

```
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, metropolis_hastings_joint
import plotly.io as pio
pio.renderers.default = "notebook_connected"
```

Exercise 6

The number of busy lines in a trunk group (Erlang system) is given by a truncated Poisson distribution $P(i) = c \cdot A^i / i!$, $i = 0, \dots, m$. 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

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

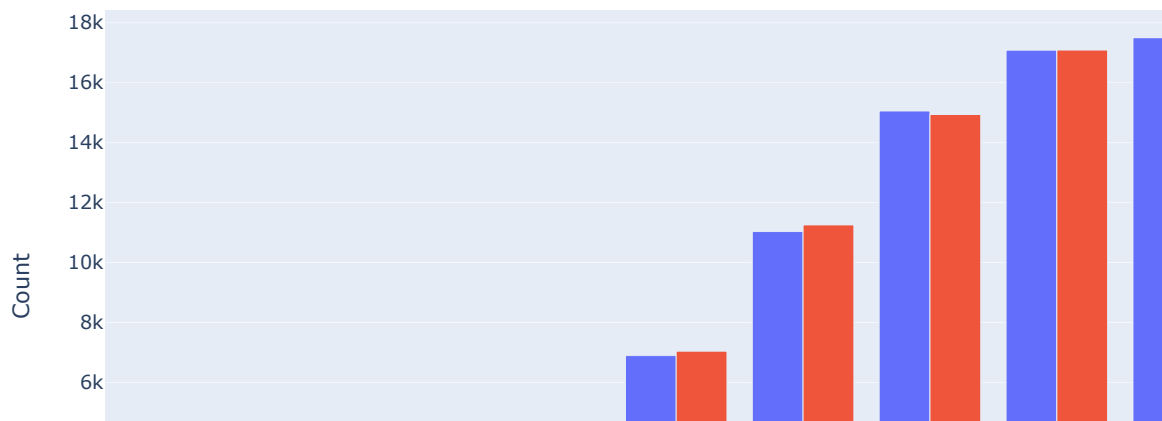
X_est = metropolis_hastings(g, N, m)
#X_est is the number of busy lines in a trunk group
#create histogram and plot it
hist_estimate, _ = np.histogram(X_est, bins = np.arange(m+2))

#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))

#Do a Plotly histogram of the estimated values and the expected values in the same plot
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 lines in a trunk group',
                  xaxis_title = 'Number of busy lines',
                  yaxis_title = 'Count')
fig.show()

