Stochastic Simulation
Professor: Prof. Fabio Nobile
Assistant: Eliott Van Dieren

Week of 2025-11-10 to 2025-11-14

Lab 9 of Thursday 13th November 2025

Exercice 1.

Consider the random walk $\{X_n \in \mathbb{Z}, n \in \mathbb{N}_0\}$ with $X_0 \sim \lambda$ on the lattice $\mathcal{X} := \{i : i \in \mathbb{Z}, |i| \leqslant 2N^2\}$, whose transition probabilities are given by

$$\mathbb{P}(X_{n+1} = i \pm 1 | X_n = i) = \alpha\left(1 \mp \frac{i}{2N^2}\right), \quad |i| \leqslant 2N^2,$$
$$\mathbb{P}(X_{n+1} = i | X_n = i) = 1 - 2\alpha,$$

for some $\alpha \in]0, \frac{1}{2}]$ and $N \in \mathbb{N}$.

1) Implement an algorithm that simulates the Markov Chain $\{X_n \in \mathbb{Z}, n \in \mathbb{N}_0\}$. Use your implementation to address the following points for different values of $N \geqslant 1$:

   a) Assess numerically that the Markov chain converges to an invariant distribution by simulating multiple (independent) chains, each starting in 0 (i.e. $\lambda = \delta_0$). That is, monitor the following quantities (rather, suitable Monte Carlo approximations) as functions of the Markov chain length n.

      i. $\mathbb{E}_\lambda({X_n}^p)^{1/p}$ for $p \in \{1, 2, 4\}$,
      ii. $M_{X_n}(t) := \mathbb{E}_\lambda(e^{tX_n})$ for $t \in [-1, 1]$.

      Speculate on the invariant distribution.

   b) For $N = 10$, compute the eigenvalues and eigenvectors of the transition matrix P. Use the obtained results to deduce the invariant distribution π. **Hint:** Use `np.linalg.eig(P)`.

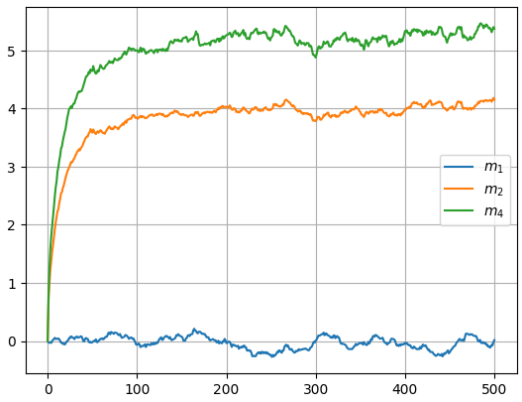
   c) Assess the validity of the ergodic theorem. That is, verify that

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} f(X_n) = \mathbb{E}_\pi(f), \quad \mathbb{P}_\lambda\text{–a.s.},$$

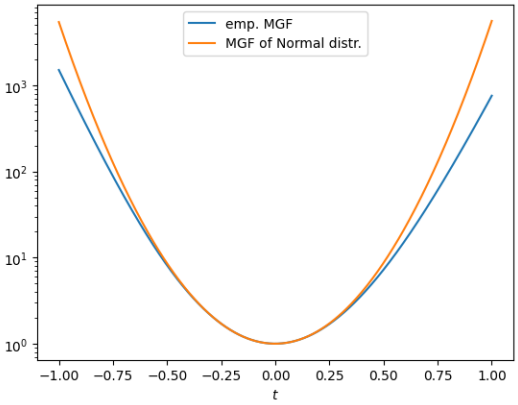
   for any $f : X \to \mathbb{R}$, with $\sum_n |f(X_n)|\pi_n < \infty$. Specifically, investigate this identity for the moments used in Point 1(a)i and monitor the rate of convergence as a function of n.

2) Consider the rescaled Markov chain $Y_n := \frac{1}{N} X_n$ with state space $\mathcal{Y} := \{x_i \equiv \frac{i}{N} : i \in \mathbb{Z}, |i| \leqslant 2N^2\}$. Show by means of numerical simulations that the invariant distribution $\nu \equiv \nu_N$ of $\{Y_n \in \mathbb{Z}, n \in \mathbb{N}_0\}$ is an accurate approximation to the standard normal measure. Moreover, illustrate that the approximation quality improves as N increases.

Solution



(a) Moments $m_p \equiv m_p(n) = \sqrt[p]{\mathbb{E}(X_n{}^p)}$ as a function of n.



(b) Empirical moment generating function compared to the one of a Normal distribution.

