Project 2 : Student number 491531 and 477789

Christian Lehre & Emil Myhre

Problem 1 : Insurance claims

Let $X(t)$ denote number of claims received by an insurance company during a time interval $\tau \in [0, t]$. One can assume that $X(t)$ is a Poisson process with continuous time index $\tau \geq 0$, where τ denotes days from January 1st, 0:00.00.

a) Homogeneous Poisson process

In this task we let the intensity, or rate of the process, be fixed at $\lambda(t) = 3$. The probability that there are more than 325 claims before May 1st (120 days) is

$$P(X(120)_H > 325) = 1 - P(X(120)_H \leq 325) = 1 - F(325) = 0.9671,$$

where $P(X(120)_H \leq 325) = F(325)$ is the cumulative distribution function of the Poisson process at time $t = 120$ days.

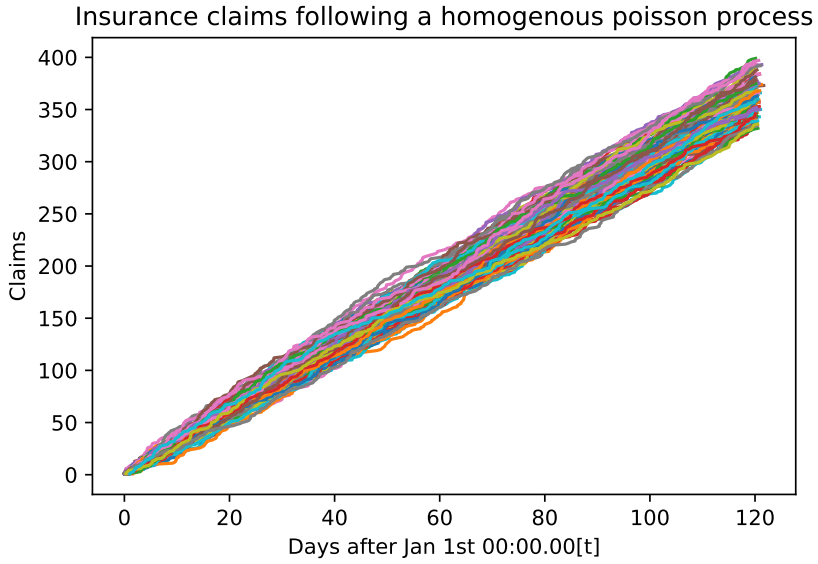


Figure 1: 100 realizations of the homogeneous poisson process with constant $\lambda(t) = 3$

We then simulated the process 1000 times, and counted the amount of claims we received in order to verify the theoretical probability of receiving

more than 325 claims. We ended up with receiving more than 325 claims in 976 out of the 1000 simulations, which is pretty much what we expect based on the probability. Visualization shown in figure 3.

b) Inhomogeneous Poisson process

In this section we simulate the number of insurance claims as an inhomogeneous poisson process with intensity $\lambda(t) = 2 + \cos\left(\frac{2\pi(t)}{365}\right)$.
The probability of more than 325 claims before May 1st (120 days) was calculated using the corresponding cumulative distribution in which we calculated numerically.

$$P(X(120)_I > 325) = 1 - P(X(120)_I \leq 325) = 1 - F(325) = 0.0235,$$

where $P(X(120)_I \leq 325) = F(325)$ is the cumulative distribution function of the Poisson process at time $t = 120$ days.

Thus, the homogeneous process gives a higher probability of receiving more than 325 claims before May 1st.

By comparing figure 1 and 2, one observe that the rate in the inhomogeneous process changes with time, so the graphs appear less linear.

As well as with the homogeneous process, we made 1000 simulations in order to verify the calculated probability. In this case, we received more than 325 claims in 24 out of the 1000 simulations, which corresponds nicely to the theoretical values. Visualization shown in figure 3.

