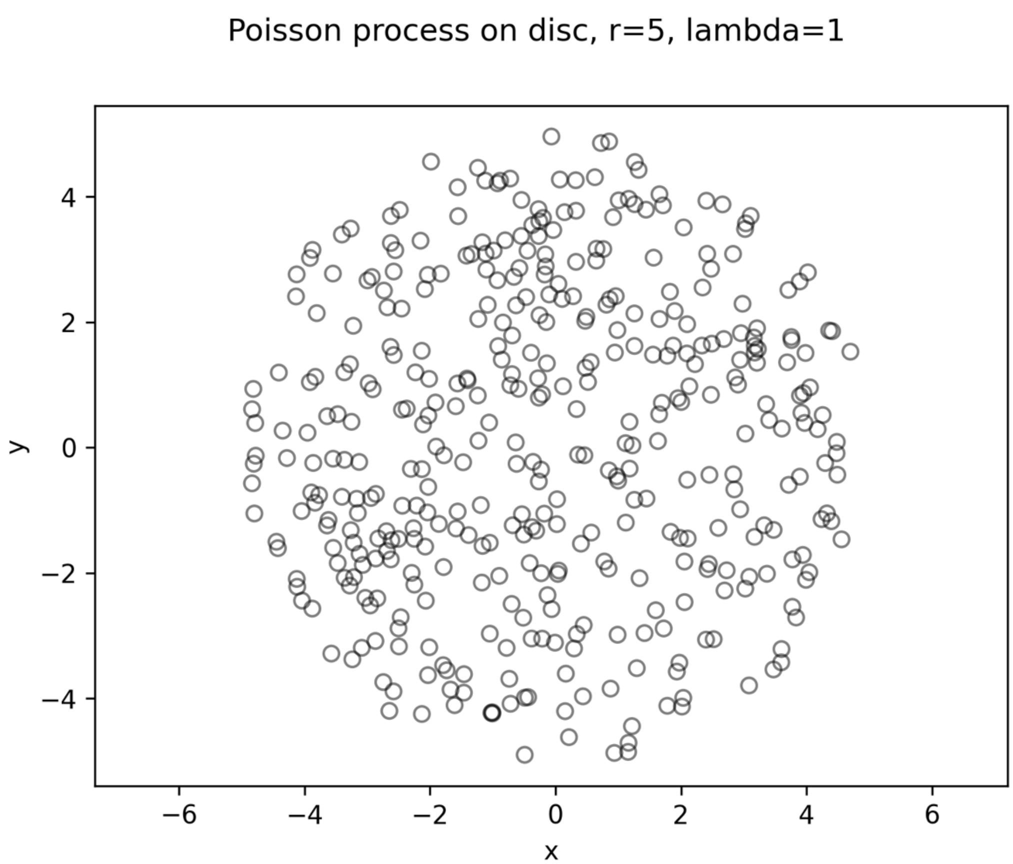
1.

```
PROBLEM 1
method 1: 172
method 2: 23
```

