Appendix A - Code Filip

Filip (s183565)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sc

def LCG(M,a,c,x0,L):
   numbers = np.zeros(L)

   for i in range(L):
       numbers[i] = (a*x0+c) %M
       x0 = numbers[i]
   return numbers



M,a,c,x0,L = 16,5,1,3,10000

randomList = LCG(M,a,c,x0,L)
randoms = randomList/M



# plt.figure()
# plt.scatter(range(L),randomList)
# #plt.ylim(0,20)



# plt.figure()
# plt.hist(randoms,bins=10,rwidth=0.95)
# plt.figure()
# plt.scatter(range(L),randoms)


###CHI SQUARE TEST
def chisq_test(randoms):

    n_classes = 10
    n_observed = (np.histogram(randoms,n_classes)[0])

    n_expected = L/n_classes

    if n_expected < 5:
        print("Warning: low number of n_expected")

    #test statistic
    T = sum(((n_observed - n_expected)**2)/n_expected)


    p = 1-sc.chi2.cdf(T , df=9)

    #test
    #print(sc.chisquare(n_observed)[1])

    print("Chisq test statistic: ",T,". p-value: ",p)
    if p<0.05:
        print("Hypothesis is rejected as p<0.05 (no significant difference)")
    else:
        print("A significant difference is detected as p>0.05)")
    return (T,p)

chisq_test(randoms)

###KOLMOGOROV SMIRNOV TEST
def ks_test(randoms):
    randomsSorted = np.sort(randoms)
    expected =np.linspace(0, 1, L)

    plt.figure()
    plt.step(randomsSorted,np.linspace(0, 1, L),where='post')
    plt.plot(expected,np.linspace(0, 1, L))

    T_ks = max(abs(randomsSorted-expected))
    #p_ks =
    return T_ks #,p_ks

    #test
    #return sc.kstest(randomsSorted, "uniform")
    #return sc.kstest(randomsSorted, expected)


ks_test(randoms)



# mdn = np.median(randoms)
# n1 = len(randoms[randoms > mdn])
# n2 = len(randoms[randoms < mdn])










def run_test1(randoms):
    mdn = np.median(randoms)
    n1 = len(randoms[randoms > mdn])
    n2 = len(randoms[randoms < mdn])
    Ra = 0
    Rb = 0
    for i in range(1,L):
        if randoms[i] > mdn and randoms[i-1] < mdn:
            Ra += 1
        elif randoms[i] < mdn and randoms[i-1] > mdn:
            Rb += 1
    if randoms[0] > mdn: Ra +=1
    else: Rb+=1
    T_r1 = Ra + Rb

    mn = (2*(n1*n2)/(n1+n2) + 1)
    sd = (2*(n1*n2*(2*n1*n2-n1-n2))/((n1+n2)**2*n1+n2-1) )

    p_r1 = 1 - sc.norm.cdf(T_r1,mn,sd)
    return T_r1,p_r1

run_test1(randoms)


"""
@author: filip
"""

import numpy as np
import random
import matplotlib.pyplot as plt
import scipy.stats as scs
import timeit

import numpy as np
p1,p2,p3=0.03,0.37,0.85

gm1= np.random.geometric(p1,10000)
gm2= np.random.geometric(p2,10000)
gm3=np.random.geometric(p3,10000)


#plt.plot(np.sort(gm3))
#Histograms
plt.figure()
plt.subplot(1, 3, 1)
plt.hist(gm1,density=True)
plt.title("p=0.03")
plt.subplot(1, 3, 2)
plt.hist(gm2,density=True,color='red')
plt.title("p=0.37")
plt.subplot(1, 3, 3)
plt.hist(gm3,density=True,color='green')
plt.title("p=0.85")
plt.tight_layout()






p = np.array([7/48,5/48,1/8,1/16,1/4,5/16])
pa = np.cumsum(p)
k=6

def stats(Y,name):
    plot =plt.figure()
    a =
 ↪  plt.hist(Y,bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='white',color='blue')
    plt.title(name)
    plt.xlabel("X")
    plt.ylabel("Density")
    #plot.savefig(r"C:\Users\Spil\OneDrive\Documents\DTU_kandidat\2.
     ↪  semester\Stochastic simulation\1_2ali_hist")
    print("Mean: ",np.mean(Y))
    print("Std: ", np.std(Y))
    print(a[0])
    print("chisq: ", scs.chisquare(a[0],p)[0:2])
    print()
    return


stats(gm1,'gm1')
stats(gm2,'gm2')
stats(gm3,'gm3')



plt.figure()
plt.plot(pa)


k = 6
"""CRUDE METHOD"""
#setup
```

```
#X=-1

def crude():
    U = np.random.random(1)
    for i in range(6):
        if U < pa[i]:
            return i+1
    return "error"
print(crude())