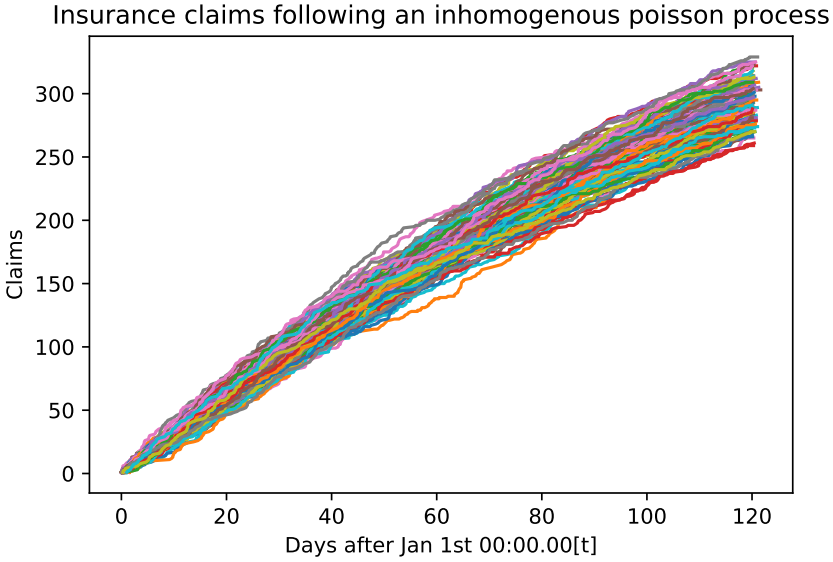


Figure 2: 100 realizations of the inhomogeneous poisson process with $\lambda(t) = 2 + \cos\left(\frac{2\pi(t)}{365}\right)$

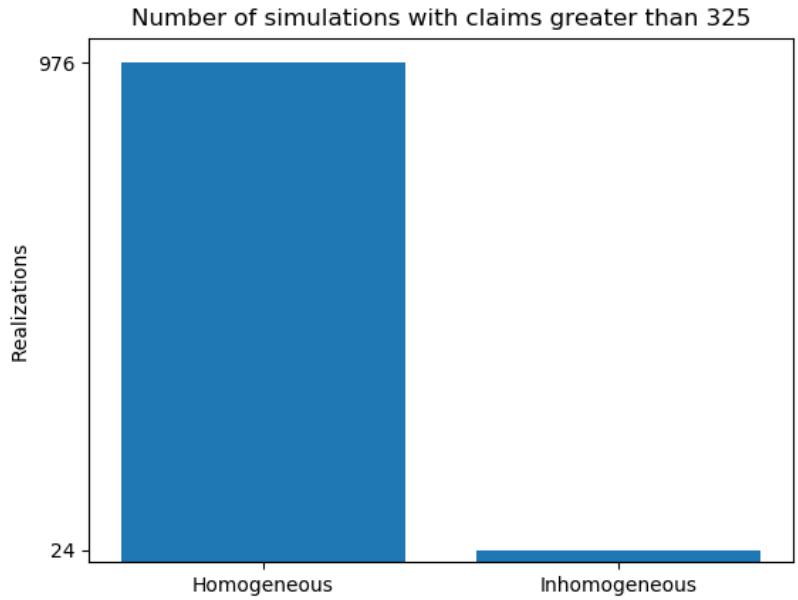


Figure 3: Number of simulations resulting in claims greater than 325 before May 1st

c) Log-Gaussian distribution

The expected value for the log-gaussian distribution, $E[C_i] = E[e^{Y_i}]$, is calculated by the fact that $Y_i \sim N(\mu, \sigma^2)$.

$$E[C_i] = E[e^{Y_i}] = \int_{\mathbb{R}} e^y \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2} dy$$

$$= \frac{1}{\sqrt{2\pi}\sigma} \int_{\mathbb{R}} e^{z\sigma+\mu-\frac{1}{2}z^2} \sigma dz = e^{\mu} \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{\frac{\sigma^2}{2}} e^{-\frac{1}{2}(z-\sigma)^2} dz$$

$$= e^{\mu+\frac{\sigma^2}{2}} \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-\frac{1}{2}(z-\sigma)^2} dz = e^{\mu+\frac{\sigma^2}{2}},$$

using the substitution $z = \frac{y-\mu}{\sigma}$, completing the square in the exponent of the integrand and the gaussian integral $\int_{\mathbb{R}} e^{(z-\sigma)^2} dz = \sqrt{2\pi}$.