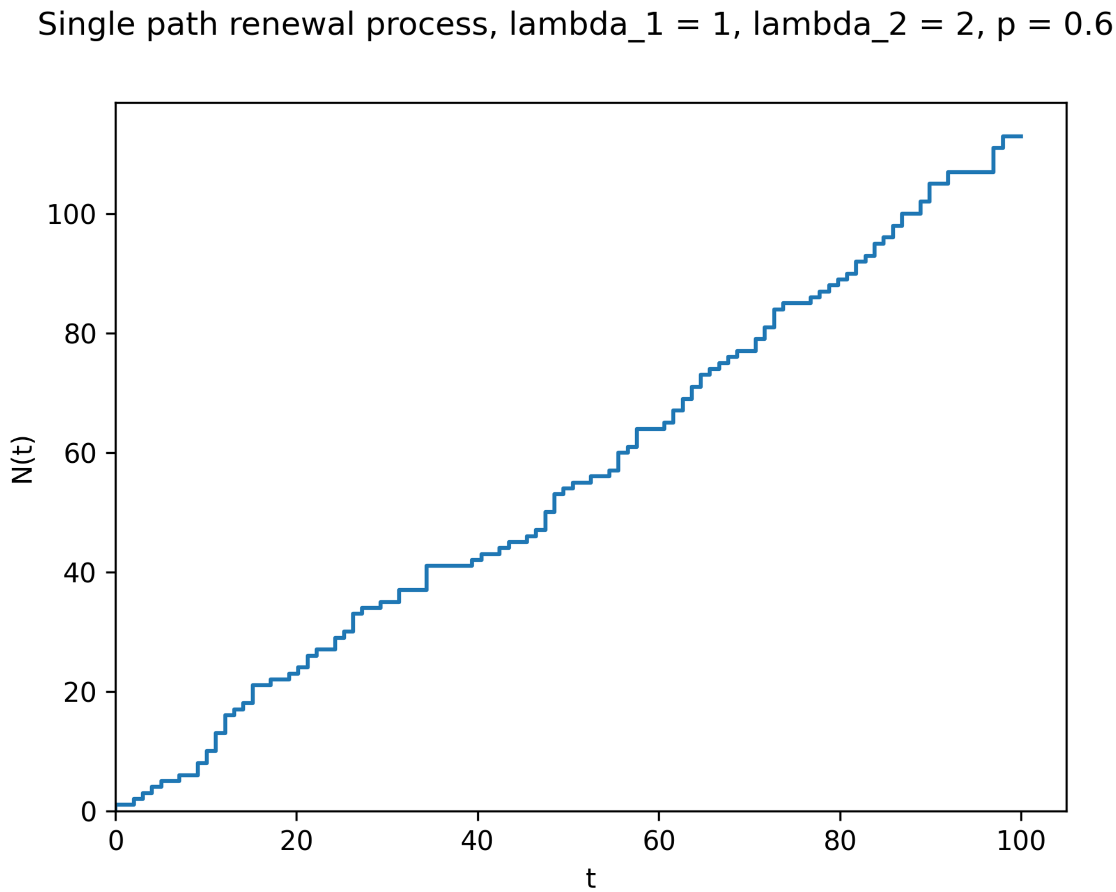
2.

method 1	method 2	method 3
0 0.46727164507769564	0: 0.00757195635120876	0: 5.799558262348227
1 1.6463823561730029	1: 0.09753210814769715	1: 3.1687068582564395
2 2.4570561898319143	2: 0.14718037905721892	2: 7.974090145433699
3 3.4769618410411103	3: 0.1642000893479767	3: 2.879918157946683
4 3.523777461900119	4: 0.32214851619460294	4: 2.989004088085324
5 4.810985165802574	5: 0.4630315987584813	5: 1.586140144367834
6 6.54913785788047	6: 0.5689215854685943	6: 7.015053801283589
7 7.106846094668085	7: 0.6030777438739323	7: 5.117271671432222
8 7.850050346683684	8: 0.6157769589651209	8: 0.20508074957909916
9 8.283458235338454	9: 0.7145950277074427	9: 3.056398976272386
10 9.060113525137526	10: 0.7591121134015201	10: 1.4029829294118739

