In [ ]:
```
from scipy.special import factorial
import numpy as np
from scipy.stats import chi2
from matplotlib.patches import Patch
from matplotlib.lines import Line2D
import matplotlib.pyplot as plt
from scipy.stats import poisson
from scipy.stats import chisquare
import plotly.graph_objects as go
from metropolis_hastings import metropolis_hastings, truncated_poisson_samples,m
import plotly.io as pio
from tests import chisquare_test
#pio.renderers.default = "notebook+pdf"
```

# Exercise 6

The number of busy lines in a trunk group (Erlang system) is given by a truncated Poisson distribution P(i) = c · Ai i! , i = 0, . . . m Generate values from this distribution by applying the Metropolis-Hastings algorithm, verify with a χ 2 -test. You can use the parameter values from exercise 4

In [ ]:
```
m = 10 #number of servers
s = 8 #mean service time
lam = 1#arrival_intensity
N =100000 #number of samples
A = lam*s
```

In [ ]:
```
#1The number of busy lines in a trunk group (Erlang system) is
#given by a truncated Poisson distribution

np.random.seed(6)

#unnormalized probability mass function
g = lambda x : A**x / factorial(x)

X_est = metropolis_hastings(g,N,m)
#X_est is the number of busy lines in a trunk group
#create histogram and plot i
hist_estimate,_ = np.histogram(X_est, bins = np.arange(m+2))
#print(hist_estimate)
#sample from the truncated poisson distribution
X_expected = truncated_poisson_samples(A, 0, m, N)
hist_expected,_ = np.histogram(X_expected, bins = np.arange(m+2))
#print(hist_expected)
#Do a Plotly histogram of the estimated values and the expected values in the sa
fig = go.Figure()
fig.add_trace(go.Bar(x = list(range(m+1)), y = hist_estimate, name = 'Estimated'
fig.add_trace(go.Bar(x = list(range(m+1)), y = hist_expected, name = 'Expected')
fig.update_layout(title = 'Estimated and Expected values of the number of busy l
                  xaxis_title = 'Number of busy lines',
                  yaxis_title = 'Count')
fig.show()
```

```
T = sum((hist_estimate - hist_expected)**2/hist_expected)
p_value = 1 - chi2.cdf(np.sum(T), len(hist_estimate)-1)
print('Chi-square test statistic:', T)
print('P-value:', p_value)
```

```
Chi-square test statistic: 11.422001852018466
P-value: 0.32559848895018306
```

*Answer*

With a Chi-square test statistic of 11.42, and a p-value of 0.32 we can conclude that the data generated from the Metropolis-Hastings algorithm is consistent with the truncated Poisson distribution.

## 2

For two different call types the joint number of occupied lines is given by ... You can use A1, A2 = 4 and m = 10

### a)

Use Metropolis-Hastings, directly to generate variates from this distribution.

In [ ]:
```
A1 = 4
A2 = 4

def g_joint(A1,A2):
    return lambda x,y : (A1**x / factorial(x)) * (A2**(y) / factorial(y))

def joint_truncated_poisson_samples(A1, A2, low, high, size=1):
    samples = []
    while len(samples) < size:
        x = np.random.poisson(A1)
        y = np.random.poisson(A2)
        if low <= x + y <= high:
            samples.append((x,y))
    return np.array(samples)
```

