Homework 2 Solution

1. Embedded Chain

Solution.

The exponential clock parameter for state i is the total outgoing rate from that state, given by $|q_{ii}|$ (absolute value of the diagonal entry in Q).

$$\begin{aligned} &\text{State 1:} \quad \lambda_1 = |-10| = 10 \\ &\text{State 2:} \quad \lambda_2 = |-5| = 5 \\ &\text{State 3:} \quad \lambda_3 = |-5| = 5 \end{aligned}$$

Calculations:

$$P_{ij} = \frac{q_{ij}}{\lambda_i} \quad \text{for } i \neq j, \quad \text{and } P_{ii} = 0.$$

- **State 1** ($\lambda_1 = 10$):

$$P_{12} = \frac{5}{10} = 0.5, \quad P_{13} = \frac{5}{10} = 0.5$$

- **State 2** ($\lambda_2 = 5$):

$$P_{21} = \frac{2}{5} = 0.4, \quad P_{23} = \frac{3}{5} = 0.6$$

- **State 3** ($\lambda_3 = 5$):

$$P_{31} = \frac{4}{5} = 0.8, \quad P_{32} = \frac{1}{5} = 0.2$$

The transition probability matrix of the embedded chain is

$$P = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0.4 & 0 & 0.6 \\ 0.8 & 0.2 & 0 \end{pmatrix}$$

The exponential clocks are exp(10), exp(5), and exp(5) respectively for state 1, 2, 3.

Homework 2: Generating Multivariate RV and Stochastic Process

Q2. Copula

Algorithm (Generating Variables from a Gaussian Copula):

Step 1: Generate the bivariate normal (W_1, W_2) using Cholesky method.

Step 2: Compute the CDF $U_i = \Phi(W_i)$.

Step 3: Return $X_i = F_i^{-1}(U_i), i = 1, \ldots, n.$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.stats import uniform
import math
import time
```

(a) Define a function to generates i.i.d. samples of (U_1,U_2) ~ C(x,y) (Gaussian Copula) given correlation coefficient rho(ρ)

```
# Generate normal via A-R method
def f(x): # target distribution : standard normal
    return (2*math.pi)**(-0.5)*math.e**(-0.5*x**2)
def g(x): # proposed distribution G
    return 0.5*math.e**(-abs(x))

c = 2*(2*math.pi)**(-0.5)*math.e**(0.5)

def gen_normal_AR(num):   # input is the number of variables to generate
    x = [] #output result
    while len(x) < num:
        X = np.random.default_rng().laplace(loc=0.0, scale=1.0, size=None) #generate a sample from proposed distribution (laplace
        U = uniform(0,1).rvs(1)[0]
        if U <= f(X)/(c *g(X)):
            x.append(X)
    return np.array(x)


# generate correlated uniform rvs from Gaussian Copula
def gen_CU_GaussCopula(rho,num=1):   # num: choose how many points to generate
    Sigma = np.array([[1,rho],[rho,1]])
    C = np.linalg.cholesky(Sigma)
    U = np.zeros((num,2))
    for i in range(num):
        Z = gen_normal_AR(2)
        W = np.matmul(C,Z.T)
        U[i]=[norm.cdf(W[0]),norm.cdf(W[1])]
    return np.array(U)

# print(gen_CU_GaussCopula(0.8,4))
```

(b) Estimate the Covariance $Cov(U1, U2)$ and plot the scatter plots for U_1,U_2 with $\rho \in \{-0.6, 0, 0.6\}$

$\text{Cov}(X, Y) = \frac{1}{n-1} \sum_i^n (x_i - \bar{x})(y_i - \bar{y})$

```
def estimateCov(U):
    MeanOfProd = np.average(U.T[0]*U.T[1]) # do the calculation in matrix
    ProdOfMean = np.prod(np.average(U, axis=0)) # take average along the column
    n = U.shape[0]
    Covariance = (MeanOfProd - ProdOfMean) * n / (n - 1)
    return Covariance


Rho = [-0.8,-0.6,-0.4,-0.2,0.2,0.4,0.6,0.8]

for rho in Rho:
    samples = gen_CU_GaussCopula(rho,10000) #generate U1,U2 for each iteration
    start_time = time.time()
    Cov1=np.cov(samples.T)[0,1].round(6)
    print("the estimated Covariance under correlation coefficient "+ str(rho)+ " is "+str(Cov1)+" (by np.cov())")
    print('with time elapsed:{:.6f} seconds'.format(time.time() - start_time))
    start_time = time.time()
    Cov2=estimateCov(samples).round(6)
    print("the estimated Covariance under correlation coefficient "+ str(rho)+ " is "+str(Cov2)+" (by estimateCov())")
    print('with time elapsed:{:.6f} seconds'.format(time.time() - start_time))
```