#For a list (more efficient)
def ncrude(nn):
    ss = np.zeros(nn)
    p = np.array([7/48,5/48,1/8,1/16,1/4,5/16])
    pa = np.cumsum(p)
    U = np.random.random(nn)
    for j in range(nn):
        for i in range(6):
            if U[j] < pa[i]:
                ss[j] = i+1
                break
    return ss
ncrude(10)



"""REJECTION METHOD"""
#setup

#method
def rejection():
    q = 1/k
    c = max(p)/q
    while(1):
        U1 = np.random.random(1)
        Y = int(k*U1)+1
        U2 = np.random.random(1)
        if U2 <= p[Y-1]/(c*q):
            Xr = Y
            return Xr
    return "error"
print(rejection())


#For a list (more efficient)
def nrejection(nn):
    q = 1/k
    c = max(p)/q
    ss = np.zeros(nn)
    U1 = np.random.random(nn)
    U2 = np.random.random(nn)
    for j in range(nn):
        while(ss[j]==0):
            Y = int(k*U1[j])+1
            if U2[j] <= p[Y-1]/(c*q):
                ss[j] = Y
                break
            else:
                U1[j],U2[j] = np.random.random(2)
    return ss
nrejection(10)




"""Alias METHOD"""
#Setup for alias using pseudo-code from slides
L=np.ones(k)*range(k)
F=k*p
G = np.where(F>=1)[0]
S = np.where(F<=1)[0]
while np.size(S)>0:
    i = G[0]
    j = S[0]
    L[j] = i+1
    F[i] = F[i] - (1 - F[j])

    if F[i] < 1-0.000001:
        G = np.delete(G,0)
        S = np.append(S,i)
    S = np.delete(S,0)


def alias():
    UA1 = np.random.random(1)
    UA2 = np.random.random(1)
    I = int(np.floor(k*UA1) + 1)
    #print(UA1,UA2,I)
    if UA2 <= F[I-1]:
        return I
    else:
        return L[I-1]

print(alias())

#For a list (more efficient)
def nalias(nn):
    ss = np.zeros(nn)
    UA1 = np.random.random(nn)
    UA2 = np.random.random(nn)
    for j in range(nn):
        I = int(np.floor(k*UA1[j]) + 1)
        #print(UA1,UA2,I)
        if UA2[j] <= F[I-1]:
            ss[j] = I
        else:
            ss[j] = L[I-1]
    return ss

nalias(10000)



"""Plots and stats for methods"""
crX = np.zeros(10000)
start1 = timeit.default_timer()
for i in range(10000):
    crX[i] = crude()
stop1 = timeit.default_timer()
stats(crX,'crude')
#
```

```
reX = np.zeros(10000)
start2 = timeit.default_timer()
for i in range(10000):
    reX[i] = rejection()
stop2 = timeit.default_timer()
stats(reX,'rejection')


alX = np.zeros(10000)
start3 = timeit.default_timer()
for i in range(10000):
    alX[i] = alias()
stop3 = timeit.default_timer()
stats(alX,'alias')


print(stop1-start1)
print(stop2-start2)
print(stop3-start3)



#efficient list methods
start1 = timeit.default_timer()
tt1 = ncrude(10000)
stop1 = timeit.default_timer()

start2 = timeit.default_timer()
tt2 = nrejection(10000)
stop2 = timeit.default_timer()

start3 = timeit.default_timer()
tt2 = nalias(10000)
stop3 = timeit.default_timer()
print(stop1-start1)
print(stop2-start2)
print(stop3-start3)
```

```
plt.hist([crX],bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='red',color='red',alpha=1
plt.hist([reX],bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='blue',color='blue',alpha
plt.hist([alX],bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='green',color='green',alp

plt.hist([crX],bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='red',color='red',alpha=0
plt.hist([reX],bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='blue',color='blue',alpha
plt.hist([alX],bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='green',color='green',alp


#plt.hist(reX,bins=6,density=1,rwidth=1,edgecolor='white',color='blue')

