], [x[1] for x in Xout],'+',markersize=ms)
ax[i].set_xlim((-1,1))
ax[i].set_ylim((-1,1))
titleString = 'N = %d' %N
ax[i].set_title(titleString,fontsize=8)
ax[i].set(aspect='equal')
ax[i].add_patch(circle)
ax[i].tick_params(axis='x',labelsize=5)
ax[i].tick_params(axis='y',labelsize=5)

plt.tight_layout()

# Generate additional data for the convergence graph
Ns_all = list(range(10,10001))

```

```

Xs_all = st.uniform.rvs(loc=-1,scale=2,size=(Ns_all[-1],n))
Zs_all = [int(np.linalg.norm(x)<1) for x in Xs_all]
mean = np.mean(Zs_all[:9])
var = np.var(Zs_all[:9])
means = []
variances = []
true_errors = []

for N in Ns_all:
    mean_new = N/(N+1)*mean + Zs_all[N-1]/(N+1)
    var_new = (N-1)/N*var + (Zs_all[N-1]-mean)**2/(N+1)
    means.append(mean_new)
    variances.append(var_new)
    true_errors.append(abs(mean_new-I)/I)

    mean = mean_new
    var = var_new

plt.figure()
plt.loglog(Ns_all, true_errors,color='black',label='Error')
plt.loglog(Ns_all, [N**-0.5 for N in Ns_all], '--',color='red',label=r'$N^{-1/2}$')
plt.xlabel(r'$N$')
plt.ylabel(r'$|I-I_N|/I$')
plt.legend()
plt.grid(which='both')
plt.gca().set_aspect(0.2)

# Plot the confidence intervals
plt.figure()
plt.semilogx([Ns_all[0], Ns_all[-1]], [I,I], label=r'True Mean')
plt.semilogx(Ns_all, means, label=r'Estimated Mean')
upperlim = [(m + st.norm.ppf(0.975)*np.sqrt(v/N)) for m,v,N in zip(means, variances,
    ↪ Ns_all)]
lowerlim = [(m - st.norm.ppf(0.975)*np.sqrt(v/N)) for m,v,N in zip(means, variances,
    ↪ Ns_all)]
plt.semilogx(Ns_all, upperlim, '--',color='red',label=r'Estimated 95% Confidence
    ↪ Interval')
plt.semilogx(Ns_all, lowerlim, '--',color='red')
plt.xlabel(r'$N$')
plt.ylabel(r'$I$')
plt.legend()
plt.grid(which='both')
plt.gca().set_aspect(3.0)
plt.show()

```

- 2) For determining the respective lower bounds for $N(\alpha, \epsilon)$, note first that since $Z \sim \text{Bernoulli}(p = \pi/4)$, we have

$$\text{Var}(Z) = \sigma^2 = \mathbb{E}(Z - p)^2 = p(1 - p),$$

and

$$\mathbb{E}|Z - p|^3 = \gamma^3 = (1 - p)^3 p + p^3(1 - p) = p(1 - p)[(1 - p)^2 + p^2].$$

When using the central limit theorem relation (called “leap of faith” as we are in the non-asymptotic setting), we may bound the probability of failure by

$$\mathbb{P}(|\bar{I}_N - I| \geq \epsilon) = \mathbb{P}\left(\sqrt{N} \frac{|\bar{I}_N - I|}{\sigma} > \frac{\sqrt{N}\epsilon}{\sigma}\right) = 2 \left(1 - \Phi\left(\frac{\sqrt{N}\epsilon}{\sigma}\right)\right).$$

To ensure that

$$2 \left(1 - \Phi\left(\frac{\sqrt{N}\epsilon}{\sigma}\right)\right) < \alpha,$$

one must use at least

$$N_{CLT}(\alpha, \epsilon) = \left\lceil \left(\frac{\sigma c_{1-\alpha/2}}{\epsilon} \right)^2 \right\rceil,$$

samples, where $\lceil x \rceil := \min\{z \in \mathbb{Z} \mid z \geq x\}$ and $c_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the standard normal distribution, i.e., $\Phi(c_{1-\alpha/2}) = 1-\alpha/2$. The growth of $N_{CLT}(\alpha, \epsilon)$ w.r.t. ϵ is $O(\epsilon^{-2})$. For $\epsilon = 0.1$ and $\alpha = 0.05$, we have $N_{CLT} = 65$. We run a Monte Carlo simulation with $M = 1000$ independent runs, each with N_{CLT} samples. The empirical probability of failure is 0.038. Increasing M to 10000 gives an empirical probability of failure of 0.0486, closer to the target α .

Python code:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
import scipy.optimize as so

np.random.seed(42)

p = np.pi/4
sigma = ( p*(1-p) )**(1/2)
gamma = p*(1-p)*((1-p)**2 + p**2)**(1/3)

alpha = 0.05
eps = 0.1
# Central Limit theorem bound
N_CLT = int(np.ceil( ( sigma*st.norm.ppf(1-alpha/2)/eps )**2 ))
print("N_CLT \t= ", N_CLT)

# -- Monte Carlo simulation to verify CLT --
M = 10000

samples = np.random.binomial(1, p, size=(M, N_CLT))
I_estimates = samples.mean(axis=1)
prob_empirical = np.mean(np.abs(I_estimates - p) > eps)

print(f"Empirical probability = {prob_empirical:.4f}")
print(f"Target alpha = {alpha:.4f}")

plt.hist(I_estimates, bins=40, density=True, alpha=0.7, color='steelblue', edgecolor='k')
plt.axvline(p, color='r', linestyle='--', label=r'\pi/4$')
plt.axvline(p + eps, color='g', linestyle='--', label=r'\pi/4 \pm \epsilon$')
plt.axvline(p - eps, color='g', linestyle='--')
plt.title(f"Distribution of MC estimates ($M={M}$, $N={N_CLT}$)")
plt.xlabel(r'\overline{I}_N$')
plt.ylabel('Density')
plt.legend()
plt.show()
```