```
the estimated Covariance under correlation coefficient -0.8 is -0.065276 (by np.cov())
with time elapsed:0.001920 seconds
the estimated Covariance under correlation coefficient -0.8 is -0.065276 (by estimateCov())
with time elapsed:0.001212 seconds
the estimated Covariance under correlation coefficient -0.6 is -0.04917 (by np.cov())
with time elapsed:0.001061 seconds
the estimated Covariance under correlation coefficient -0.6 is -0.04917 (by estimateCov())
with time elapsed:0.000438 seconds
the estimated Covariance under correlation coefficient -0.4 is -0.032785 (by np.cov())
with time elapsed:0.000629 seconds
the estimated Covariance under correlation coefficient -0.4 is -0.032785 (by estimateCov())
with time elapsed:0.000676 seconds
the estimated Covariance under correlation coefficient -0.2 is -0.016952 (by np.cov())
with time elapsed:0.000622 seconds
the estimated Covariance under correlation coefficient -0.2 is -0.016952 (by estimateCov())
with time elapsed:0.000604 seconds
the estimated Covariance under correlation coefficient 0.2 is 0.0162 (by np.cov())
with time elapsed:0.000767 seconds
```

```
the estimated Covariance under correlation coefficient 0.2 is 0.0162 (by estimateCov())
with time elapsed:0.000421 seconds
the estimated Covariance under correlation coefficient 0.4 is 0.032793 (by np.cov())
with time elapsed:0.000786 seconds
the estimated Covariance under correlation coefficient 0.4 is 0.032793 (by estimateCov())
with time elapsed:0.000765 seconds
the estimated Covariance under correlation coefficient 0.6 is 0.047765 (by np.cov())
with time elapsed:0.000635 seconds
the estimated Covariance under correlation coefficient 0.6 is 0.047765 (by estimateCov())
with time elapsed:0.000674 seconds
the estimated Covariance under correlation coefficient 0.8 is 0.066304 (by np.cov())
with time elapsed:0.000606 seconds
the estimated Covariance under correlation coefficient 0.8 is 0.066304 (by estimateCov())
with time elapsed:0.000539 seconds
```

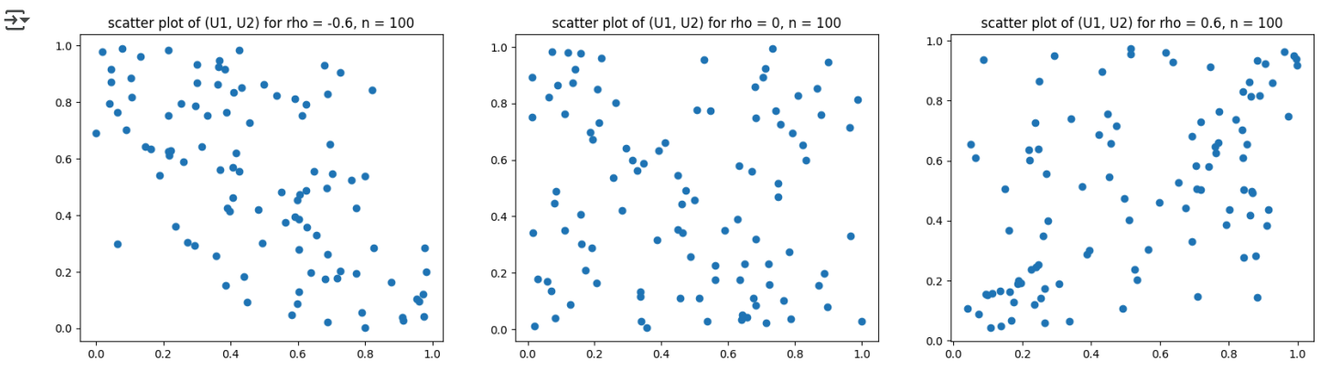
```
u1 = gen_CU_GaussCopula(-0.6,100)
u2 = gen_CU_GaussCopula(0,100)
u3 = gen_CU_GaussCopula(0.6,100)

plt.figure(figsize=[20,5])
plt.figure(1)

ax1=plt.subplot(131)
plt.scatter(u1.T[0], u1.T[1])
plt.title("scatter plot of (U1, U2) for rho = " + str(-0.6) + ", n = 100")

ax2=plt.subplot(132)
plt.scatter(u2.T[0], u2.T[1])
plt.title("scatter plot of (U1, U2) for rho = " + str(0) + ", n = 100")

ax3=plt.subplot(133)
plt.scatter(u3.T[0], u3.T[1])
plt.title("scatter plot of (U1, U2) for rho = " + str(0.6) + ", n = 100")
plt.show()
```



(c&d) Generate dependent exponential rvs (X_1, X_2) and estimate $Cov(X_1, X_2)$.

For the exponential with rate 1, the inverse CDF is $F^{-1}(u) = -\ln(1-u)$