#aaaa =
 ↪  plt.hist(reX,bins=[1,2,3,4,5,6,7],density=1,rwidth=1,edgecolor='white',color='blue')


#plt.plot(- np.log(np.random.random(100))/1)


import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
import scipy.stats as stats
import random
import math

from scipy.stats import expon
import matplotlib.pyplot as plt

from scipy.stats import t


n = 10
#%% Exponential distribution

lmd = 1.3

#Simulation
exp_sim = -np.log(np.random.random(n))/lmd
```

```
#Analytical
exp_a = stats.expon.rvs(scale=lmd, size=n)

#plots
plt.figure()
plt.hist(exp_sim,   color="blue",               density=True,rwidth=2)
plt.hist(exp_a,     color="red",    alpha=0.7,  density=True,rwidth=2)




#%% Normal dist

u1 = np.random.rand(n)
u2 = np.random.rand(n)

x1 = np.sqrt(-2*np.log(u1) ) * np.cos(2*np.pi*u2)
x2 = np.sqrt(-2*np.log(u1) ) * np.sin(2*np.pi*u2)

xx = np.concatenate([x1,x2])



plt.figure()
plt.hist(xx,density=True,color="blue")
nd = stats.norm.rvs(size=n)
plt.hist(nd, color='black',density=True,alpha=0.5)


#%% 3





ci = np.zeros((100,4))

#confidence lvel
CL = 0.999
#deg of freedom
DF = n-1
```

```
#this z-value might be wrong
z = np.abs(t.ppf( (1-CL)/2,DF ) )

for i in range(100):
    u1 = np.random.rand(n)
    u2 = np.random.rand(n)
    x1 = np.sqrt(-2*np.log(u1) ) * np.cos(2*np.pi*u2)
    #x1 = np.random.normal(0,1,n)
    m = np.mean(x1)
    s = np.std(x1)

    pm = z * s/np.sqrt(n)

    ci[i,:] = [m+pm, m-pm,m,s]


plt.plot(ci[1:100,:2])
plt.plot(ci[1:100,2],color="gray")

plt.ylim(-2,2)


# -*- coding: utf-8 -*-
"""
Created on Tue Jun  6 10:15:13 2023

@author: filip
"""

import numpy as np
import scipy.stats as stats
from scipy.stats import t
import matplotlib.pyplot as plt

n=10000

m = 10      #service units
mst = 8     #mean service time
mtbc = 1    #mean time between customers
```

```
#service time
def service_time(meanST,typ):
    if typ == "exp":
        return stats.expon.rvs(scale=meanST,size=1)
    if typ == "constant":
        return meanST
    if typ == "pareto":
        k=1.05
        return pareto(k,1)
    if typ == "normal":
        nd = stats.norm.rvs(mst,2)
        if nd > 0:  return nd
        else:   return 0
    #if typ == "unif":
    #    return random.random



#arrival
def arrival(meanTBC,typ):
    if typ == "erlang":
        return stats.erlang.rvs(a=1,scale=meanTBC,size=1) #basically
         ↪  exponential
                                        #alternative
                                         ↪  stats.erlang.rvs(a=4,scale=meanTBC/4,size=1)
    if typ == "pois":
        return stats.expon.rvs(scale=meanTBC,size=1)
    if typ == "hyperexp":
        p1,l1,l2 = 0.8,0.8333,5.0
        return hyperexp(p1, l1, l2)


arrival(mtbc,"pois")


def hyperexp(p1, l1, l2):
    U = np.random.random(1)
    if U<=p1:
        return stats.expon.rvs(scale=1/l1,size=1)
    else:
        return stats.expon.rvs(scale=1/l2,size=1)


def pareto(k,N):
    U = np.random.random(N)
    return mst/(k/(k-1)) *(U)**(-1/k)


def sim(N,a_typ,s_typ):
    blocked = 0
    service_unit = np.zeros(m)
    arriveTime = 0
    arrivals = np.zeros(N)

    for i in range(N):
        arrivals[i] = arrival(mtbc,a_typ)
        arriveTime += arrivals[i]
        serviceTime = service_time(mst,s_typ)
        if np.min(service_unit) <= arriveTime:
            for s in range(m):
                if service_unit[s] <= arriveTime:
                    service_unit[s] = arriveTime + serviceTime
                    break
        else:
            blocked +=1
    pBlock = blocked/N
    return pBlock


print(sim(n,"pois","exp"))