In [ ]:
```
#joint samples
X_est_joint = metropolis_hastings_joint(g_joint(A1,A2),N, y_sampling_func=y_samp
X_true_joint = joint_truncated_poisson_samples(A1,A2,0,m,N)

## Take every 175th observation to eliminate dependencies
X_est_joint = X_est_joint[::175]
X_true_joint = X_true_joint[::175]
#joint histogram estimate
hist_joint_estimate,_,_ = np.histogram2d(X_est_joint[:,0],X_est_joint[:,1], bins
#joint histogram true
hist_joint_true,_,_ = np.histogram2d(X_true_joint[:,0],X_true_joint[:,1], bins =
#plot the joint estimated joint histogram
fig = go.Figure(data = [go.Heatmap(z = hist_joint_estimate)])
fig.update_layout(title = 'Metropolis-hastings estimated joint histogram of the
                  xaxis_title = 'Number of busy lines in trunk 1',
                  yaxis_title = 'Number of busy lines in trunk 2')
fig.show()
 #plot the joint true joint histogram
fig = go.Figure(data = [go.Heatmap(z = hist_joint_true)])
```

```
fig.update_layout(title = 'True joint histogram of the number of busy lines in a
                  xaxis_title = 'Number of busy lines in trunk 1',
                  yaxis_title = 'Number of busy lines in trunk 2')
fig.show()
```

In [ ]:
```
#calculate the test statistic for all histograms > 0
# T, p_value = chisquare(hist_joint_estimate.flatten()[hist_joint_estimate.flatt


#print('Chi-square test statistic:', T)
#print('P-value:', p_value)
T = sum( np.divide((hist_joint_estimate[hist_joint_true > 0] - hist_joint_true[h

#T = sum(sum((hist_joint_estimate - hist_joint_true)**2/hist_joint_true))
p_value = 1 - chi2.cdf(np.sum(T), np.shape(hist_joint_true)[0]*np.shape(hist_joi
print('Chi-square test statistic:', T)
print('P-value:', p_value)
```

```
Chi-square test statistic: 96.6127280718713
P-value: 0.9425846369077646
```

*Answer*

The Metropolis-Hastings algorithm was used to generate variates from the distribution. The data was then tested using a Chi-square test. The test statistic was 131 and the p-value was 0.22. This indicates that the data generated from the Metropolis-Hastings algorithm is consistent with the truncated Poisson distribution.

Note: We achieved an acceptable distribution of p values when running 100k samples and using every 175th observation. However, this leaves us with only 572 observations and thus the states (i,j) with low probability mass were not hit more than 5 times meaning we don't have enough observations. When we tried to increase the sample size to 1 million, we observed extremely low p-values because the there is a problem with the low probability states (i.e we observe way higher frequencies compared to the theoretical values).

## C

Use Gibbs sampling to sample from the distribution. This is (also) coordinate-wise but here we use the exact conditional distributions. You will need to find the conditional distributions analytically

*Answer*

The truncated Poisson distribution is given by

$$P(i) = c_1 \frac{A^i}{i!}, i = 0, \ldots, m$$

where $c_1$ is a normalizing constant. The joint distribution is given by

$$P(i,j) = c_2 \frac{A_1^i A_2^j}{i!j!}, 0 \leq i + j \leq m$$

where $c_2$ is a normalizing constant. The conditional distribution of $i$ given $j$ through the definition of conditional probability

$$P(i|j) = \frac{P(i,j)}{P(j)} \qquad = \frac{c_2 \frac{A_1^i A_2^j}{i!j!}}{c_1 \frac{A_2^j}{j!}} = c_i \frac{A_1^i}{i!}$$

Which is essentially a truncated Poisson distribution, but under the condition, $0 \leq j \leq m \quad \& \quad 0 \leq i+j \leq m \Rightarrow 0 \leq i \leq m-j$. We can use the conditional distribution to sample $i$ and $j$ coordinate-wise using the Gibbs sampling algorithm.

