# Metropolis-Hastings Poisson Distribution

January 1, 2023

## 1 Metropolis-Hasting Sampling of Poisson Distribution

### 1.1 Poisson Distribution

$$p(k|\mu) = \frac{\mu^k e^{-\mu}}{k!} \quad \text{for } k \geq 0 \, .$$

where $k$ is an integer and $\mu$ is called the shape parameter. The mean and variance of this distribution are both equal to $\mu$.

### 1.2 Meaning of Samplig Posisson Distribution

`sampling a Poisson distribution` we mean we will obtain a set of values: $k$ values: $\{k_1, k_2, k_3, ...\}$ that follow this distribution

### 1.3 The Algorithm

1. Choose an initial $k$ (call it $k0$), having already fixed mean ($\mu$)
2. Given $k_i$, sample a uniform random number $x$ from 0 to 1 (so $x \sim U(0,1)$) and propose $k' = k_i + 1$ if the $x > 0.5$, otherwise propose $k' = k_i - 1$
3. Compute the Metropolis ratio $r = min(1, p(k'|\mu) \, / \, p(k_i|\mu))$ using the discrete Poisson distribution.
4. Given another uniform random number $y \sim U(0,1)$, $k_{i+1} = k'$ if $y \leq r$, else $k_{i+1} = k_i$ (i.e., keep the same value for the next $k$).
5. Repeat 2.-4. until you think you have enough samples of $k$.

### 1.4 Notes on Implementation

1. We need two sets of $n$ numbers of random numbers between 0 and 1 from uniform distribution, we can generate them at once outside of the loop
2. Since we cannot compute factorial for negative number we will not let kp be less than 0

### 1.5 Implementation

#### 1.5.1 Import the Necessary Libaries

[1]:
```
import numpy as np
from math import factorial,ceil
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn
```

```
plt.style.use('seaborn')
```

### 1.5.2 Define Poission Distribution Function

[2]:
```
def poisson_function(k, mean):
    """
    Returns a Poisson distribution value for k with mean
    """
    return mu**k * np.exp(-mean) / factorial(k)
```

### 1.5.3 Intalize the Value of mean($\mu$) and $K_0$

[3]:
```
mu=3.2
k0=10
```

### 1.5.4 Define Number of Steps

[4]:
```
n_steps = 10000
```

### 1.5.5 Generate Uniform Random Variable to be used later

[5]:
```
uniform_1 = stats.uniform.rvs(size=n_steps)
uniform_2 = stats.uniform.rvs(size=n_steps)
```

### 1.5.6 Intalize an empty array to store simulated result

[6]:
```
k_array = np.zeros(n_steps, dtype=int)
k_array[0] = k0
accept_rate_array = []
accept_rate_count = 0
```

[7]:
```
for i in range(0,n_steps-1):
    k_now = k_array[i]
    if uniform_1[i] > 0.5:
        kp = k_now + 1
    else:
        # Since we cannot compute factorial for negative number we will not let␣
 ↪kp be less than 0
        kp = max(0, k_now - 1)
    metropolis_r = min(1,poisson_function(kp, mu) / poisson_function(k_now, mu))

    if uniform_2[i] <= metropolis_r:
        k_array[i+1] = kp
        accept_rate_count+=1
        accept_rate_array.append(accept_rate_count/(i+1))
```

```
    else:
        k_array[i+1] = k_now
        accept_rate_array.append(accept_rate_count/(i+1))
```