The variance is calculated as

$$Var[C_i] = E[C_i^2] - E[C_i]^2 = e^{\mu+\sigma^2}(e^{\sigma^2} - 1),$$

using $E[C_i^2] = E[e^{2Y_i}] = e^{2(\mu+\sigma^2)}$.

The total claim amount is denoted by $Z = \sum_{i=1}^{X(t)} C_i$. One can calculate the expected value and variance in total claim amounts using the law of total expectation and -variance, respectively:

$$E[Z] = E\left[\sum_{i=1}^{X(t)} C_i\right] = E\left[E\left[\sum_{i=1}^{X(t)} C_i | X(t)\right]\right]$$

$$= E\left[\sum_{i=1}^{X(t)} E[C_i]\right] = E[X(t)E[C_i]] = E[X(t)e^{\mu+\frac{1}{2}\sigma^2}]$$

$$E[Z] = E[X(t)]e^{\mu+\frac{1}{2}\sigma^2} = \lambda t e^{\mu+\frac{1}{2}\sigma^2},$$

where we have used $E[C_i]$ from above and $E[X(t)] = \lambda t$ for the homogeneous poisson process $X(t)$.

$$Var[Z] = Var\left[\sum_{i=1}^{X(t)}\right] = E\left[Var\left[\sum_{i=1}^{X(t)} C_i | X(t)\right]\right] + Var\left[E\left[\sum_{i=1}^{X(t)} C_i | X(t)\right]\right]$$

$$= E\left[\sum_{i=1}^{X(t)} Var[C_i]\right] + Var\left[\sum_{i=1}^{X(t)} E[C_i]\right]$$

$$= E[X(T)Var[C_i]] + Var[X(t)E[C_i]] = E[X(t)]Var[C_i] + Var[X(t)]E[C_i]$$

$$E[Z] = \lambda t e^{2(\mu+\sigma^2)},$$

	E(.)	Var(.)
C_i	0.22	0.09
Z	80.33	48.72

Table 1: Theoretical variance and expected values for individual and total claim amounts

using $E[C_i]$ and $Var[C_i]$ from above, and that $E[X(t)] = Var[X(t)] = \lambda t$ for the homogeneous poisson process $X(t)$.
The numerical values are listed in the table 1.

Simulations of the claim amounts was simulated for both the homogeneous process with $\lambda = 3$ and the inhomogeneous process with
$\lambda(t) = 3 + cos(\frac{2\pi t}{365})$. for 100 realizations. Visuals of the simulations are depicted in figure (4-5) below.
As one can see from figure 4 below, the expected value for the total claim amounts fits well to the theoretical value listed in table 1. The observed variance was calculated to $Var[Z] = 50.1$ MNOK, which also within reach of the theoretical value in table 1.

We observe that the expected total claim amounts is less for the inhomogenous than the homogenous process (see figures 4-5). This is expected, as the rate of the inhomogenous process is always less than or equal to the rate of the homogenous process.