T, p_value = chisquare(hist_estimate/N, hist_expected/N)
print('Chi-square test statistic:', T)
print('P-value:', p_value)
```

Estimated and Expected values of the number of busy lines in a trunk group



Chi-square test statistic: 0.000386439217199234

P-value: 1.0

Answer

With a Chi-square test statistic of 0.00023, and a p-value of 1.0 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 $A_1, A_2 = 4$ 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)

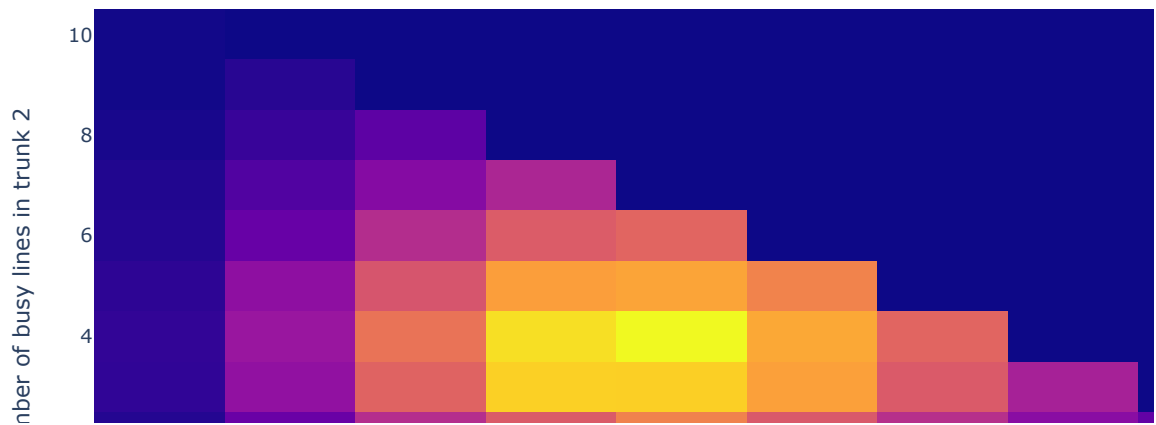
def joint_truncated_poisson_samples(A1, A2, low, high, size=1):
    samples = []
    while len(samples) < size:
        x = np.random.poisson(A1)
        y = np.random.poisson(A2)
        if low <= x + y <= high:
            samples.append((x,y))
    return np.array(samples)

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

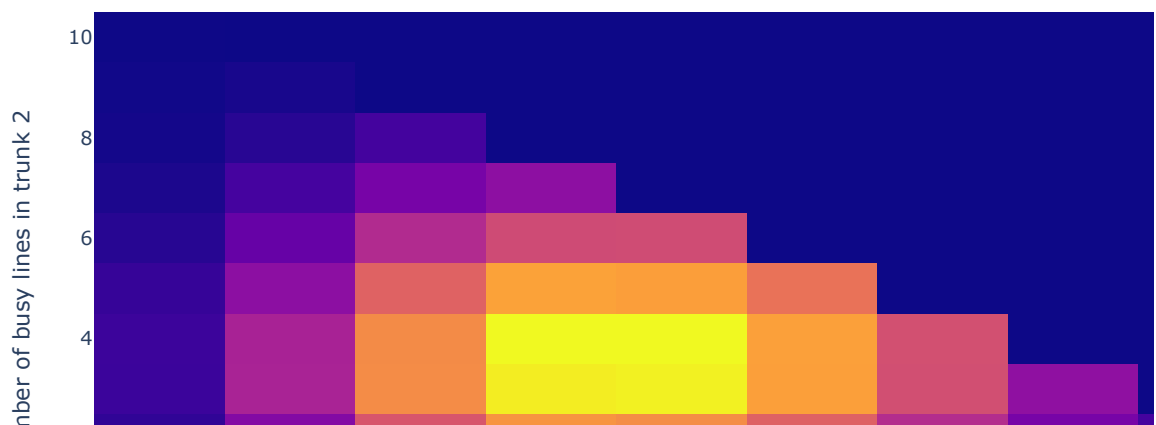
#joint histogram estimate
hist_joint_estimate,_,_ = np.histogram2d(X_est_joint[:,0],X_est_joint[:,1], bins = [np.arange(m+2),np.arange(m+2)])
#joint histogram true
hist_joint_true,_,_ = np.histogram2d(X_true_joint[:,0],X_true_joint[:,1], bins = [np.arange(m+2),np.arange(m+2)])
#plot the joint estimated joint histogram
fig = go.Figure(data = [go.Heatmap(z = hist_joint_estimate)])
```

```
fig.update_layout(title = 'Metropolis-hastings estimated joint histogram of the number of busy lines in
                    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 trunk group',
                    xaxis_title = 'Number of busy lines in trunk 1',
                    yaxis_title = 'Number of busy lines in trunk 2')
fig.show()
```

Metropolis-hastings estimated joint histogram of the number of busy lines in a trunk group



True joint histogram of the number of busy lines in a trunk group



```
In [ ]: #calculate the test statistic for all histograms > 0
T, p_value = chisquare(hist_joint_estimate.flatten()[hist_joint_estimate.flatten() > 0]/N, hist_joint_tr
print('Chi-square test statistic:', T)
print('P-value:', p_value)
```

Chi-square test statistic: 0.03395151192646065
P-value: 1.0

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 0.036 and the p-value was 1.0. This indicates that the data generated from the Metropolis-Hastings algorithm is consistent with the truncated Poisson distribution.

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

$$P(i) = c_1 \frac{A_1^i}{i!}, i = 0, \dots, m$$

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

$$P(i|j) = \frac{P(i, j)}{P(j)} = \frac{c_2 \frac{A_1^i A_2^j}{i! j!}}{c_1 \frac{A_2^j}{j!}} = c_3 \frac{A_1^i}{i!}$$

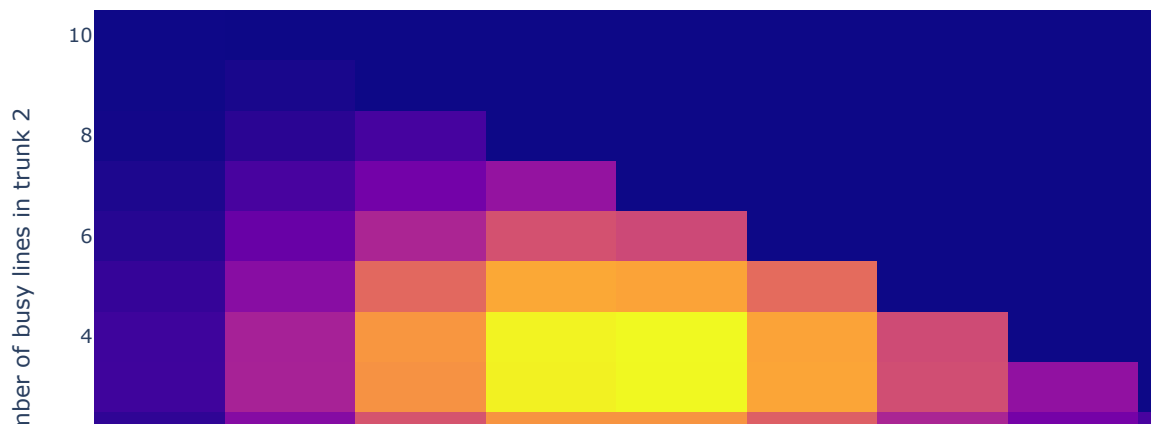
Which is essentially a truncated Poisson distribution, but under the condition,