- 3) We rerun the Monte Carlo simulations for $n = 6$. The results are shown in Fig. 3 and Fig. 4.

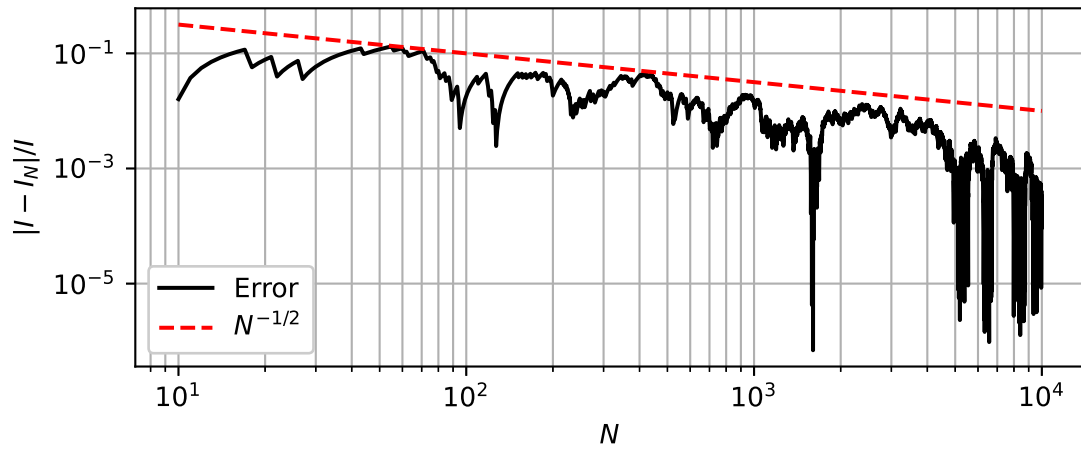


Figure 3: Numerical approximation of the 6-dimensional unit ball volume by respectively the Monte Carlo method