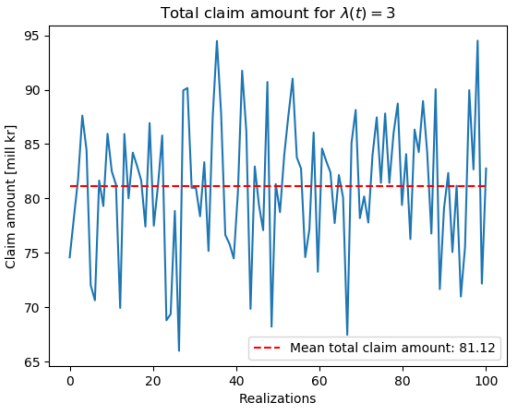


Figure 4: 100 realizations of the inhomogeneous poisson process with $\lambda(t) = 3$

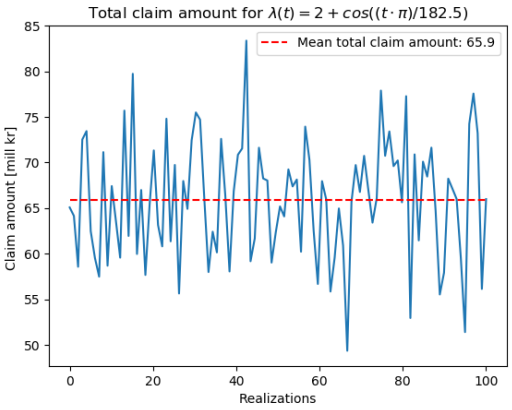


Figure 5: 100 realizations of the inhomogeneous poisson process with $\lambda(t) = 2 + \cos\left(\frac{2\pi(t)}{365}\right)$

Problem 2 : Server jobs

In this problem we are considering the following birth-death process:
A computer server can handle at most 32 jobs at a time. Arrivals of jobs are defined to be Poisson distributed with rate $\lambda = 25$ per hour. The duration of each job is exponentially distributed, with rate parameter $\mu = 1$ per hour. Any arriving job when the servers capacity is full, is forwarded to another server. Thus the current amount of jobs on the server is bounded, $X(t) \in \{0, 1, ..., 32\}$.

a)

By using long-term equality of rates in and out of states we can calculate the long-term probabilities

$$\pi_k = \prod_{i=1}^{k-1} \frac{\lambda^i}{\mu^{i+1}} \frac{\pi_0}{i!}$$

We observe that all long-term probabilities are determined by π_0. We know

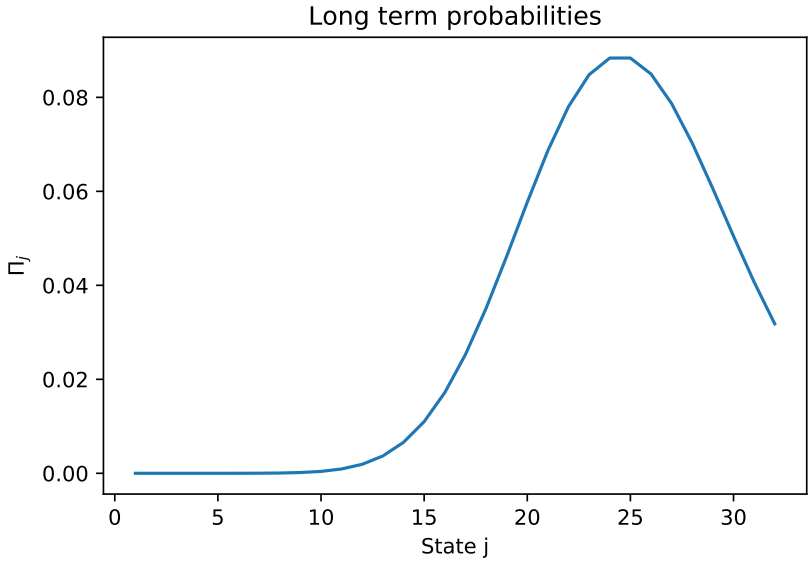


Figure 6: Equilibrium probabilities P(X(t) = n), n = 0,1,...,32.

all probabilities must sum to 1, which yields the following expression for π_0:

$$\pi_0 = \frac{1}{1 + \sum_{k=1}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1} \cdot i!}}$$