Figure 1: Convergence plots

1) A possible implementation is shown below:

```
def latticeRW(n=100,X0=0,N=2,aa=0.5):
    X=np.zeros(n+1)
    X[0]=X0
    t=np.arange(0,n+1)
    for i in range(n):
        xcurr=X[i]
        pup=(1-xcurr/2/N**2)*aa*(np.abs(xcurr)<=2*N**2)
        plow=(1+xcurr/2/N**2)*aa*(np.abs(xcurr)<=2*N**2)
        r=np.random.random(1)
        if (r<=plow):                    #decreses state
            X[i+1]=X[i]-1
        elif( (plow<r)*(r<=plow+pup)): # increses state
            X[i+1]=X[i]+1
        else:
            X[i+1]=X[i]
    return X,t
```

a) The numerical results shown here are obtained for $\alpha = 1/4$, $N = 4$, and the expectations are approximated by Monte Carlo generating $R = 10^3$ independent chains. When inspecting the evolution of the resulting moment approximations as a function of n, Fig. 1a, we notice an exponential convergence to a stationary state. Moreover, the transition probabilities of the chain leads us to suspect that the invariant distribution could be close to a Gaussian. This suspicion is further strengthened when comparing the approximated moment generating function at $n = 500$ with the moment generating function of a normal distribution, as is shown in Figure 1b.

b) If the Markov chain converges, then we expect exponential convergence $e(n) = |\lambda_2|^n$, where λ_2 is the eigenvalue with largest absolute value smaller than one. This is also visible in the numerical results of the previous point. Estimates of this value for $\alpha = 1/4$ are shown below:

```
N = 1: expected rate of convergence (in TV norm): C*|lambda_2|^n = C*0.75^n
N = 2: expected rate of convergence (in TV norm): C*|lambda_2|^n = C*0.9375^n
N = 4: expected rate of convergence (in TV norm): C*|lambda_2|^n = C*0.984375^n
N = 8: expected rate of convergence (in TV norm): C*|lambda_2|^n = C*0.996094^n
N = 10: expected rate of convergence (in TV norm): C*|lambda_2|^n = C*0.9975^n
N = 20: expected rate of convergence (in TV norm): C*|lambda_2|^n = C*0.999375^n
```

c) As a consequence of the CLT for Markov chains, we observe the canonical $1/\sqrt{n}$ convergence rate. We observe the trend in Fig. 2

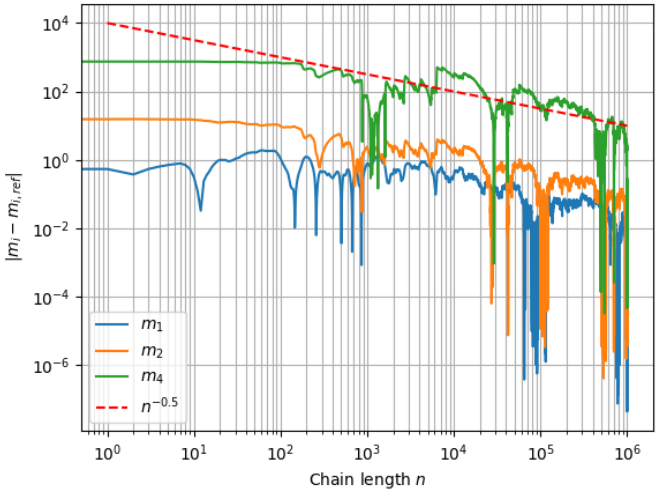


Figure 2: Convergence of ergodic estimator to reference value

Python code for lattice simulation:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Notice that this takes quite a bit of time to run.
"""
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

def latticeRW(n=100,X0=0,N=4,aa=0.25):
        X=np.zeros(n+1)
        X[0]=X0
        t=np.arange(0,n+1)
        for i in range(n):
                xcurr=X[i]
                pup=(1-xcurr/2/N**2)*aa*(np.abs(xcurr)<=2*N**2)
                plow=(1+xcurr/2/N**2)*aa*(np.abs(xcurr)<=2*N**2)
                r=np.random.random(1)
                if (r<=plow):                   #decreses state
                        X[i+1]=X[i]-1
                elif( (plow<r)*(r<=plow+pup)): # increses state
```

```
                        X[i+1]=X[i]+1
                else:
                        X[i+1]=X[i]
        return X,t


#
#   Defines some parameters
#
N=4
x0=0
aa=1/4
N_runs=int(1E3) #this might be quite a large number
n=500
x1=np.zeros((N_runs,n+1))
x2=np.zeros((N_runs,n+1))
x4=np.zeros((N_runs,n+1))
M=np.zeros((n+1,n+1))
tau=np.linspace(-1,1,n+1)