$0 \leq j \leq m$ & $0 \leq i + j \leq m \Rightarrow 0 \leq i \leq m - j$. We can use the conditional distribution to sample i and j coordinate-wise using the Gibbs sampling algorithm.

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

```
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 number of busy lines in
                    xaxis_title = 'Number of busy lines in trunk 1',
                    yaxis_title = 'Number of busy lines in trunk 2')
fig.show()
```

Metropolis-hastings estimated joint histogram of the number of busy lines in a trunk



```
In [ ]: #chi square test
T, p_value = chisquare(hist_est_gibbs.flatten()[hist_est_gibbs.flatten() > 0]/N, hist_joint_true.flatten())
print('Chi-square test statistic:', T)
print('P-value:', p_value)
```

Chi-square test statistic: 0.0009924092728747901

P-value: 1.0

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 0.0008854790928604727 and the p-value was 1.0. 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 $X_i \sim N(\Theta, \Psi)$, where the prior distribution of the pair $(\Xi, \Gamma) = (\log(\Theta), \log(\Psi))$ is standard normal with correlation ρ . The posterior distribution of (Θ, Ψ) is given by... 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: 3.75919441029067

ψ: 0.9646365118052521

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

$$\theta = 1.101843408693259$$

$$\psi = 2.493859910186823$$

b)

Generate $X_i = 1, \dots, n$ 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 $X_i \sim N(\Theta, \Psi)$ and sampled i.i.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-likelihood is computed, in order to avoid numerical issues).

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

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

$$\begin{aligned} p(\Theta, \Psi|X) &\propto p(X|\Theta, \Psi)f(\Theta, \Psi) \\ p(\Theta, \Psi|X) &= c \cdot p(X|\Theta, \Psi)f(\Theta, \Psi) \end{aligned}$$

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 follows Normal(θ,ψ)
likelihood = lambda X,θ,ψ : np.exp(-n/2*np.log(2*np.pi) - n/2*np.log(ψ**2) - 1/(2*ψ**2+ 1e-16)*np.sum((X-θ)**2))
# 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))*(np.log(x)**2 - 2*rho*np.log(x)*np.log(y) + np.log(y)**2))
#y is sampled from a folded normal distribution (has commutative property) parameters are chosen arbitrarily
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)

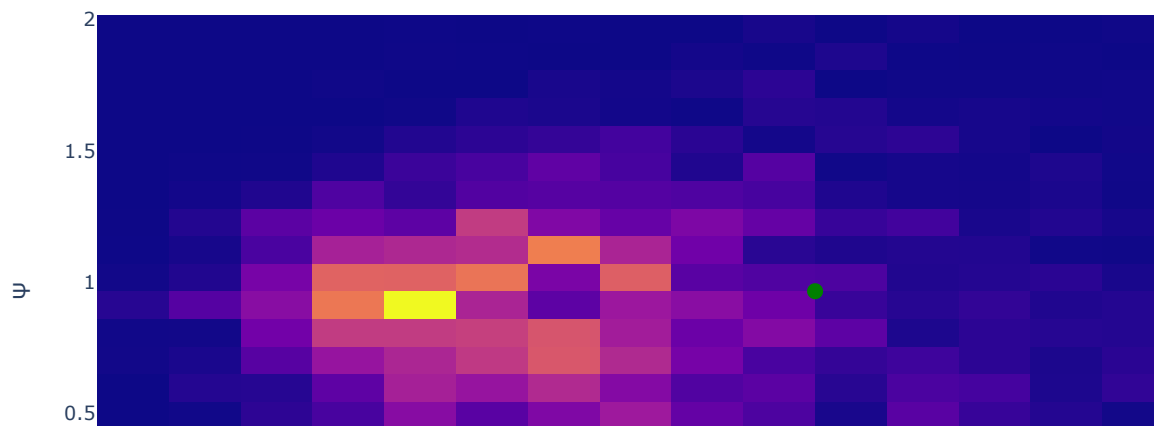
Generate MCMC samples from the posterior distribution of (Θ, Ψ) using the Metropolis Hastings method.