From figure 6 we conclude that the long term probability is greatest for 25 parallel jobs. This makes sense intuitively, because from the Poisson distributed arrivals, we expect 25 new jobs per hour. Simultaneously, we expect 25 jobs to be finished per hour, when each job has an exponential living time of 1 hour. Thus at 25 jobs, we have statistical equilibrium between births and deaths. We expect the process to oscillate around this value.

b)

In this task we are simulating the birth-death process for 168 hours. In figure 7 we have displayed 3 simulations, showing that the process is oscillating around the equilibrium at ≈ 25. As the process reached 25 parallel jobs, we considered it to be in equilibrium. Based on these simulations, the end of the transient part of the process was estimated at ≈ 2.42 hours, illustrated in figure 7.

Figure 7: 3 simulations of the server job process

In figure 8 we have compared the theoretical long term probabilities with realizations from 100 simulations, to get as accurate results as possible. Clearly, the simulations correspond pretty well with the analytical calculations, which verifies the probabilities derived in a).

Finally, we also counted the amount of jobs being forwarded to another

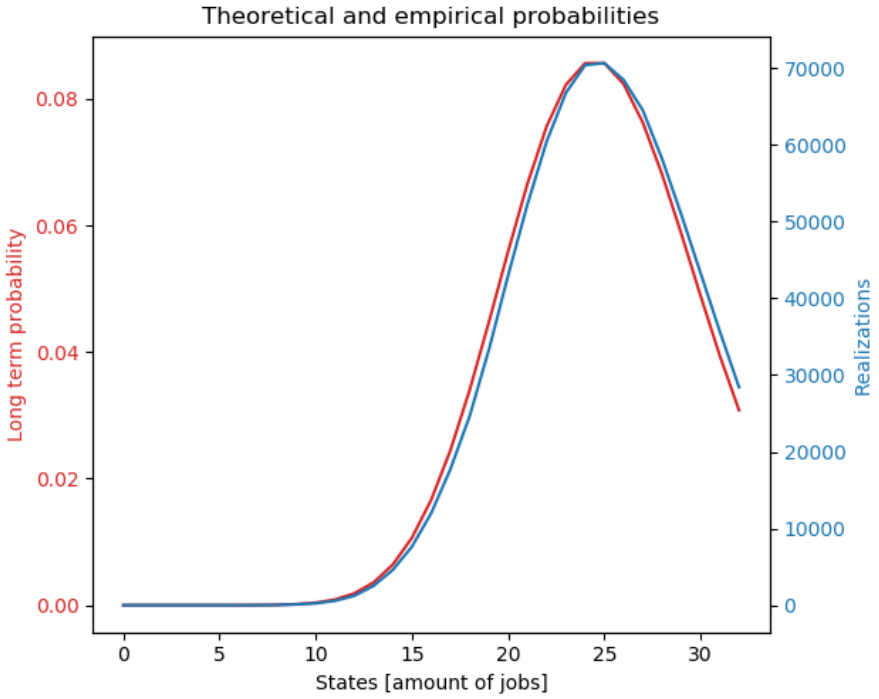


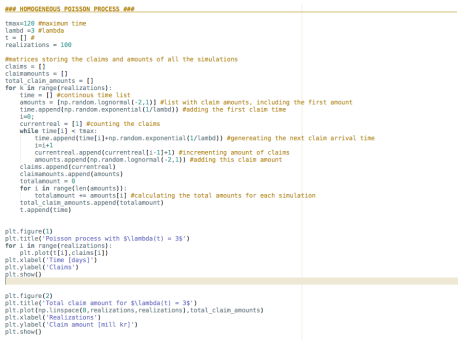
Figure 8: Comparison between the theoretical long term probabilities and the sum of realizations from 100 simulations

server in our simulations, and from 100 simulations the process averaged ≈ 0.75 forwarded jobs per hour.