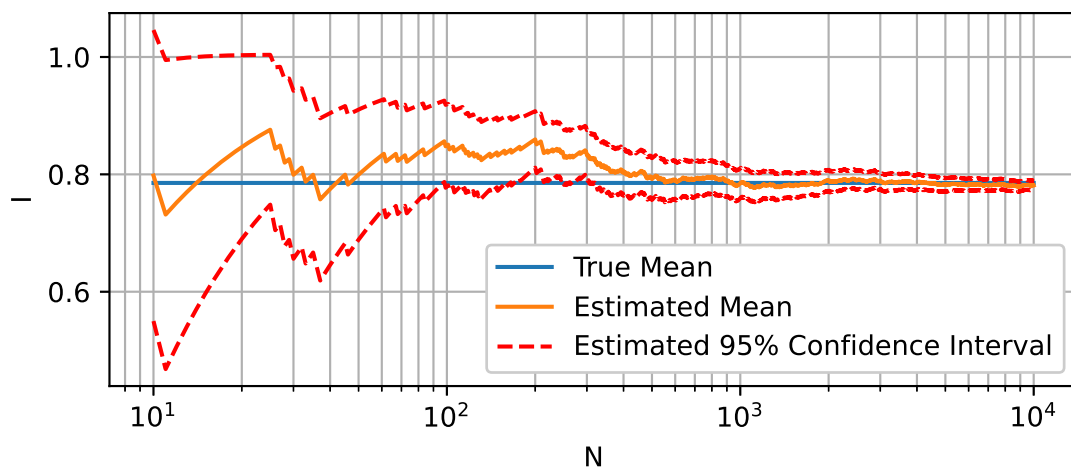


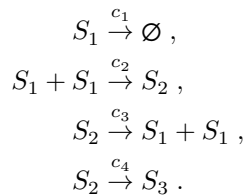
Figure 4: Estimated mean and confidence interval for different sample sizes

We observe that the Monte Carlo method approximates the reference solution with a rate $1/2$ even for higher dimensions.

Exercise 2.

Note: Refer to Section 4.7 of the lecture notes.
 Consider the chemical reactions between three species S_1, S_2, S_3 , which are determined by the

following four reaction channels:



To simulate this system, consider the process $N_t = (N_t^1, N_t^2, N_t^3) \in \mathbb{N}_0^3$, where N_t^i denotes the number of molecules of species S_i at time $t \geq 0$. In fact, this process is a time-continuous Markov chain with transition probabilities given by

$$\begin{aligned} \mathbb{P}(N_{t+h} = N_{t,1} = (N^1 - 1, N^2, N^3) | N_t = (N^1, N^2, N^3)) &= a_1(N_t)h + o(h), \\ \mathbb{P}(N_{t+h} = N_{t,2} = (N^1 - 2, N^2 + 1, N^3) | N_t = (N^1, N^2, N^3)) &= a_2(N_t)h + o(h), \\ \mathbb{P}(N_{t+h} = N_{t,3} = (N^1 + 2, N^2 - 1, N^3) | N_t = (N^1, N^2, N^3)) &= a_3(N_t)h + o(h), \\ \mathbb{P}(N_{t+h} = N_{t,4} = (N^1, N^2 - 1, N^3 + 1) | N_t = (N^1, N^2, N^3)) &= a_4(N_t)h + o(h), \\ \mathbb{P}(N_{t+h} = N_{t,5} = (N^1, N^2, N^3) | N_t = (N^1, N^2, N^3)) &= 1 - h \sum_{j=1}^4 a_j(N_t) + o(h), \end{aligned}$$

for h sufficiently small, where $N_{t,k}, k \in \{1, \dots, 5\}$ indexes the possible transitions. Here, the so-called propensity functions are

$$a_1(N) = c_1 N^1, \quad a_2(N) = c_2 \frac{N^1(N^1 - 1)}{2}, \quad a_3(N) = c_3 N^2, \quad a_4(N) = c_4 N^2,$$

with $N = (N^1, N^2, N^3)$.

- 1) Try to construct the transition matrix corresponding to the above transition probabilities and note the challenges. Is it possible to simulate the chemical reaction without the explicit Q matrix?

Hint. Think back to how you simulated the process in Exercise 2.1.