```
def gen_Exp(rho,num):
    X = np.zeros((num,2))
    for i in range(num):
        U = gen_CU_GaussCopula(rho,1)
        X[i]= -1*np.log(np.array([1,1])-U)
    return X

# print(gen_Exp(0.8,6))
```

```
Rho = [-0.8,-0.6,-0.4,-0.2,0.2,0.4,0.6,0.8]
for rho in Rho:
    start_time = time.time()
    samples = gen_Exp(rho,10000) #generate U1,U2 for each iteration
    print('Spend {:.6f} seconds to producte the samples'.format(time.time() - start_time))
    Cov1=np.cov(samples.T)[0,1].round(6)
    print("the estimated Covariance for Exponential rvs under correlation coefficient "+ str(rho)+ " is "+str(Cov1)+" (by np.cov
    print('with time elapsed:{:.6f} seconds'.format(time.time() - start_time))
#     start_time = time.time()
#     Cov2=estimateCov(samples).round(6)
#     print("the estimated Covariance for Exponential rvs under correlation coefficient "+ str(rho)+ " is "+str(Cov2)+"(by estim
#     print('with time elapsed:{:.6f} seconds'.format(time.time() - start_time))
```

```
Spend 29.194216 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient -0.8 is -0.539766 (by np.cov())
with time elapsed:29.195145 seconds
Spend 28.419742 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient -0.6 is -0.425506 (by np.cov())
with time elapsed:28.420565 seconds
Spend 29.761439 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient -0.4 is -0.291599 (by np.cov())
with time elapsed:29.762520 seconds
Spend 28.479680 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient -0.2 is -0.17151 (by np.cov())
with time elapsed:28.480792 seconds
Spend 29.670917 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient 0.2 is 0.166055 (by np.cov())
with time elapsed:29.672970 seconds
Spend 27.834960 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient 0.4 is 0.354802 (by np.cov())
```

```
with time elapsed:27.835903 seconds
Spend 29.009094 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient 0.6 is 0.548397 (by np.cov())
with time elapsed:29.010112 seconds
Spend 29.447043 seconds to producte the samples
the estimated Covariance for Exponential rvs under correlation coefficient 0.8 is 0.803297 (by np.cov())
with time elapsed:29.448091 seconds
```

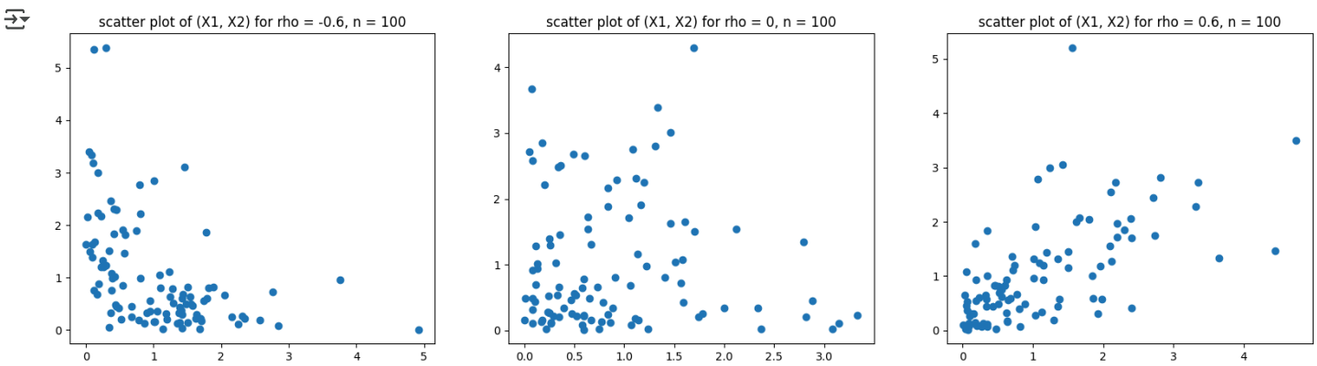
```
x1 = gen_Exp(-0.6,100)
x2 = gen_Exp(0,100)
x3 = gen_Exp(0.6,100)