Appendix

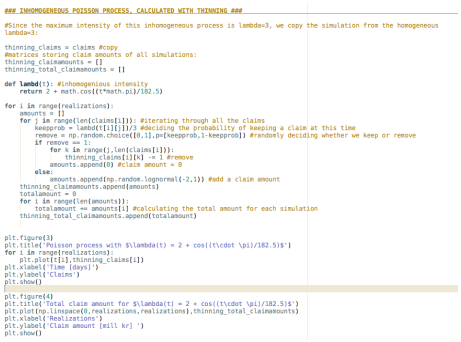
Source code problem 1

Source code for problem 1 are contained in the screen dumps below



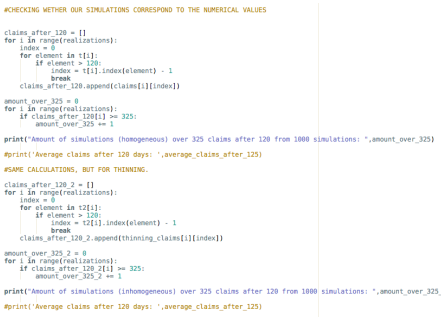
```
### HOMOGENEOUS POISSON PROCESS ###

tmax=120 #maximum time
lambd =3 #lambda
t = [] #
realizations = 100

#matrices storing the claims and amounts of all the simulations
claims = []
claimamounts = []
total_claim_amounts = []
for k in range(realizations):
    time = [] #continous time list
    amounts = [np.random.lognormal(-2,1)] #list with claim amounts, including the first amount
    time.append(np.random.exponential(1/lambd)) #adding the first claim time
    i=0;
    currentreal = [1] #counting the claims
    while time[i] < tmax:
        time.append(time[i]+np.random.exponential(1/lambd)) #genereating the next claim arrival time
        i=i+1
        currentreal.append(currentreal[i-1]+1) #incrementing amount of claims
        amounts.append(np.random.lognormal(-2,1)) #adding this claim amount
    claims.append(currentreal)
    claimamounts.append(amounts)
    totalamount = 0
    for i in range(len(amounts)):
        totalamount += amounts[i] #calculating the total amounts for each simulation
    total_claim_amounts.append(totalamount)
    t.append(time)


plt.figure(1)
plt.title('Poisson process with $\lambda(t) = 3$')
for i in range(realizations):
    plt.plot(t[i],claims[i])
plt.xlabel('Time [days]')
plt.ylabel('Claims')
plt.show()


plt.figure(2)
plt.title('Total claim amount for $\lambda(t) = 3$')
plt.plot(np.linspace(0,realizations,realizations),total_claim_amounts)
plt.xlabel('Realizations')
plt.ylabel('Claim amount [mill kr]')
plt.show()
```

Figure 9: Code for simulating the homogeneous Poisson process and its claim amounts



```
### INHOMOGENEOUS POISSON PROCESS, CALCULATED WITH THINNING ###

#Since the maximum intensity of this inhomogeneous process is lambda=3, we copy the simulation from the homogeneous
lambda=3:

thinning_claims = claims #copy
#matrices storing claim amounts of all simulations:
thinning_claimamounts = []
thinning_total_claimamounts = []

def lambd(t): #inhomogenious intensity
    return 2 + math.cos((t*math.pi)/182.5)

for i in range(realizations):
    amounts = []
    for j in range(len(claims[i])): #iterating through all the claims
        keepprob = lambd(t[i][j])/3 #deciding the probability of keeping a claim at this time
        remove = np.random.choice([0,1],p=[keepprob,1-keepprob]) #randomly deciding whether we keep or remove
        if remove == 1:
            for k in range(j,len(claims[i])):
                thinning_claims[i][k] -= 1 #remove
            amounts.append(0) #claim amount = 0
        else:
            amounts.append(np.random.lognormal(-2,1)) #add a claim amount
    thinning_claimamounts.append(amounts)
    totalamount = 0
    for i in range(len(amounts)):
        totalamount += amounts[i] #calculating the total amount for each simulation
    thinning_total_claimamounts.append(totalamount)


plt.figure(3)
plt.title('Poisson process with $\lambda(t) = 2 + cos((t\cdot \pi)/182.5)$')
for i in range(realizations):
    plt.plot(t[i],thinning_claims[i])
plt.xlabel('Time [days]')
plt.ylabel('Claims')
plt.show()

plt.figure(4)
plt.title('Total claim amount for $\lambda(t) = 2 + cos((t\cdot \pi)/182.5)$')
plt.plot(np.linspace(0,realizations,realizations),thinning_total_claimamounts)
plt.xlabel('Realizations')
plt.ylabel('Claim amount [mill kr] ')
plt.show()
```