- 2) Utilise Algorithm 1 to simulate the chemical reaction system. Plot a time series for each species' number of molecules for $t \in [0, T]$, $T = 0.2$, for the reaction rates

$$c_1 = 1, \quad c_2 = 5, \quad c_3 = 15, \quad c_4 = \frac{3}{4},$$

using $N_0 = (400, 800, 0)$ as initial number of molecules. Repeat the simulation for the same reaction rates c_1, \dots, c_4 also for $T = 5$.

Solution

Let $\{N_t \in \mathbb{N}_0^3: t \in [0, T]\}$ be the Markov jump process that describes the number of each species present in the chemical reaction system. That is, unlike the processes covered during the lecture, we have to deal with both a vector-valued process and the fact that the state space may be unbounded. Regardless of these differences, the Q -matrix could be constructed as usual, namely by $Q = (q(n, m), n, m \in \mathbb{N}_0^3)$, where

$$q(n, m) = \begin{cases} \lim_{h \rightarrow 0} \frac{\mathbb{P}(N_{t+h}=n | N_t=m)}{h}, & n \neq m, \\ -\lim_{h \rightarrow 0} \frac{1 - \mathbb{P}(N_{t+h}=n | N_t=n)}{h}, & n = m. \end{cases}$$

Algorithm 1: Reaction simulation

- 1: Set $N_0 = (N_0^1, N_0^2, N_0^3)$, $J_0 = 0$
 - 2: **for** $n = 1, 2, \dots$ **do**
 - 3: Compute $\lambda = \sum_{j=1}^4 a_j(N_{J_{n-1}})$
 - 4: Generate $S_n \sim \text{Exp}(\lambda)$ and set $J_n = J_{n-1} + S_n$
 - 5: Generate $I \in \{1, 2, 3, 4\}$ with probability mass function $\mathbb{P}(I = j) = \frac{a_j(N_{J_{n-1}})}{\sum_{l=1}^4 a_l(N_{J_{n-1}})}$, which is the probability that the j^{th} reaction happens.
 - 6: Set $N_t = N_{J_{n-1}} \forall t \in [J_{n-1}, J_n)$ and $N_{J_n} = N_{t,I}$
 - 7: **end for**
-

If one wanted to implement this matrix, then one would, of course, have to truncate the state space appropriately. However, implementing this matrix explicitly is neither needed nor advisable! In fact, the code below shows an exemplary implementation, which is used to produce the times series shown in Figure 5. There first the jump times are generated, before its is determined which reaction takes place.

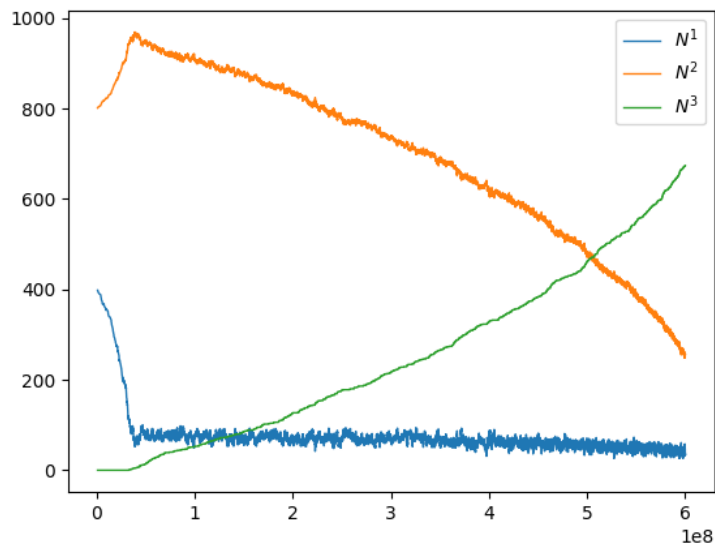


Figure 5: Species' concentration as a function of $t \in [0, 5]$.

