1. Text

   Description automatically generatedText

   Description automatically generated with medium confidence
2. Text

   Description automatically generated
3. Chart, scatter chart

   Description automatically generatedText

   Description automatically generated
4. Chart, scatter chart

   Description automatically generatedChart, line chart, scatter chart

   Description automatically generated
5. Sfa
6. Chart, line chart

   Description automatically generatedChart, histogram

   Description automatically generated

Code

import numpy as np

from matplotlib import pyplot as plt

import math

import random

def problem\_1():

print('\nPROBLEM 1')

# method 1

N = np.random.poisson(5)

X\_i = {i: 0 for i in range(N)}

for i in range(1,N):

u = random.random()

X\_i[i] = 20+math.floor(math.log(21\*u))

X\_1 = sum(X\_i.values())

print('method 1:',X\_1)

# method 2

t = 0

N, T = {}, []

while t < 1:

u = random.random()

t -= math.log(u)/5

T.append(t)

N[t] = math.ceil(19+21\*u)

X\_2 = list(N.values())[-1]

print('method 2:', X\_2)

def NH\_poisson\_lambda(t):

global rate

if t <= 5:

rate = t/5

elif 5 < t <= 10:

rate = 1 + 5\*(t-5)

return rate

def problem\_2():

print('\nPROBLEM 2')

# method 1

N, event\_times\_1 = 0, {}

while N <= 10:

t = 0

u\_1, u\_2 = random.random(), random.random()

t -= np.log(u\_1)

if NH\_poisson\_lambda(t) < u\_2:

if N == 0:

event\_times\_1[N] = t

else:

event\_times\_1[N] = event\_times\_1[N-1] + t

N += 1

print('method 1')

for t in event\_times\_1:

print(t, event\_times\_1[t])

# method 2

N, event\_times\_2, max\_rate = 0, {}, 26

while N <= 10:

t = 0

u\_1, u\_2 = random.random(), random.random()

t -= np.log(u\_1)/max\_rate

if u\_2 <= NH\_poisson\_lambda(t)/max\_rate:

if N == 0:

event\_times\_2[N] = t

else:

event\_times\_2[N] = event\_times\_2[N-1]+t

N += 1

print('\nmethod 2')

for t in event\_times\_2:

print(f'{t}:',event\_times\_2[t])

# method 3

event\_times\_3, N = {}, 0

while N <= 10:

t = 0

u\_1 = random.random()

x = -np.log(1-u\_1)

u\_2 = random.random()

if u\_2 <= NH\_poisson\_lambda(t+x)/26:

event\_times\_3[N] = x

N += 1

print('\nmethod 3')

for t in event\_times\_3:

print(f'{t}:', event\_times\_3[t])

def problem\_3():

print('\nPROBLEM 3')

""" ROSS 5.32 """

# disk info

r = 5

x\_center, y\_center = 0, 0

area\_disc = np.pi \* (r \*\* 2)

# Point process parameters (poisson intensity)

lambda\_poisson = 5

# Simulate Poisson point process

# Generate area of disc number of random points using poisson distribution

numbPoints = np.random.poisson(lambda\_poisson \* area\_disc)

# polar angular coordinates for points

theta = 2 \* np.pi \* np.random.uniform(0, 1, numbPoints)

# polar radial coordinates for points

rho = r \* np.sqrt(np.random.uniform(0, 1, numbPoints))

# Convert from polar to Cartesian coordinates

x\_generated = rho \* np.cos(theta)

y\_generated = rho \* np.sin(theta)

# Shift disk center

x\_generated = x\_generated + x\_center

y\_generated = y\_generated + y\_center

# Plot

fig = plt.figure()

plt.scatter(x\_generated, y\_generated, edgecolor='k', facecolor='none', alpha=0.5)

plt.xlabel("x")

plt.ylabel("y")

plt.axis('equal')

fig.suptitle('Poisson process on disc, r=5, lambda=1')

plt.savefig('poisson disc.png',dpi=300)

plt.close()

def problem\_4():

T = 100

rate = 1

X\_T = np.random.poisson(rate, size=T)

S = [np.sum(X\_T[0:i]) for i in range(T)]

X = np.linspace(0, T, T)