Figure 10: Code for simulating the inhomogeneous Poisson process and its claim amounts



```
#CHECKING WETHER OUR SIMULATIONS CORRESPOND TO THE NUMERICAL VALUES


claims_after_120 = []
for i in range(realizations):
    index = 0
    for element in t[i]:
        if element > 120:
            index = t[i].index(element) - 1
            break
    claims_after_120.append(claims[i][index])

amount_over_325 = 0
for i in range(realizations):
    if claims_after_120[i] >= 325:
        amount_over_325 += 1

print("Amount of simulations (homogeneous) over 325 claims after 120 from 1000 simulations: ",amount_over_325)

#print('Average claims after 120 days: ',average_claims_after_125)

#SAME CALCULATIONS, BUT FOR THINNING.

claims_after_120_2 = []
for i in range(realizations):
    index = 0
    for element in t2[i]:
        if element > 120:
            index = t2[i].index(element) - 1
            break
    claims_after_120_2.append(thinning_claims[i][index])

amount_over_325_2 = 0
for i in range(realizations):
    if claims_after_120_2[i] >= 325:
        amount_over_325_2 += 1

print("Amount of simulations (inhomogeneous) over 325 claims after 120 from 1000 simulations: ",amount_over_325_2)

#print('Average claims after 120 days: ',average_claims_after_125)
```

Figure 11: Source code to verify the theoretical probabilities in 1)

Source code problem 2

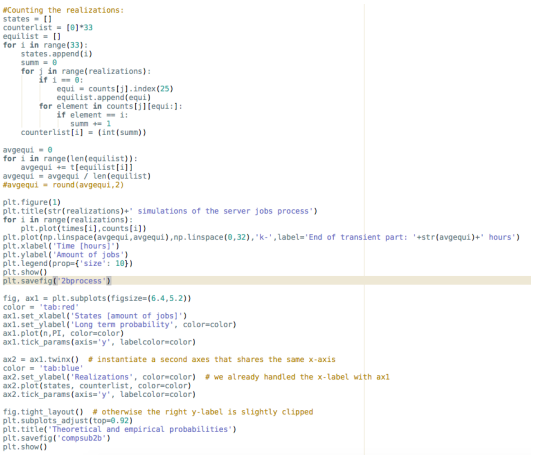
Source code for solving problem 2 are shown in the screen dumps below

```
lambd = 25 #Birth rate
mu = 1 #Death rate

###### CALCULATING THE EQUILIBRIUM PROBABILITIES ########

N = 32 # Number of states
summ = 0

#Calculating PI_0
for i in range(33):
    summ += (lambd/mu)**(i)/(math.factorial(i))
pi0 = 1/summ

#Function that calculates PI_n using PI_0
def pi(n,pi0,lambd,mu):
    return (pi0/math.factorial(n))*(lambd/mu)**n


n = np.linspace(0,32,33)
PI = [pi(r,pi0,lambd,mu) for r in n]

plt.figure(3)
plt.title('Long term probabilities')
plt.xlabel('State j')
plt.ylabel('$\Pi_{j}$')
plt.plot(n,PI)
plt.savefig('longTermProbs.pdf')
plt.show()
```