#Analytical solution of pBlock:
A = mtbc*mst
B = (A**m / np.math.factorial(m)) / sum([A**i/np.math.factorial(i) for i
 ↪  in range(m+1)])


def calculateCI(blocks):
    N = len(blocks)
    ci= np.zeros(2)

    #confidence lvel
    CL = 0.975
    #deg of freedom
    DF = N-1

    #this z-value might be wrong
    z = np.abs(t.ppf( (1-CL)/2,DF ) )

    m = np.mean(blocks)
    s = np.std(blocks,ddof=1)

    pm = z * s/np.sqrt(N)

    ci = [m-pm, m+pm]
    return(ci)


sims = 10
Blocks = np.zeros(sims)
for i in range(sims):
    Blocks[i] = sim(n,"pois","exp")


expB_CI = (calculateCI(Blocks))

np.mean(Blocks)


plt.scatter(range(10),Blocks)
plt.axhline(expB_CI[0],color='red')
plt.axhline(expB_CI[1],color='green')


#%% Question 2



sim(n,"pois","exp")
sim(n,"erlang","exp")
sim(n,"hyperexp","exp")




#%% Question 3

#arrival, service

sim(n,"hyperexp","exp")




#%% q4


#arrival distributions confidence intervals

sims = 10
B_erl = np.zeros(sims)
for i in range(sims):
    B_erl[i] = sim(n,"erlang","exp")

sims = 10
B_hyp = np.zeros(sims)
for i in range(sims):
    B_hyp[i] = sim(n,"hyperexp","exp")



print(calculateCI(Blocks))
print(calculateCI(B_erl))
print(calculateCI(B_hyp))

plt.scatter([1,1],calculateCI(Blocks))
plt.scatter([2,2],calculateCI(B_erl))
plt.scatter([3,3],calculateCI(B_hyp))
plt.axhline(B)
```

```
#service time distributions confidence intervals
sims = 10
B_const = np.zeros(sims)
for i in range(sims):
    B_const[i] = sim(n,"pois","constant")

sims = 10
B_par = np.zeros(sims)
for i in range(sims):
    B_par[i] = sim(n,"pois","pareto")



print(calculateCI(Blocks))
print(calculateCI(B_const))
print(calculateCI(B_par))

"""
@author: filip
"""

import numpy as np
import scipy.stats as stats
from scipy.stats import t
import matplotlib.pyplot as plt



n=100




def CI(samples):
    N = len(samples)
    CI = np.zeros(3)
    CL = 0.95
    DF = N-1
    z = np.abs(t.ppf( (1-CL)/2,DF ) )
    m = np.mean(samples)
    s = np.std(samples,ddof=1)
    pm = z * s/np.sqrt(N)

    CI = [m-pm,m, m+pm]
    return CI


def crude_estimator(N):
    I = 100
    X=np.zeros(N)
    for i in range(N):
        x = np.random.random(100) #100 = estimation precision
        X[i] = sum(np.exp(x))/N
    return  X


t1 = CI(crude_estimator(n))






def antithetic_var(N):
    Y = np.zeros(N)
    for i in range(N):
        U = np.random.random(N)
        ev = np.exp(U)
        Y[i] = sum((ev + np.exp(1)/ev)/2)/N
    return Y

t2 = CI(antithetic_var(n))

def control_var(N):
    U = np.random.random(N)
    X = np.exp(U)
    np.mean(U)

    co = np.mean(U*np.exp(U)) - np.mean(U)*np.mean(np.exp(U))
    #va = sum(U - np.mean(U) )**2 / (N-1)
    va = np.var(U)
    c = -co/va
    Z = X + c*(U-(1/2))
    return Z

t3 = CI(control_var(n))



def stratified(N):
    I = 10
    I,N = 10,n
    U = np.zeros((I,N))
    for i in range(I):
        U[i,:] = np.random.random(N)

    W = np.zeros(I)
    W = sum(np.exp(U[:,nn]/N + nn/N) for nn in range(N))/N
    return W


t4 = CI(stratified(n))





import timeit

s1 = timeit.default_timer()
t1 = crude_estimator(n)
e1 = timeit.default_timer()

s2 = timeit.default_timer()
t2 = antithetic_var(n)
e2 = timeit.default_timer()

s3 = timeit.default_timer()
t3 = control_var(n)
e3 = timeit.default_timer()

s4 = timeit.default_timer()
t4 = stratified(n)
```

```
e4 = timeit.default_timer()

print(e1-s1)
print(e2-s2)
print(e3-s3)
print(e4-s4)