Python code

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt

np.random.seed(42)

#defines constant and preallocates
c = np.array([1,5,15,3./4])
```

```

x, y, z = 400, 800, 0
xx = np.array([])
yy = np.array([])
zz = np.array([])
tt = np.array([])
t1 = 600000000
t = 0
i = 0
#simulates dynamics
while t < t1:
    v = np.array([x, x*(x - 1)/2., y, y])
    a = c * v
    lam = np.sum(a)
    #samples
    p = a / np.sum(a)
    rand = st.uniform.rvs()
    r = np.where(np.cumsum(p) >= rand)[0].min()
    t = t + st.expon.rvs(scale=lam)
    print(t, lam)
    #changes dynamics according to r
    if r == 0:
        x = x - 1
    elif r == 1:
        x, y = x - 2, y + 1
    elif r == 2:
        x, y = x + 2, y - 1
    elif r == 3:
        y, z = y - 1, z + 1

    xx = np.append(xx,x)
    yy = np.append(yy,y)
    zz = np.append(zz,z)
    tt = np.append(tt,t)
    i = i + 1
#plots
plt.plot(tt, xx, linewidth = 1., label = r'$N^1$')
plt.plot(tt, yy, linewidth = 1., label = r'$N^2$')
plt.plot(tt, zz, linewidth = 1., label = r'$N^3$')
plt.legend()
plt.show()

```

Exercise 3.

Let $\{N_t \in \mathbb{N}_0 : t \geq 0, N_0 = 0\}$ be a Poisson process with rate λ .

- 1) Show that, conditional on the event $\{N_T = n\}$, the jump times J_1, \dots, J_n have joint density function

$$f_{J_1, \dots, J_n}(j_1, \dots, j_n) = n! T^{-n} \mathbb{I}(0 \leq j_1 \leq \dots \leq j_n \leq T).$$

In other words, show that conditional on $\{N_T = n\}$, the jump times J_1, \dots, J_n have the same distribution as an ordered sample of size n from the uniform distribution on $[0, T]$.

Hint. Use the joint distribution of the holding times S_1, \dots, S_{n+1} to first derive the joint distribution of the jump times, where $S_{i+1} = J_{i+1} - J_i$. Then compute the conditional distribution of the jump times given that $N_T = n$, using the fact that $\{N_T = n\} = \{J_n \leq T < J_{n+1}\}$ a.s.

- 2) Use the property above to propose an algorithm to generate the process $N_t, t \in (t_1, t_2)$, conditional upon $N_{t_1} = n_1$ and $N_{t_2} = n_2 > n_1$. Such a process is called *Poisson bridge*.

Solution