Figure 12: Source code for calculating the long term probabilities in 2a)

```
####### SIMULATION ##########

realizations = 100
time = 24*7 #hours
times = [] #matrix to store all the time lists from the simulations
counts = [] #matrix to store all realizations for the simulations
overload_counts = []
for i in range(realizations):
    count = [0] #list which is counting the amount of jobs at every time
    t = [0] #continious time list
    i = 0 #iterator
    deathtimes = [] #list keeping track of the time of death of all the current jobs
    overload_count = 0 #counting how many jobs are being forwarded to another server
    while t[i] < time: #simulating a whole week (24*7 hours)
        if count[i] == 0: #when we have no jobs
            if len(count) == 1:
                nextjob = t[i] + np.random.exponential(1/lambd) #adding the first job arrival time (from poisson)
            t.append(nextjob) #adding the time of the birth to the timelist
            count.append(1) #updating the current amount of jobs
            deathtimes.append(nextjob + np.random.exponential(mu)) #death time for this job (birth time + living time)
            i += 1
        if count[i] == 32: #if the server is full of jobs
            nextjob = t[i] + np.random.exponential(1/lambd)
            nextdeath = min(deathtimes) #when do we have the next death
            index = deathtimes.index(nextdeath) #finding the first death
            if nextjob < nextdeath: #checking whether a birth arrives before a death
                overload_count += 1 #counting jobs forwarded
                t.append(nextjob)
                count.append(count[i])
                i = i+1
            if nextdeath < nextjob: #death befor birth
                count.append(count[i]-1) #decreasing current jobs
                t.append(nextdeath)
                del deathtimes[index] #deleting the corresponding deathtime
                i = i+1
        else:
            if count[i] > count[i-1]:
                nextjob = t[i] + np.random.exponential(1/lambd)#generating a new job, if we previously added a job
            nextdeath = min(deathtimes)
            index = deathtimes.index(nextdeath)
            if nextjob < nextdeath: #next incident is a birth
                t.append(nextjob)
                count.append(count[i] + 1)
                deathtimes.append(nextjob + np.random.exponential(mu)) #creating a death time for this job
                i += 1
            if nextdeath < nextjob: #next incident is a death
                count.append(count[i]-1)
                t.append(nextdeath)
                del deathtimes[index]
                i = i+1
    counts.append(count)
    times.append(t)
    overload_counts.append(overload_count)
```

Figure 13: Source code for simulating 2b)



```
#Counting the realizations:
states = []
counterlist = [0]*33
equilist = []
for i in range(33):
    states.append(i)
    summ = 0
    for j in range(realizations):
        if i == 0:
            equi = counts[j].index(25)
            equilist.append(equi)
        for element in counts[j][equi:]:
            if element == i:
                summ += 1
    counterlist[i] = (int(summ))

avgequi = 0
for i in range(len(equilist)):
    avgequi += t[equilist[i]]
avgequi = avgequi / len(equilist)
#avgequi = round(avgequi,2)

plt.figure(1)
plt.title(str(realizations)+' simulations of the server jobs process')
for i in range(realizations):
    plt.plot(times[i],counts[i])
plt.plot(np.linspace(avgequi,avgequi),np.linspace(0,32),'k-',label='End of transient part: '+str(avgequi)+' hours')
plt.xlabel('Time [hours]')
plt.ylabel('Amount of jobs')
plt.legend(prop={'size': 10})
plt.show()
plt.savefig('2bprocess')

fig, ax1 = plt.subplots(figsize=(6.4,5.2))
color = 'tab:red'
ax1.set_xlabel('States [amount of jobs]')
ax1.set_ylabel('Long term probability', color=color)
ax1.plot(n,PI, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
color = 'tab:blue'
ax2.set_ylabel('Realizations', color=color)  # we already handled the x-label with ax1
ax2.plot(states, counterlist, color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.subplots_adjust(top=0.92)
plt.title('Theoretical and empirical probabilities')
plt.savefig('compsub2b')
plt.show()
```

Figure 14: Source code for plotting 2b)