plt.figure(figsize=[20,5])
plt.figure(1)

ax1=plt.subplot(131)
plt.scatter(x1.T[0], x1.T[1])
plt.title("scatter plot of (X1, X2) for rho = " + str(-0.6) + ", n = 100")

ax2=plt.subplot(132)
plt.scatter(x2.T[0], x2.T[1])
plt.title("scatter plot of (X1, X2) for rho = " + str(0) + ", n = 100")

ax3=plt.subplot(133)
plt.scatter(x3.T[0], x3.T[1])
plt.title("scatter plot of (X1, X2) for rho = " + str(0.6) + ", n = 100")
plt.show()
```



Q3. Poisson Processes

(i) Implement method 1 and generate a Poisson process with rate 5 on time interval [0, 2]. Verifying your implementation by showing that the simulated total number of arrivals follows a Poisson distribution with mean 5.

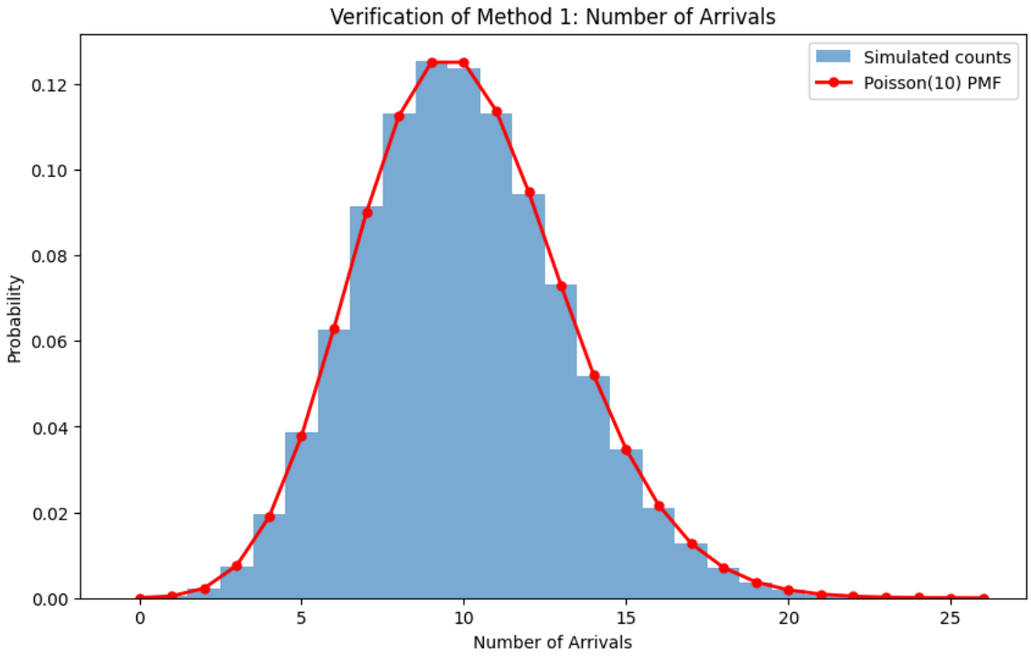
```
import numpy as np
import math
import matplotlib.pyplot as plt

def generate_method1(rate, T):
    S = []
    t = 0
    while True:
        tau = np.random.exponential(1 / rate)
        t += tau
        if t > T:
            break
        S.append(t)
    return S
```

```
# Simulate 100,000 times
# np.random.seed(0)
n_simulations = 100000
counts = [len(generate_method1(5, 2)) for _ in range(n_simulations)]

# Plot histogram for num of counts
plt.figure(figsize=(10, 6))
max_count = max(counts)
bins = np.arange(0, max_count + 1) - 0.5
plt.hist(counts, bins=bins, density=True, alpha=0.6, label='Simulated counts')