- 1) Since the Poisson process $\{N_t \in \mathbb{N}_0 : t \geq 0, N_0 = 0\}$ is non-decreasing, the condition $N_T = n$ implies that n jumps need to take place. Let's denote these jump times by J_1, \dots, J_n . As a matter of fact, the jump times are related to the $(n+1)$ holding times S_1, \dots, S_{n+1} by $S_{i+1} = J_{i+1} - J_i$ for $i = 0, \dots, n$, with the convention that $J_0 = 0$. We know that the holding times are i.i.d. $\text{Exp}(\lambda)$ for some $\lambda > 0$. Consequently, their joint PDF reads

$$f_{S_1, \dots, S_{n+1}}(s_1, \dots, s_{n+1}) = \lambda^{n+1} \exp\left(-\lambda \sum_{i=1}^{n+1} s_i\right) \mathbb{I}(s_1, \dots, s_{n+1} \geq 0).$$

As $\sum_{i=1}^{n+1} S_i = \sum_{i=1}^{n+1} (J_i - J_{i-1}) = J_{n+1}$, the joint PDF of the jumping times J_1, \dots, J_{n+1} is given by

$$f_{J_1, \dots, J_{n+1}}(t_1, \dots, t_{n+1}) = \lambda^{n+1} e^{(-\lambda t_{n+1})} \mathbb{I}(0 \leq t_1 \leq \dots \leq t_{n+1}).$$

For any (Borel set) $A \subseteq \mathbb{R}^n$ we thus find

$$\begin{aligned} \mathbb{P}((J_1, \dots, J_n) \in A | N_T = n) &= \frac{\mathbb{P}((J_1, \dots, J_n) \in A, N_T = n)}{\mathbb{P}(N_T = n)} \\ &= \mathbb{P}((J_1, \dots, J_n) \in A, J_n \leq T < J_{n+1}) \\ &= \frac{n! T^{-n}}{\lambda^n e^{-\lambda T}} \int_T^\infty \left(\int_A \lambda^{n+1} e^{-\lambda t_{n+1}} \mathbb{I}(0 \leq t_1 \leq \dots \leq t_n \leq T) dt_1 dt_2 \dots dt_n \right) dt_{n+1} \\ &= \frac{\lambda n! T^{-n}}{e^{-\lambda T}} \int_T^\infty e^{-\lambda t_{n+1}} dt_{n+1} \int_A \mathbb{I}(0 \leq t_1 \leq \dots \leq t_n \leq T) dt_1 dt_2 \dots dt_n \\ &= n! T^{-n} \int_A \mathbb{I}(0 \leq t_1 \leq \dots \leq t_n \leq T) dt_1 \dots dt_n, \end{aligned}$$

as claimed.

- 2) A possible Python code is shown below.

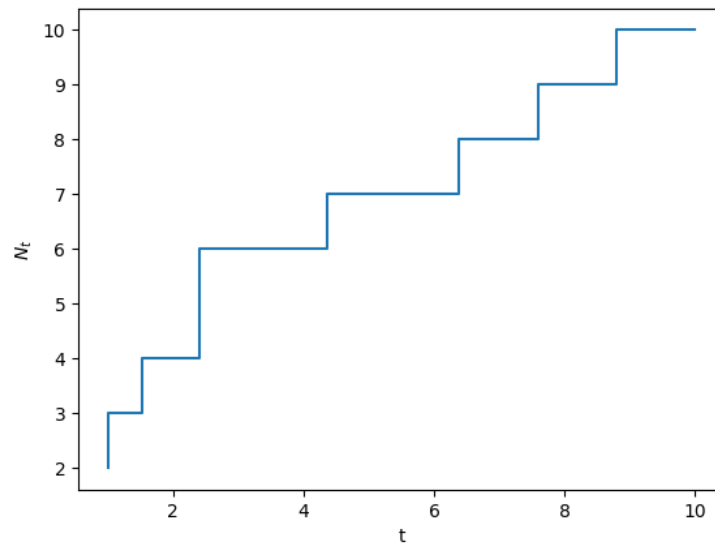


Figure 6: Simulation of the Poisson jump process.

Python code:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt

np.random.seed(42)

#defines some parameters
t1 = 1.
t2 = 10.
dt = t2 - t1
n1 = 2
n2 = 10
#starts the simulation
n = n2 - n1
J = np.sort(dt * st.uniform.rvs(size = (n,)))
t = t1 + np.hstack([0., J])
t = np.hstack([t, t2])
N = n1 + np.array(range(n+1))
N = np.hstack([N,n2])
# Plots
plt.step(t, N)
plt.xlabel('t')
plt.ylabel(r'$N_t$')
plt.show()
```