In [ ]:
```
#Gibbs sampling
np.random.seed(2098936829)
N = 200000
As = [4, 4]
x0 = [3,3]
m = 10
n_burn = 1000
#print(N)
Xs = np.array(Gibbs(As, N + n_burn, x0, m))
xs = Xs[0,n_burn:]
ys = Xs[1,n_burn:]

xs = xs[::125]
ys = ys[::125]

N = len(xs)
```

In [ ]:
```
hist_est_gibbs = np.histogram2d(xs,ys, bins = [np.arange(m+2),np.arange(m+2)])[0
fig = go.Figure(data = [go.Heatmap(z = hist_est_gibbs)])
fig.update_layout(title = 'Metropolis-hastings estimated joint histogram of the
                  xaxis_title = 'Number of busy lines in trunk 1',
                  yaxis_title = 'Number of busy lines in trunk 2')
fig.show()
```

In [ ]:
```
#chi square test
#T, p_value = chisquare(hist_est_gibbs.flatten()[hist_est_gibbs.flatten() > 0],
T = 0
for i in range(m+1):
    for j in range(m+1):
        if hist_joint_true[i,j] > 0:
            T += (hist_est_gibbs[i,j] - hist_joint_true[i,j])**2/hist_joint_true


X_true_joint = joint_truncated_poisson_samples(A1,A2,0,m,N)
hist_joint_true,_,_ = np.histogram2d(X_true_joint[:,0],X_true_joint[:,1], bins =
```

In [ ]:
```
p_value = 1 - chi2.cdf(np.sum(T), 65)

print('Chi-square test statistic:', T)
print('P-value:', p_value)
```

```
Chi-square test statistic: 78.3546628813818
P-value: 0.12362308383372511
```

*Answer*

The Gibbs sampling algorithm was used to sample from the distribution. The data was then tested using a Chi-square test. The test statistic was 106.01 and the p-value was 0.815 This indicates that the data generated from the Gibbs sampling algorithm is consistent with the truncated Poisson distribution.

## 3

We consider a Bayesian statistical problem. The observations are Xi ~ N(Θ, Ψ), where the prior distribution of the pair (Ξ, Γ) = (log (Θ), log (Ψ)) is standard normal with correlation ρ. The posterior distribution of (Θ, Ψ) is given by

$$P(\Theta, \Psi|X) = \frac{P(X|\Theta, \Psi) \cdot P(\Theta, \Psi)}{P(X)},$$

which can be derived using a standard change of variable technique. The task of this exercise is now to sample from the posterior distribution of (Θ, Ψ) using Markov Chain Monte Carlo.

### a)

Generate a pair (θ, ψ) from the prior distribution, i.e. the distribution for the pair (Θ, Ψ), by first generating a sample (ξ, γ) of (Ξ, Γ).

In [ ]:
```
#xi and gamma of (Ξ, Γ) = (log (θ), log (Ψ))
#joint density of Theta and Psi is normal with correlation rho = 0.5
rho = 0.5

#sample from the joint density
θ, ψ =np.exp(np.random.multivariate_normal([0,0],[[1,rho],[rho,1]]))

print('θ:',θ)
print('ψ:',ψ)
```

```
θ: 1.101843408693259
ψ: 2.493859910186823
```

The θ and ψ sampled from the prior distribution are given as follows: $$

$$\theta = 1.101843408693259$$
$$\psi = 2.493859910186823$$

### b)

Generate Xi = 1, . . . , n with the values of (θ, ψ) you obtained in item 3a. Use n = 10

In [ ]:
```
n = 10
#generate n samples from the univariate distribution N(ϑ,ψ)
np.random.seed(100)
X = np.random.normal(θ,ψ,n)
```

Derive the posterior distribution of (Θ, Ψ) given the sample. *Answer*

The posterior distribution of can be computed by using the Bayes theorem. Under the

assumption that X_i ~ N(Θ, Ψ) and sampled i.d.d. from the distribution, the likelihood function is given by the product of the pdf evaluated at the observed data points (For the implementation the exponential of the log-likelood is computed, in order to avoid numerical issues).

$$p(X|\Theta,\Psi) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\Psi}} \exp\left(-\frac{(X_i-\Theta)^2}{2\Psi}\right)$$
$$= \frac{1}{(2\pi\Psi)^{n/2}} \exp\left(-\frac{1}{2\Psi}\sum_{i=1}^{n}(X_i-\Theta)^2\right)$$

The prior distribution is given by $f(x,y)$ as described in the exercise. The posterior distribution is then given by

$$p(\Theta,\Psi|X) \propto p(X|\Theta,\Psi)f(\Theta,\Psi)$$
$$p(\Theta,\Psi|X) = c \cdot p(X|\Theta,\Psi)f(\Theta,\Psi)$$

where $c = 1/P(X)$ is a normalizing constant, which doesn't need to be computed for the MCMC sampling.