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

```
In [ ]: #joint histogram of the X_est_joint samples in the interval \[θ-1,θ+1\] and \[ψ-1,ψ+1\]
X_est_joint_interval = X_est_joint[(X_est_joint[:,0] > θ-1)*( X_est_joint[:,0] < θ+1)*(X_est_joint[:,1] > ψ-1)*(X_est_joint[:,1] < ψ+1)]
hist_joint_estimate,_,_ = np.histogram2d(X_est_joint_interval[:,0],X_est_joint_interval[:,1], bins = [20,20])
#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,θ+1,20), y = np.linspace(ψ-1,ψ+1,20))])
fig.add_trace(go.Scatter(x = [θ], y = [ψ], mode = 'markers', marker = dict(size = 10, color = 'green')),
fig.update_layout(title = 'Metropolis hastings samples of θ and ψ from posterior distribution with n = 10000',
xaxis_title = 'θ',
yaxis_title = 'ψ',
showlegend = True,
legend = dict(x = 0.8, y = 0.8))

fig.show()
```

Metropolis hastings samples of θ and ψ from posterior distribution with $n = 10$



e)

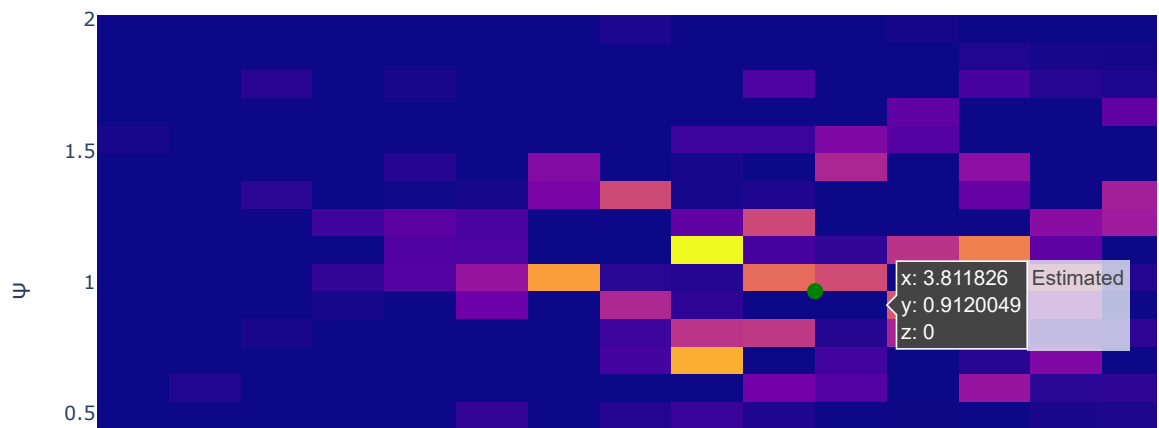
Repeat item 3d with $n = 100$ and $n = 1000$, still using the values of (θ, ψ) from item 3a. Discuss the results.

```
In [ ]: #generate samples from the joint density
n = 100
X = np.random.normal(0,1,n)
X_est_joint = metropolis_hastings_joint(posterior(X),N, y_sampling_func= y_sampling_bays)

#joint histogram of the X_est_joint samples in the interval \[-1,1\] and \[\psi-1,\psi+1\]
X_est_joint_interval = X_est_joint[(X_est_joint[:,0] > -1)*( X_est_joint[:,0] < 1)*(X_est_joint[:,1]
hist_joint_estimate,_,_ = np.histogram2d(X_est_joint_interval[:,0],X_est_joint_interval[:,1], bins = [20
#Plotly histogram of the joint histogram along with the true values of theta and psi
fig = go.Figure(data = [go.Heatmap(z = hist_joint_estimate, x = np.linspace(-1,1,20), y = np.linspace
fig.add_trace(go.Scatter(x = [0], y = [1], mode = 'markers', marker = dict(size = 10, color = 'green'),
fig.update_layout(title = 'Metropolis hastings samples of theta and psi from posterior distribution with n = 10
                    axis_title = 'theta',
                    yaxis_title = 'psi',
                    showlegend = True,
                    legend = dict(x = 0.8, y = 0.8))

fig.show()
```

Metropolis hastings samples of θ and ψ from posterior distribution with $n = 100$



Answer:

Above it's seen 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).