# Poisson PMF
rate = 10
x = np.arange(0, max_count + 1)
plt.plot(x, poisson.pmf(x, rate), 'ro-', lw=2, markersize=5, label='Poisson(10) PMF')
plt.xlabel('Number of Arrivals')
plt.ylabel('Probability')
plt.title('Verification of Method 1: Number of Arrivals')
plt.legend()
plt.show()
```



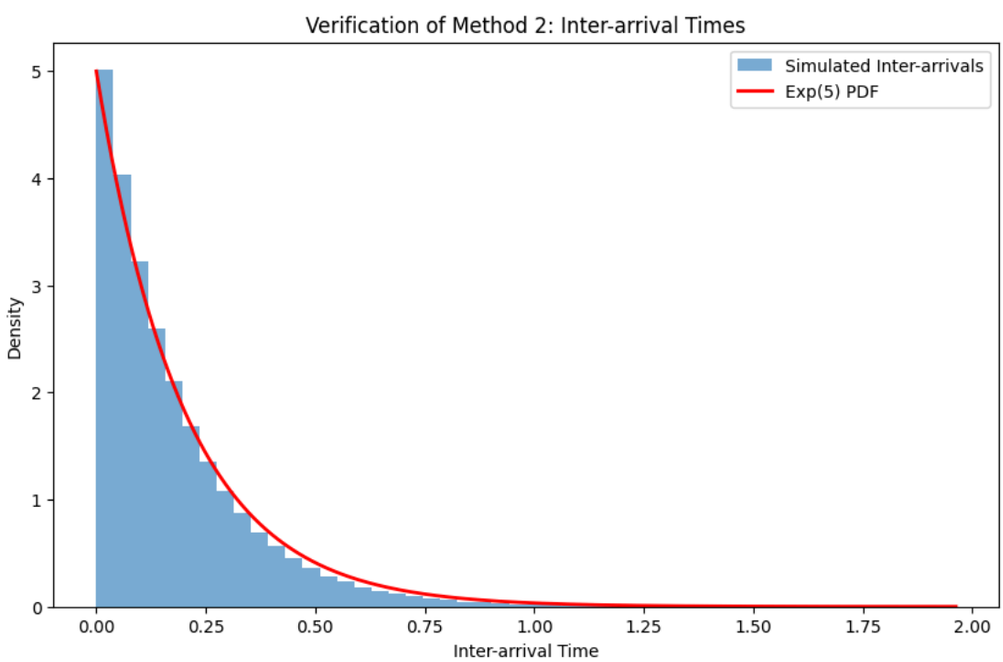
(ii) Implement method 2 and generate a Poisson process with rate 5 on time interval [0, 2]. Verifying your implementation by showing that the simulated inter-arrivals follows exponential distributions with rate 5.

```
def generate_method2(rate, T):
    N = np.random.poisson(rate * T)
    S = np.sort(np.random.uniform(0, T, N))
    inter_arrivals = np.diff(S) # collect inter-arrival times
    return S, inter_arrivals

# Simulate 100,000 times
# np.random.seed(0)
n_simulations = 100000
inter_arrivals = []
for _ in range(n_simulations):
    inter_arrivals.extend(generate_method2(5, 2)[1])

# Plot histogram for inter-arrival times
plt.figure(figsize=(10, 6))
plt.hist(inter_arrivals, bins=50, density=True, alpha=0.6, label='Simulated Inter-arrivals')

# Exponential PDF
x = np.linspace(0, max(inter_arrivals), 1000)
plt.plot(x, expon.pdf(x, scale=1/5), 'r-', lw=2, label='Exp(5) PDF')
plt.xlabel('Inter-arrival Time')
plt.ylabel('Density')
plt.title('Verification of Method 2: Inter-arrival Times')
plt.legend()
plt.show()
```



(iii) Do you think if method 2 can be extended to non-homogeneous Poisson process? If yes, please write down the pseudo code. If no, please explain your reason.

Answer: For a non-homogeneous Poisson process with rate function $\lambda(t)$ and time interval over $[0, T]$, the pseudo code is as follows:

step 1. Set a list $S = []$

step 2. Generate $N \sim \text{Poisson}(\Lambda(T))$ where $\Lambda(T)$ is the cumulative intensity function s.t.

$$\Lambda(T) = \int_0^T \lambda(t)\, dt$$

(In homogeneous case, we use $N \sim \text{Poisson}(\lambda * T)$*.)*

step 3. Generate uniform samples $U_1, U_2, \dots, U_N \sim \text{Uniform}(0, \Lambda(T))$

step 4. Sort uniform samples $U_{(1)} < U_{(2)} < \cdots < U_{(N)}$

step 5. Compute inverse cumulative intensity $t_i = \Lambda^{-1}(U_{(i)})$ and append t_i to S for $i = 1, 2, \dots, N$. *(In homogeneous case, we don't need to do the time transformation.)*

step 6. Returen S

Remark: except step 2 and 5, the pseudo code is almost the same as the method 2 psedo code in homogeneous case.