#%%exercise 5

n=100

m = 10      #service units
mst = 8     #mean service time
mtbc = 1    #mean time between customers

#service time
def service_time(meanST,typ):
    if typ == "exp":
        return stats.expon.rvs(scale=meanST,size=1)
    if typ == "constant":
        return meanST
    if typ == "pareto":
        k=1.05
        return pareto(k,1)
    if typ == "normal":
        nd = stats.norm.rvs(mst,2)
        if nd > 0:  return nd
        else:   return 0
    #if typ == "unif":
    #    return random.random

#arrival
def arrival(meanTBC,typ):
    if typ == "erlang":
        return stats.erlang.rvs(a=1,scale=meanTBC,size=1) #basically
         ↪  exponential
                                        #alternative
                                         ↪  stats.erlang.rvs(a=4,scale=meanTBC/4,size=1)
    if typ == "pois":
        return stats.expon.rvs(scale=meanTBC,size=1)
    if typ == "hyperexp":
        p1,l1,l2 = 0.8,0.8333,5.0
        return hyperexp(p1, l1, l2)
    # if typ == "hyperexpNew":
    #     p1,l1,l2 = 0.8,0.8333,5.0
    #     return hyperexpNew(p1, l1, l2)

def hyperexp(p1, l1, l2):
    U = np.random.random(1)
    if U<=p1:
        return stats.expon.rvs(scale=1/l1,size=1)
    else:
        return stats.expon.rvs(scale=1/l2,size=1)


# def hyperexpNew(p1, l1, l2):
#     U = np.random.random(1)
#     return
 ↪  stats.expon.ppf(U,scale=1/l1)*0.8+stats.expon.ppf(U,scale=1/l1)*0.2



def pareto(k,N):
    U = np.random.random(N)
    return mst/(k/(k-1)) *(U)**(-1/k)

def sim(N,a_typ,s_typ):

    blocked = 0
    service_unit = np.zeros(m)
    arriveTime = 0
    arrivals = np.zeros(N)

    for i in range(N):
        arrivals[i] = arrival(mtbc,a_typ)
        arriveTime += arrivals[i]
        serviceTime = service_time(mst,s_typ)
        if np.min(service_unit) <= arriveTime:
            for s in range(m):
                if service_unit[s] <= arriveTime:
                    service_unit[s] = arriveTime + serviceTime
                    break
        else:
            blocked +=1
    pBlock = blocked/N
    return pBlock, np.mean(arrivals)


def control_var5(N):
    X_a = np.zeros(N)
    X_b = np.zeros(N)
    for i in range(N):
        X_b[i],X_a[i] = sim(10000,"pois","exp")
    co = np.cov(X_a,X_b)[0,1]

    vaXA = np.var(X_a)
    vaXB = np.var(X_b)
    c = -co/vaXA
    Z = X_b + c*(X_a-mtbc)
    vaZ = np.var(Z)
    return CI(Z),CI(X_b),vaZ,vaXB


e5Res = control_var5(10)



#%% Question 6
# Some of the functions are updated to include predefined random numbers
#NOTE that only "exp" service time is supported (also constant)



np.random.seed(seed=100)

#service time
```

```
def service_time(meanST,typ,rand):
    if typ == "exp":
        return stats.expon.ppf(scale=meanST,q=rand)
    if typ == "constant":
        return meanST
    if typ == "pareto":
        k=1.05
        return pareto(k,1)
    if typ == "normal":
        nd = stats.norm.rvs(mst,2)
        if nd > 0:  return nd
        else:   return 0
    #if typ == "unif":
    #    return random.random