# Plot the graph

fig = plt.figure()

plt.step(X, S)

fig.suptitle('Single path renewal process, lambda\_1 = 1, lambda\_2 = 2, p = 0.6')

plt.ylim(0)

plt.xlim(0)

plt.xlabel('t')

plt.ylabel('N(t)')

plt.savefig('single path renewal process.png', dpi=300)

plt.close()

paths = {i: (None, None) for i in range(50)}

T = 100

rate = 1

multiples = 50

for k in range(multiples):

X\_T = np.random.poisson(rate, size=T)

S = [np.sum(X\_T[0:i]) for i in range(T)]

X = np.linspace(0, T, T)

paths[k] = (X,S)

fig = plt.figure()

fig.suptitle('50 paths renewal process lambda\_1 = 1, lambda\_2 = 2, p = 0.6')

for k in range(multiples):

plot, = plt.step(paths[k][0],paths[k][1])

plt.ylim(0)

plt.xlim(0)

plt.xlabel('t')

plt.ylabel('N(t)')

plt.savefig('50 paths renewal process.png', dpi=300)

plt.close()

multiples = 1000

iterates = {i: (None, None) for i in range(multiples)}

for k in range(multiples):

X\_T = np.random.poisson(rate, size=T)

S = [np.sum(X\_T[0:i]) for i in range(T)]

X = np.linspace(0, T, T)

iterates[k] = (X,S)

E\_t = {x: np.mean([iterates[k][1][x] for k in iterates]) for x in range(T)}

Var\_t = {x: np.var([iterates[k][1][x] for k in iterates]) for x in range(T)}

IDC\_t = {x: Var\_t[x]/E\_t[x] for x in range(T)}

print('IDC\_50:', IDC\_t[49], '\nIDC\_90:', IDC\_t[89])

COV = math.sqrt(sum([(E\_t[x])\*\*2 for x in range(T)]))/sum([E\_t[x] for x in range(T)])

print('COV:', COV)

def problem\_6():

T, lambda\_0, a, b = 100, 1, 0.8, 1.2

event\_times\_6, lambda\_times\_6, s, n = {}, {}, 0, 0

while n < T:

lambda\_s = lambda\_0 + np.sum([a\*b\*math.exp(-b\*(s-t)) for t in event\_times\_6])

u\_1 = random.random()

w = -np.log(u\_1/lambda\_s)

s += w

u\_2 = random.random()

if u\_2\*lambda\_s <= lambda\_s:

event\_times\_6[n] = s

lambda\_times\_6[n] = lambda\_s

n += 1

fig = plt.figure()

plt.step(event\_times\_6.keys(), event\_times\_6.values())

fig.suptitle("Single path hawkes process N(t)")

plt.ylim(0)

plt.xlim(0)

plt.xlabel('t')

plt.ylabel('N(t)')

plt.savefig('single path hawkes process N(t).png', dpi=300)

plt.close()

fig = plt.figure()

plt.step(event\_times\_6.keys(), lambda\_times\_6.values())

fig.suptitle("Single path hawkes process lambda(t)")

plt.ylim(0)

plt.xlim(0)

plt.xlabel('t')

plt.ylabel('lambda(t)')

plt.savefig('single path hawkes process lambda(t).png', dpi=300)

plt.close()

multiples = 1000

iterates = {i: None for i in range(multiples)}

for k in range(multiples):

s, n = 0, 0

event\_times, lambda\_times = {}, {}

while n < T:

lambda\_s = lambda\_0 + np.sum([a \* b \* math.exp(-b \* (s - t)) for t in event\_times])

u\_1 = random.random()

w = -np.log(u\_1 / lambda\_s)

s += w

u\_2 = random.random()

if u\_2 \* lambda\_s <= lambda\_s:

event\_times[n] = s

lambda\_times[n] = lambda\_s

n += 1

iterates[k] = event\_times

E\_t\_2 = {x: np.mean([iterates[k][x] for k in iterates]) for x in range(T)}

Var\_t\_2 = {x: np.var([iterates[k][x] for k in iterates]) for x in range(T)}

IDC\_t = {x: Var\_t\_2[x]/E\_t\_2[x] for x in range(T)}

print('IDC\_50:', IDC\_t[49], '\nIDC\_90:', IDC\_t[89])

# executable code

problem\_1()

problem\_2()

problem\_3()

problem\_4()

problem\_6()