# runs the chain many times and obtains the quantities needed
for i in range(N_runs):
    x1[i,:]=latticeRW(n,x0,N,aa)[0]
    x2[i,:]=x1[i,:]**2.0
    x4[i,:]=x1[i,:]**4.0

#computes moments
e1=np.mean(x1,0)
e2=np.mean(x2,0)**0.5
e4=np.mean(x4,0)**0.25
Mx=np.mean(M,0)

plt.plot(e1);
plt.plot(e2);
plt.plot(e4);
plt.grid(True,which='both')
plt.legend([r'$m_1$',r'$m_2$',r'$m_4$'])
plt.show()

# computes the MGF
tauvec=np.linspace(-1,1,n+1)
ntau=n+1
MGF=np.zeros((n+1,ntau))
for r in range(N_runs):
    for i in range(ntau):
        MGF[:,i]=MGF[:,i]+np.exp(tauvec[i]*x1[r,:])
MGF=MGF/N_runs
mu=e1[-1]
si2=e2[-1]**2.0-mu**2
plt.semilogy(tauvec,MGF[-1,:])
plt.semilogy(tauvec,np.exp(tauvec*mu+si2/2*tauvec**2.));
plt.legend(['emp. MGF','MGF of Normal distr.']);
plt.xlabel(r'$t$');

plt.grid(True,which='both')

plt.show()
```

Python code for matrix construction:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```python
#Constructs matrix and outputs 2nd largest eig

import numpy as np
import matplotlib.pyplot as plt


def build_matrix(a,N):
    Nt=4*N**2+1
    P=np.zeros((Nt,Nt))

    for i in range(Nt):
        xcurr=i-2*N**2
        P[i,i]=1-2*a
        if (i<Nt-1):
            P[i,i+1]=(1-xcurr/2/N**2)*a*(np.abs(xcurr)<=2*N**2)
        if (i>0):
            P[i,i-1]=(1+xcurr/2/N**2)*a*(np.abs(xcurr)<=2*N**2)
    # obtains second largest eig
    e,v=np.linalg.eig(P)
    e.sort()
    return P,e[-2]

a=1/4
N=[1,2,4,8,10,20]
for n in N:
    _,e=build_matrix(a,n)
    e=np.real(e) # to remove the 0j imaginary part
    print('N = '+str(n) + ': expected rate of convergence (in TV norm): C*|lambda_2|^n =
    ↪  C*'+str(round(e,6))+'^n')
```

Exercice 2.

Recall that the standard Metropolis-Hastings algorithm accepts a new candidate state j drawn from the transition matrix Q, given the current state i, with probability $\alpha(i,j) = \min\left(\frac{\pi_j Q_{ji}}{\pi_i Q_{ij}}, 1\right)$, where π is the target probability measure. Consider now a Metropolis-Hastings algorithm that uses the follwing alternative acceptance probabilities

$$\alpha_1(i,j) = \frac{\pi_j Q_{ji}}{\pi_j Q_{ji} + \pi_i Q_{ij}},$$

and

$$\alpha_2(i,j) = \frac{\delta_{ij}}{\pi_i Q_{ij}},$$

with δ such that $\delta_{ij} \leqslant \pi_i Q_{ij} \forall i,j$. Show that, in both cases, the produced Markov chain satisfies the detailed balance condition.

Solution

Note that the detailed balance condition reads:

$$\alpha(i,j)\pi_i Q_{ij} = \alpha(j,i)\pi_j Q_{ji}.$$

Consider the first case $\alpha_1(i,j) = \frac{\pi_j Q_{ji}}{\pi_j Q_{ji} + \pi_i Q_{ij}}$. We then have that

$$\alpha_1(i,j)\pi_i Q_{ij} = \frac{\pi_j Q_{ji}\pi_i Q_{ij}}{\pi_j Q_{ji} + \pi_i Q_{ij}} = \alpha_1(j,i)\pi_j Q_{ji}.$$

Consider now the second case $\alpha_2(i,j) = \frac{\delta_{ij}}{\pi_i Q_{ij}}$ where $\delta_{ij} = \delta_{ji}$ is chosen such that $\alpha_2(i,j) \leqslant 1$, $\forall i,j$. Then

$$\alpha_2(i,j)\pi_i Q_{ij} = \delta_{ij} = \delta_{ji} = \frac{\delta_{ji}}{\pi_j Q_{ji}} \pi_j Q_{ji} = \alpha_2(j,i)\pi_j Q_{ji}$$