#arrival
def arrival(meanTBC,typ,rand):
    if typ == "pois":
        return stats.expon.ppf(scale=meanTBC,q=rand)
    if typ == "hyperexp":
        p1,l1,l2 = 0.8,0.8333,5.0
        return hyperexp(p1, l1, l2,rand)
    # if typ == "hyperexpNew":
    #     p1,l1,l2 = 0.8,0.8333,5.0
    #     return hyperexpNew(p1, l1, l2)

def hyperexp(p1, l1, l2,rand):
    U = np.random.random(1)
    if U<=p1:
        return stats.expon.ppf(scale=1/l1,q=rand)
    else:
        return stats.expon.ppf(scale=1/l2,q=rand)

def sim(N,a_typ,s_typ):
    np.random.seed(100)

    randoms1 = np.random.random(N)
    randoms2 = np.random.random(N)

    blocked = 0
    service_unit = np.zeros(m)
    arriveTime = 0
    arrivals = np.zeros(N)

    for i in range(N):
        arrivals[i] = arrival(mtbc,a_typ,randoms1[i])
        arriveTime += arrivals[i]
        serviceTime = service_time(mst,s_typ,randoms2[i])
        if np.min(service_unit) <= arriveTime:
            for s in range(m):
                if service_unit[s] <= arriveTime:
                    service_unit[s] = arriveTime + serviceTime
                    break
        else:
            blocked +=1
    pBlock = blocked/N
    return pBlock, np.mean(arrivals)


theta1 = sim(10000,"pois","exp")[0]
theta2 = sim(10000,"hyperexp","exp")[0]

print(theta2 - theta1)




#%% Question 7





def crude_estimator7(N,a):
    I = 100
    P = np.ones(N)
    for i in range(N):
        x = stats.norm.rvs(size=100) #100 = estimation precision
        P[i] = sum(x>a)/N
    return P

CI(crude_estimator7(100,2))
```

```python
def important_samp(N,a,sig):
    Y = stats.norm.rvs(a,sig,size=N)
    h = Y > a

    g = stats.norm.pdf(Y,a,sig)
    f = stats.norm.pdf(Y)

    Z = h*f/g
    return Z

CI(important_samp(10000,2,1))

1-stats.norm.cdf(a)



#%% Question 9

def important_samp9(N,k):
    Y = stats.pareto.rvs(k-1,size=N)
    h = Y

    g = stats.pareto.pdf(Y,k-1)
    f = stats.pareto.pdf(Y,k)

    Z = h*f/g
    return Z

CI(important_samp9(100,1.05))


# -*- coding: utf-8 -*-
"""
Created on Thu Jun  8 11:09:58 2023

@author: filip
"""

import numpy as np
import scipy.stats as stats
from scipy.stats import t
```

```
import matplotlib.pyplot as plt


n=10000
m = 10      #service units
mst = 8     #mean service time
mtbc = 1    #mean time between customers


def poisson():
    P = [mst**i/ np.math.factorial(i) for i in range(m+1)]
    return P/np.sum(P)




def g(x):
    A=mst
    return A**x /np.math.factorial(x)


def Metropolis_Hasting(N):
    X = np.zeros(N) #states
    for i in range(N-1):
        Y = np.random.randint(0, m+1) #dx sampled from some symmetric
 ↪  dist

        if g(Y) >= g(X[i]):
            X[i+1] = Y

        elif (g(Y) <= g(X[i]) ) and (np.random.random() <= g(Y)/g(X[i])
         ↪  ):
            X[i+1] = Y
        else:
            X[i+1] = X[i]
    return X

r = Metropolis_Hasting(n)

plt.plot(poisson(),color='red')
plt.hist(r,bins=[-0.5, 0.5, 1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,
 ↪  8.5,  9.5, 10.5],density=1,edgecolor="white")
```

```
An = poisson()

pvals = np.zeros(100)

sampL = 4
for i in range(100):
    run = Metropolis_Hasting(n)[::sampL]

    obs = np.zeros(m+1)
    s = 0
    for k in np.unique(run):
        obs[s] = sum(run == k)
        s+=1
    pvals[i] = stats.chisquare(obs,An*n/sampL)[1]

#plt.hist(run,bins=11,density=1)
#plt.plot(obs,color="red")


plt.figure()
plt.hist(pvals)