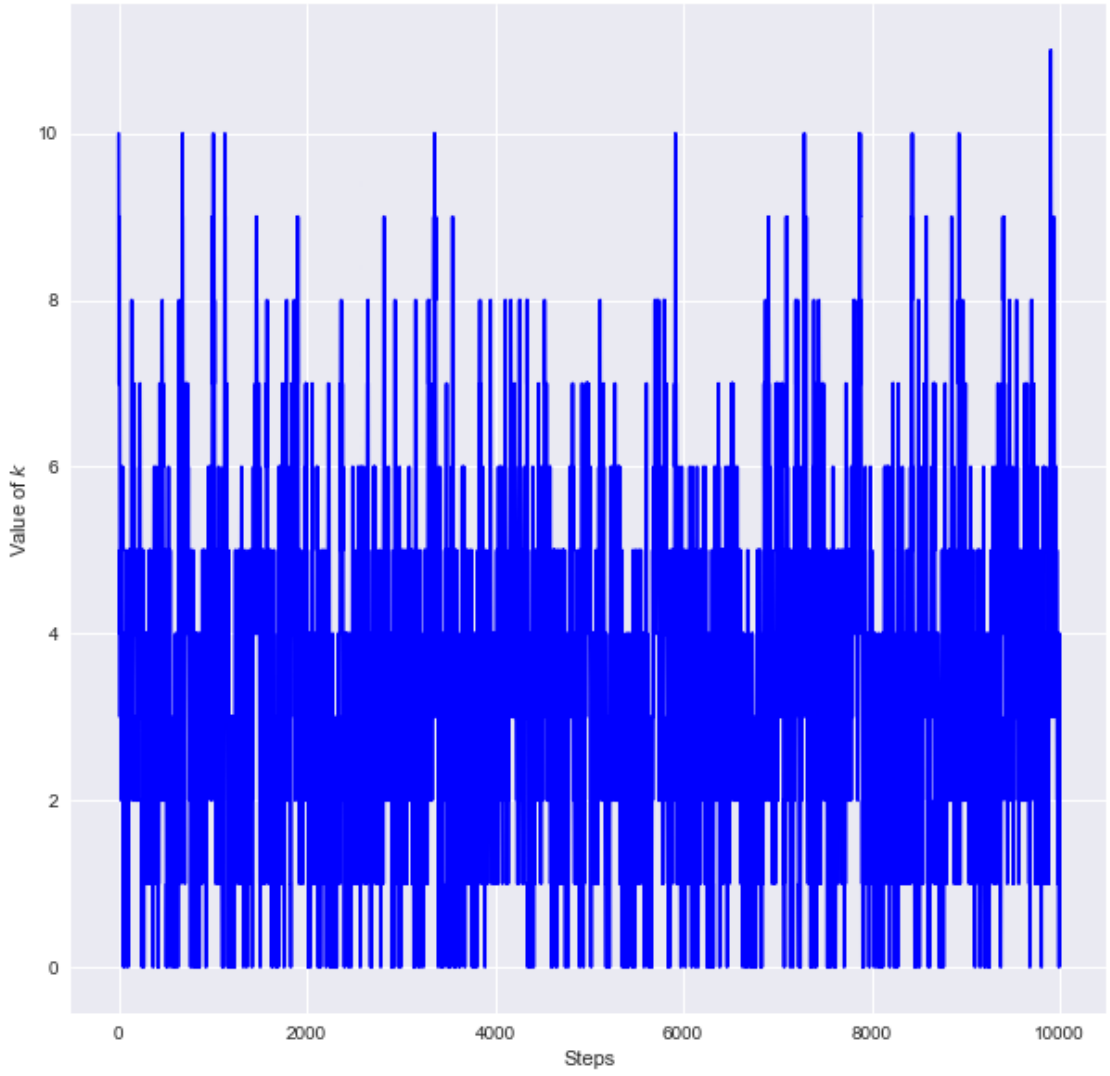
### 1.5.7 Burn In Steps

[8]:
```
burn_in_steps = ceil(0.10*n_steps)
```

[9]:
```
# Check the mean and standard deviations from the samples against exact
print(f' MCMC mean = {np.mean(k_array[burn_in_steps:]):.2f}')
print(f'Exact mean = {stats.poisson.mean(mu=mu):.2f}')
print(f'   MCMC sd = {np.std(k_array[burn_in_steps:]):.2f}')
print(f'  Exact sd = {stats.poisson.std(mu=mu):.2f}')
```

```
 MCMC mean = 3.38
Exact mean = 3.20
   MCMC sd = 1.93
  Exact sd = 1.79
```

[10]:
```
plt.figure(figsize=(10,10))
plt.plot(range(n_steps), k_array, color='blue')
plt.xlabel('Steps')
plt.ylabel('Value of $k$')
plt.show()
```




[11]:
```
plt.figure(figsize=(10,10))
plt.plot(range(burn_in_steps,n_steps-1), accept_rate_array[burn_in_steps:],␣
↪color='blue')
plt.xlabel('Steps')
plt.ylabel('Acceptance Rate')
plt.show()
```


### 1.6 Refrences:

1. Physics-8820 University of York in 2019
2. Probabilistic Modelling and Reasoning