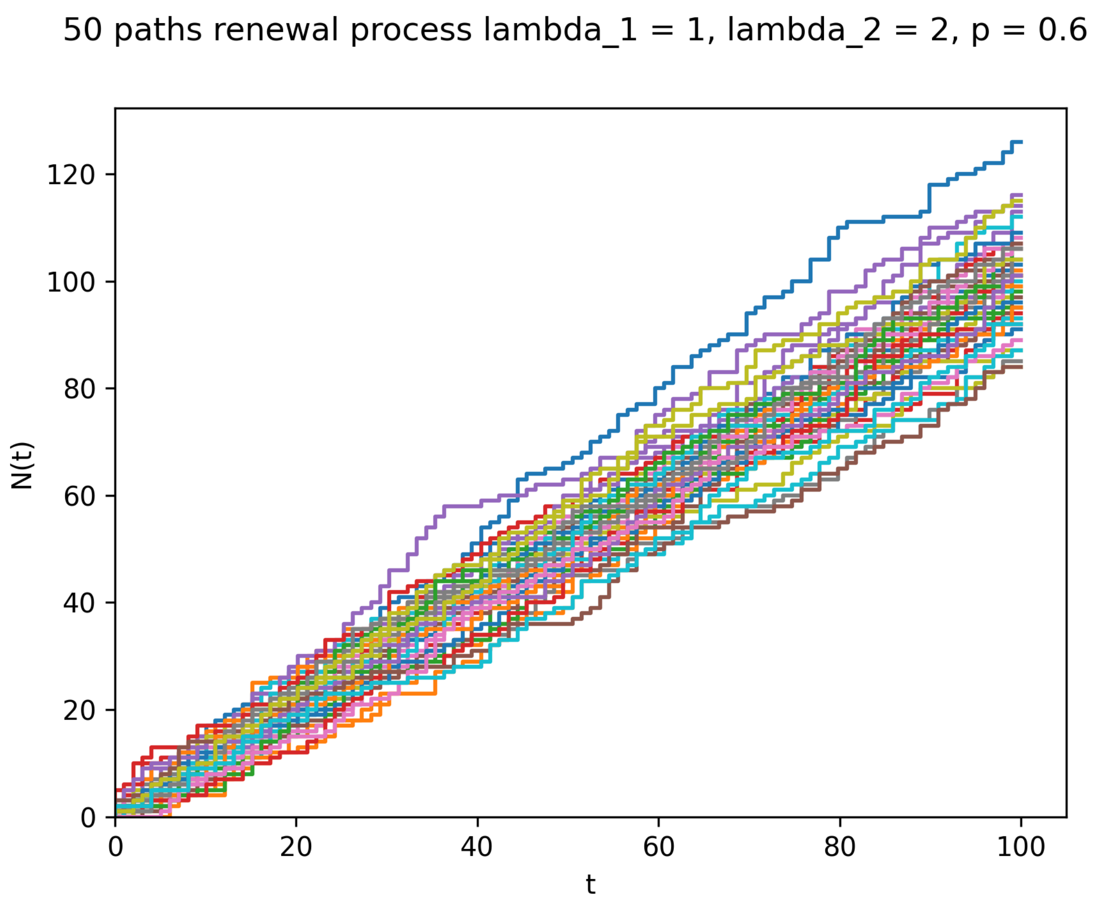
3.



4.