In [ ]:
```
#The log-likelihood of the data X given the parameters ϑ and ψ is. Each X follow
likelihood = lambda X,θ,ψ : np.exp(-n/2*np.log(2*np.pi) - n/2*np.log(ψ**2) - 1/(
#  The joint density f(x, y) of (θ, Ψ) is
prior = lambda  x,y : 1/(2*np.pi*x*y*np.sqrt(1-rho**2))*np.exp(-1/(2*(1-rho**2))

#y is sampled from a folded normal distribution (has commutative property) param
y_sampling_bays = lambda : np.abs(np.random.normal(0,4,2))

posterior = lambda X : lambda θ,ψ : likelihood(X,θ,ψ)*prior(θ,ψ)
```

## d)

Generate MCMC samples from the posterior distribution of (Θ, Ψ) using the Metropolis Hastings method.

In [ ]:
```
X_est_joint = metropolis_hastings_joint(posterior(X),N, y_sampling_func= y_sampl
```

In [ ]:
```
#joint histogram of the X_est_joint samples in the interval \[ϑ-1,ϑ+1\] and \[ψ-
X_est_joint_interval = X_est_joint[(X_est_joint[:,0] > θ-1)*( X_est_joint[:,0] <
hist_joint_estimate,_,_ = np.histogram2d(X_est_joint_interval[:,0],X_est_joint_i
#Plotly histogram of the joint histogram along with the true values of ϑ and ψ
fig = go.Figure(data = [go.Heatmap(z = hist_joint_estimate, x = np.linspace(θ-1,
fig.add_trace(go.Scatter(x = [θ], y = [ψ], mode = 'markers', marker = dict(size
fig.update_layout(title = 'Metropolis hastings samples of θ and ψ from posterior
                  xaxis_title = 'θ',
                  yaxis_title = 'ψ',
                    showlegend = True,
                    legend = dict(x = 0.8, y = 0.8))

fig.show()
```

## e)

Repeat item 3d with n = 100 and n = 1000, still using the values of (θ, ψ) from item 3a.

Discuss the results.

In [ ]:
```
#generate samples from the joint density
n = 100
X = np.random.normal(θ,ψ,n)
X_est_joint = metropolis_hastings_joint(posterior(X),N, y_sampling_func= y_sampl

#joint histogram of the X_est_joint samples in the interval \[ϑ-1,ϑ+1\] and \[ψ-
X_est_joint_interval = X_est_joint[(X_est_joint[:,0] > θ-1)*( X_est_joint[:,0] <
hist_joint_estimate,_,_ = np.histogram2d(X_est_joint_interval[:,0],X_est_joint_i
#Plotly histogram of the joint histogram along with the true values of ϑ and ψ
fig = go.Figure(data = [go.Heatmap(z = hist_joint_estimate, x = np.linspace(θ-1,
fig.add_trace(go.Scatter(x = [θ], y = [ψ], mode = 'markers', marker = dict(size
fig.update_layout(title = 'Metropolis hastings samples of θ and ψ from posterior
                  xaxis_title = 'θ',
                  yaxis_title = 'ψ',
                    showlegend = True,
                    legend = dict(x = 0.8, y = 0.8))

fig.show()
```

*Answer*:

Above it's seeen that when the number of samples is increased, the posterior distribution is more dense around the values θ and ψ used to generate $X$. This is because the likelihood function is more informative when the number of samples is increased. The results for N=1000 are not shown, as they could not be computed at floating point precision, (Due to taking the exponential of an extremely negative number).