#%% Question 2a & 2b

m=10
A1,A2 = 4,4

def g2(i,j):
    if i+j > 10:
        return 0
    return A1**i /np.math.factorial(i) * A2**j /np.math.factorial(j)


def poisson2():
    P = np.zeros((m+1,m+1))
    for i in range(m+1):
        P[i,:] = [g2(i,j) for j in range(m+1)]
    return P/np.sum(P)


def showAnalytical():
    plt.figure()
    pA = poisson2()
    plt.imshow(pA)
    plt.colorbar()
    return

def showPlot(rr):
    #MH direct
    plt.figure()
    pHS = np.histogram2d(x=rr[:,0], y=rr[:,1],density=True,bins=11)[0]
    plt.imshow(pHS)
    plt.colorbar()
    return

def Metropolis_Hasting_2Direct(N):
    X = np.zeros((N,2)) #states
    for i in range(N-1):
        Y1 = np.random.randint(0, m+1)
        Y2 = np.random.randint(0, m+1-Y1)

        if g2(Y1,Y2) >= g2(X[i,0],X[i,1]):
            X[i+1,:] = Y1,Y2

        elif (g2(Y1,Y2) <= g2(X[i,0],X[i,1]) ) and (np.random.random()
         ↪  <= g2(Y1,Y2)/g2(X[i,0],X[i,1]) ):
            X[i+1,:] = Y1,Y2
        else:
            X[i+1,:] = X[i,:]
    return X
r2 = Metropolis_Hasting_2Direct(n)



#Analytical
```

```
showAnalytical()
showPlot(r2)


def sample2Values(U):
    List = np.zeros((66,2))
    k=0
    for j in range(m+1):
        for i in range(m+1-j):
            List[k,:] = [i,j]
            k+=1
    return List[np.int(np.random.random()*66),:]


def Metropolis_Hasting_2Coord(N):
    X = np.zeros((N,2)) #states
    Y1,Y2 = sample2Values(np.random.random())
    switch = True
    for i in range(N-1):
        if g2(Y1,Y2) >= g2(X[i,0],X[i,1]):
            X[i+1,:] = Y1,Y2

        elif (g2(Y1,Y2) <= g2(X[i,0],X[i,1]) ) and (np.random.random()
         ↪  <= g2(Y1,Y2)/g2(X[i,0],X[i,1]) ):
            X[i+1,:] = Y1,Y2
        else:
            X[i+1,:] = X[i,:]

        if switch == True:
            Y1 =np.random.randint(0, m+1-Y2)
        else:
            Y2 =np.random.randint(0, m+1-Y1)
        switch = (switch == False)
    return X

r3 = Metropolis_Hasting_2Coord(n)
showPlot(r3)
```

```
#pvalues tests
sims = 100
n = 10000

UniqList = np.zeros((66,2))
k=0
for j in range(m+1):
    for i in range(m+1-j):
        UniqList[k,:] = [i,j]
        k+=1

An2 = poisson2()
An2 = An2[An2!=0]
pvals2 = np.zeros(sims)

sampL2 = 4
for i in range(sims):
    run = Metropolis_Hasting_2Direct(n)[::sampL2]

    obs = np.zeros(66)
    for k in range(66):#np.unique(run,axis=0):
        for rr in range(np.shape(run)[0]):
            obs[k] += np.logical_and(UniqList[k,0] == run[rr,0],
 ↪  UniqList[k,1] == run[rr,1])

    pvals2[i] = stats.chisquare(obs,An2*n/sampL2)[1]

plt.figure()
plt.plot(obs)
plt.plot(An2*n/sampL2)

plt.figure()
plt.hist(pvals2)


#%% Question

# -*- coding: utf-8 -*-
"""
```

```
Created on Fri Jun  9 10:39:36 2023

@author: filip
"""

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.stats as scs
import random
import math

#%%


#importing Travelling Salesman Matrix
df = pd.read_csv(r"C:\Users\filip\OneDrive\Documents\DTU_kandidat\4.
 ↪  semester\Stochastic sim\cost.csv",header=None)
TSM = df.to_numpy()




#%%
""" Question a """

#Cooling scheme
def cool(k):
    #return 1/np.sqrt(1+k)
    return 1/ np.log(k+1)
    #1 - np.log(k+1)

#cost of Question a
def costEuclid(route):
    return sum(abs(np.diff(route)))

def random_neighbor(state):
    a = np.array(state)
    r1,r2 = np.random.randint(0, 20, 2)
    a[r1],a[r2] = a[r2],a[r1]
    #a[0:-1] = np.random.randint(0, 20, 20)
    if a[0] != a[-1]:
        a[-1] = a[0]
    return a


def random_coordinates(X):
    C= np.random.randint(0,100+1,[len(X)-1,2]).tolist()
    C.insert(X[-1],C[X[-1]])
    return np.array(C)

#cost of Question a
def newCostEuclid(route,coord):
    dist = [ np.sqrt( (coord[route[N+1] ,0] - coord[route[N] ,0])**2 +
 ↪  (coord[route[N+1] ,1] - coord[route[N] ,1])**2)  for N in
 ↪  range(len(route)-1)]
    return np.sum(dist)