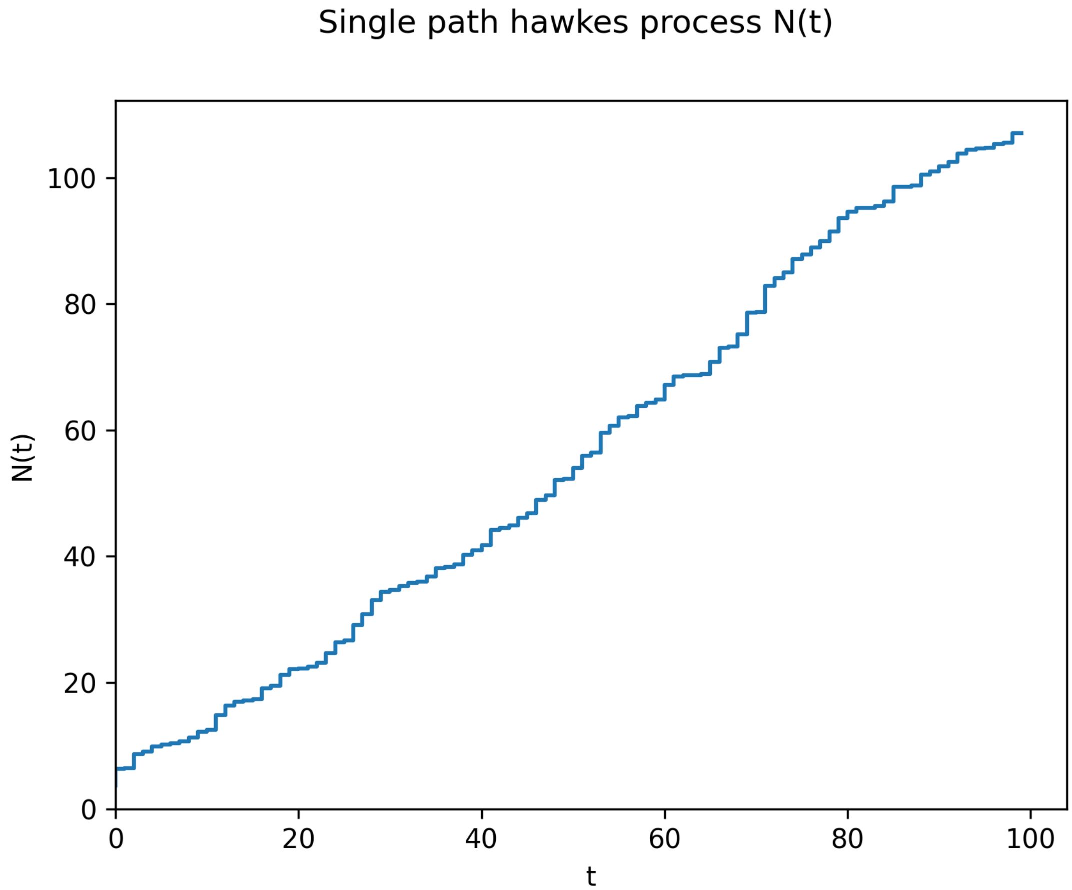


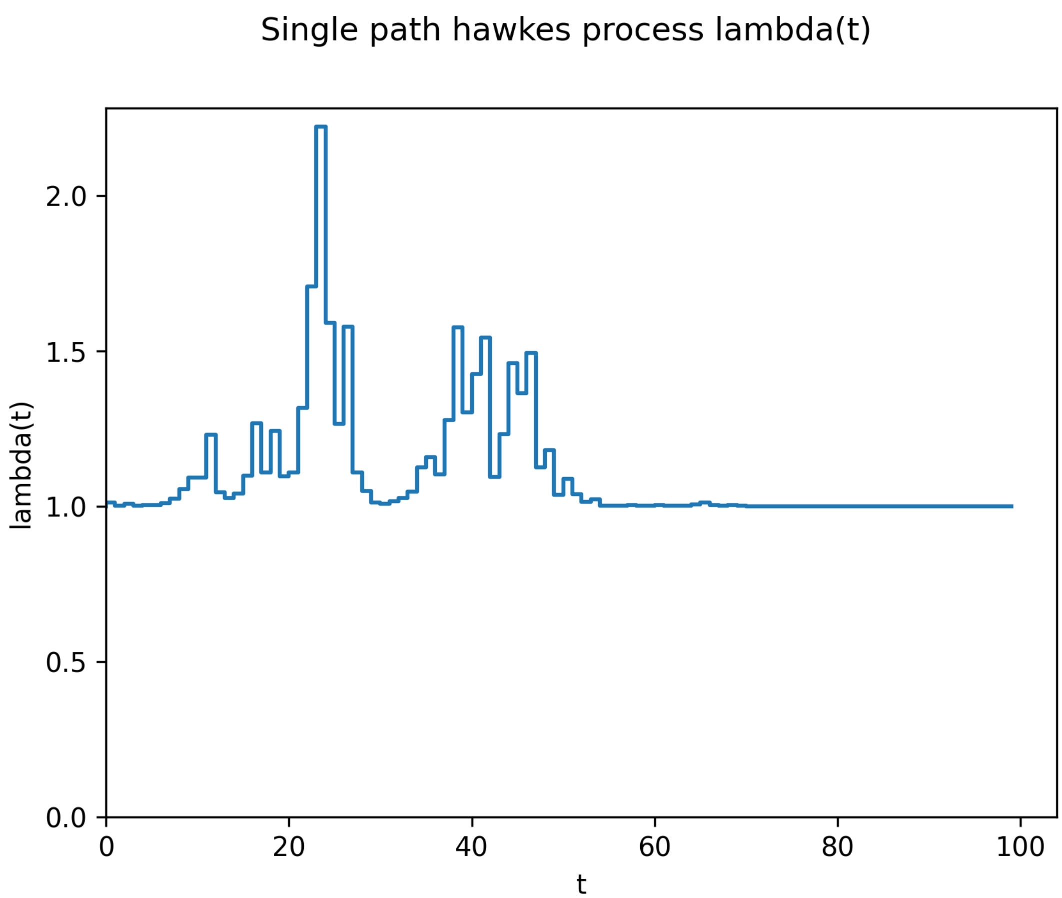
Single path renewal process, lambda_1 = 1, lambda_2 = 2, p = 0.6
N(t)
t
0
20
40
60
80
100



50 paths renewal process lambda_1 = 1, lambda_2 = 2, p = 0.6
N(t)
t
0
20
40
60
80
100
120

6.


Single path hawkes process N(t)
N(t)
0
20
40
60
80
100
0
20
40
60
80
100
t


Single path hawkes process lambda(t)
lambda(t)
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
t

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()
```