Exercise 4.

Let $\{N_t, t \geq 0, N_0 = 0\}$ be a non-homogeneous Poisson process with rate $\lambda : [0, \infty) \mapsto \mathbb{R}_+$. In addition, define $\Lambda(t) = \int_0^t \lambda(s) ds$, and let $\{\tilde{N}_t, t \geq 0, \tilde{N}_0 = 0\}$ be a homogeneous Poisson process with rate one.

- 1) Show that the non-homogeneous Poisson process can be obtained as $N_t = \tilde{N}_t \circ \Lambda(t)$, i.e., $N_t = \tilde{N}_{\Lambda(t)}$.
- 2) Simulate a non-homogeneous Poisson process with rate function $\lambda(t) = \sin^2(t)$ on the interval $[0, 50]$.

(Optional) Exercise 5.

- 1) Generate a random walk $\{X_n \in \mathbb{Z}, n \in \mathbb{N}_0, X_0 = 0\}$ with transition probabilities $\mathbb{P}(X_{n+1} = j | X_n = j - 1) = \mathbb{P}(X_{n+1} = j | X_n = j + 1) = a$, $\mathbb{P}(X_{n+1} = j | X_n = j) = 1 - 2a$, for some $0 < a \leq 1/2$.
- 2) Consider the rescaled process $Y_{t_i} := \sqrt{\Delta t / (2a)} X_i$ for $i = 0, \dots, n$ with $t_i = i \Delta t$. Compare this process with the process W_{t_i} , $i = 0, \dots, n$, where W_t denotes a Wiener process with $W_0 = 0$. That is, show that both processes “look similar” in the limit as $\Delta t \rightarrow 0$ by plotting multiple realizations of both processes for $n = \lceil 1/\Delta t \rceil$.
- 3) **(Optional):** More theoretical analysis of the observed phenomenon:
 - a) Consider the spatial mesh $x_m = m \Delta x = m \sqrt{\Delta t / (2a)}$ for $m \in \mathbb{Z}$ and the following notation for the rescaled process’ probability mass function at time t_i :

$$\bar{u}(t_i, x_m) := \mathbb{P}(Y_{t_i} = x_m | Y_0 = 0), \quad m \in \mathbb{Z}, i = 0, 1, \dots$$

Use the discrete Chapman–Kolmogorov formula

$$\mathbb{P}(Y_{t_{i+1}} = x_m | Y_0 = 0) = \sum_k \mathbb{P}(Y_{t_{i+1}} = x_m | Y_{t_i} = x_k) \mathbb{P}(Y_{t_i} = x_k | Y_0 = 0) \quad (5.1)$$

to derive a difference equation for $\bar{u}(t_{i+1}, x_m)$ in terms of $\bar{u}(t_i, \cdot)$.

- b) Show that the difference equation obtained in [3a](#) corresponds to a finite difference approximation of the one dimensional heat equation

$$u_t(t, x) = \frac{u_{xx}(t, x)}{2}, \quad x \in \mathbb{R}, t > 0,$$

on a uniform grid $x_i = i \Delta x$ and $t_j = j \Delta t$ with $\Delta t = 2a \Delta x^2$, using a second order centered finite difference stencil in space and a first order forward Euler scheme in time.

- c) For the standard Wiener process with $\mathbb{P}(W_0 = 0) = 1$, we denote the probability density function at time $t > 0$ by

$$u(t, x) := \frac{e^{-x^2/(2t)}}{\sqrt{2\pi t}}, \quad x \in \mathbb{R}.$$

For all $t > 0$ and $x \in \mathbb{R}$, show that the density satisfies the same heat equation introduced in point [3b](#).

Solution

1.–2. A possible Python implementation for both parts is shown at end of this section. Figure 7 illustrates typical realizations of both the Wiener process W_t and the rescaled random walk Y_t for two values of Δt . We observe that, as $\Delta t \rightarrow 0$, both processes “look similar” in the sense that one could not identify the rescaled random walk Y_t , if it wasn't for the legend. Furthermore, similarly to the Wiener process, one can easily show that the increments of the rescaled Random Walk are independent with variance equal to $\sqrt{\Delta t}$.

3. a) We have that,

$$\begin{aligned}
 \bar{u}(t_{i+1}, x_m) &= \mathbb{P}(Y_{t_{i+1}} = x_m | Y_0 = 0) \\
 &= \sum_{k \in \mathbb{Z}} \mathbb{P}(Y_{t_{i+1}} = x_m | Y_{t_i} = x_k) \mathbb{P}(Y_{t_i} = x_k | Y_0 = 0) \\
 &= \sum_{k \in \mathbb{Z}} \mathbb{P}(Y_{t_{i+1}} = x_m | Y_{t_i} = x_k) \bar{u}(t_i, x_k) \\
 &= (1 - 2a) \bar{u}(t_i, x_m) + a \bar{u}(t_i, x_{m-1}) + a \bar{u}(t_i, x_{m+1}) \tag{5.2}
 \end{aligned}$$

b) Using the Taylor series for the solution of the heat equation, we have that

$$\begin{aligned}
 u(t_i, x_{m-1}) &= u(t_i, x_m) - \Delta x u_x(t_i, x_m) + \frac{\Delta x^2}{2} u_{xx}(t_i, x_m) + \mathcal{O}(\Delta x^4), \\
 u(t_i, x_{m+1}) &= u(t_i, x_m) + \Delta x u_x(t_i, x_m) + \frac{\Delta x^2}{2} u_{xx}(t_i, x_m) + \mathcal{O}(\Delta x^4), \\
 u(t_{i+1}, x_m) &= u(t_i, x_m) + \Delta t u_t(t_i, x_m) + \mathcal{O}(\Delta t^2).
 \end{aligned}$$