#initial random solution
X = np.array(range(len(TSM)))
random.shuffle(X) #randomize starting route

X = np.concatenate([X,[X[0]]])
init_X = X
XC = random_coordinates(X)
U = newCostEuclid(X,XC)


n = 100000#number of simulations



cost_tracker = np.zeros(n)

ks = 0.0001
for k in range(n):
    T = cool(ks)
    X_candidate = random_neighbor(X)


    U_candidate = newCostEuclid(X_candidate,XC)
    if U_candidate < U:
        X = X_candidate
        U = U_candidate
    elif np.exp(-(U_candidate-U)/T ) > random.random():
        #print(np.exp(-(U_candidate-U)/T ))
        X = X_candidate
        U = U_candidate
    cost_tracker[k] = U
    ks = ks+0.0001#+0.005
    #print(ks)

plt.figure()

plt.scatter(XC[X][:,0],XC[X][:,1])
plt.plot(XC[X][:,0],XC[X][:,1])

plt.figure()
plt.scatter(XC[:,0],XC[:,1])
plt.plot(XC[:,0],XC[:,1])

plt.figure()
plt.plot(cost_tracker)


#%% Question 2



def cost(route):
    c = 0
    for k in range(len(route)-1):
        c += TSM[route[k],route[k+1]]
    return c
#initial random solution
init = np.array(range(len(TSM)))
random.shuffle(init) #randomize starting route

init = np.concatenate([init,[init[0]]])
#print(init)
init_cost = cost(init)
```

```
n = 100000#number of simulations
X = init
U = cost(X)

cost_tracker = np.zeros(n)


ks = 0.0001
for k in range(n):
    T = cool(ks)
    X_candidate = random_neighbor(X)
    U_candidate = cost(X_candidate)

    if U_candidate < U:
        X = X_candidate
        U = U_candidate
    elif np.exp(-(U_candidate-U)/T ) > random.random():
        #print(np.exp(-(U_candidate-U)/T ))
        X = X_candidate
        U = U_candidate
    cost_tracker[k] = U
    ks = ks+0.0001#+0.005

plt.figure()
plt.plot(range(n),cost_tracker,marker=' ')
plt.xlabel('k')
plt.ylabel('Cost')


# -*- coding: utf-8 -*-
"""
Created on Fri Jun  9 14:25:59 2023

@author: filip
"""


import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as stats
import random
import math



#%% Question 1


def bootstrap(X):
    n = len(X)
    return np.random.choice(X,n),n

r = 100
X = np.array([56, 101, 78, 67, 93, 87, 64, 72, 80, 69])
a,b=-5,5

count=0
for i in range(r):
    sim,n = bootstrap(X)
    mu = np.mean(sim)
    if np.logical_and(a < sum(X/n)-mu,sum(X/n)-mu < b):
        count+=1

P = count/r

print(P)


#%% Question 2


X = [5,4,9,6,21,17,11,20,7,10,21,15,13,16,8]


r = 1000
var = np.zeros(r)
for i in range(r):
    sim,n = bootstrap(X)
    var[i] = np.var(sim,ddof=1)
```

```
print(np.var(var,ddof=1))


#%% Question 3



def bootstrapEstimateMed(X,r):
    sample_med = np.median(X)
    n = len(X)

    bts = np.random.choice(X,[n,r])
    b_med = np.median(bts,axis=0)
    b_var = np.var(b_med,ddof=1)

    return sample_med, b_var


N = 200
X = stats.pareto.rvs(1.05,size=N)
r = 100



#%% A
np.mean(X)
np.median(X)


#%% B
def bootstrapEstimateMean(X,r):
    sample_mean = np.mean(X)
    n = len(X)

    bts = np.random.choice(X,[n,r])
    b_mean = np.mean(bts,axis=0)
    b_var = np.var(b_mean,ddof=1)
    return sample_mean, b_var

print(bootstrapEstimateMean(X,r))
```

```
#%%
print(bootstrapEstimateMed(X, r))

#%%


N = 10000
X = stats.pareto.rvs(1.05,size=N)
r = 100

print("mean: ", bootstrapEstimateMean(X,r))

print("median: ",bootstrapEstimateMed(X,r))


#it is easier to estimate the median than the mean
```