With the choice $\Delta x = \sqrt{\Delta t/2a}$, this leads to

$$\begin{aligned}
 u(t_{i+1}, x_m) &= u(t_i, x_m) + \frac{\Delta t}{2} u_{xx}(t_i, x_m) + \mathcal{O}(\Delta t^2) \\
 &= u(t_i, x_m) + \frac{\Delta t}{2} \frac{u(t_i, x_{m-1}) + u(t_i, x_{m+1}) - 2u(t_i, x_m)}{\Delta x^2} + \mathcal{O}(\Delta t^2 + \Delta x^4) \\
 &= (1 - 2a)u(t_i, x_m) + au(t_i, x_{m-1}) + u(t_i, x_{m+1}) + \mathcal{O}(\Delta t^2),
 \end{aligned}$$

which corresponds to Eq. (5.2), up to error terms that vanish as $\Delta t \rightarrow 0$.

c) This can be seen by simply substituting the density into the standard heat equation.

Python code

```

import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt

np.random.seed(42)

#defines the standard random walk
def RW(steps = 100, a = 1./2):
    assert a <= 1./2, 'Parameter a should be at most 0.5.'
    N = steps
    X = np.zeros(N+1)
    for i in range(1, N+1):
        U = st.uniform.rvs()
        X[i] = X[i-1] + (U <= a) - (a < U) * (U <= 2*a)

```

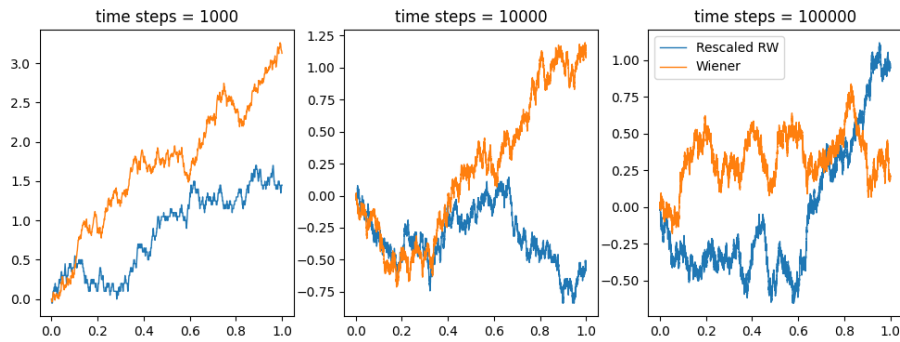


Figure 7: Realizations of Y_t and W_t for different dt .

```

return X

#defines the rescaled random walk
def rescaledRW(T = 1., n = 100, a = 0.2):
    X = RW(n, a)
    dt = T / (n+1)
    return np.sqrt(dt / (2*a)) * X

#defines the Wiener process
def Wiener(T = 1., n = 100):
    W = np.zeros(n+1)
    dt = T / (n+1)
    for i in range(1,n+1):
        W[i] = W[i-1] + np.sqrt(dt) * st.norm.rvs()
    return W

#X = rescaledRW(n = 1000, a = 0.2)
#W = Wiener(n = 1000)
fig, axes = plt.subplots(nrows = 1, ncols = 3, figsize = (12,4))
i = 1
for ax in axes:
    n = 100*(10**i)
    print('running for n = ', n)
    X = rescaledRW(n = n, a = 0.2)
    W = Wiener(n = n)
    t = np.linspace(0., 1, n+1)
    ax.plot(t, X, linewidth = 1, label = 'Rescaled RW')
    ax.plot(t, W, linewidth = 1, label = 'Wiener')
    ax.set_title('time steps = '+str(n))
    i = i + 1
plt.legend()
plt.show()

```