HOMEWORK 3
COMPUTATIONAL METHODS FOR DATA SCIENCE
FALL SEMESTER 2022

D11948002 資料科學博士班一年級 巫哲嘉

1. Generate Random Variables for Common Distributions. *(20 points)*

(a) Normal

We can use Inverse Method,

Since we know the CDF of normal distribution:

$$\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{x}\exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)dt = \frac{1}{2}\left[1+\mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

then, we can generate U (0,1) random variable (r.v.) and input the r.v. into inverse function:

$$x = F^{-1}(u) = \mu + \sigma\sqrt{2}\,erf^{-1}(2u-1),$$
$$where\ \mathrm{erf}(x) = \tfrac{1}{\sqrt{\pi}}\int_{-x}^{x} e^{-t^2}\,dt\ \ .$$

Therefore, we can generate Normal r.v.

(b) Exponential

We can also use Inverse Method,

The CDF:

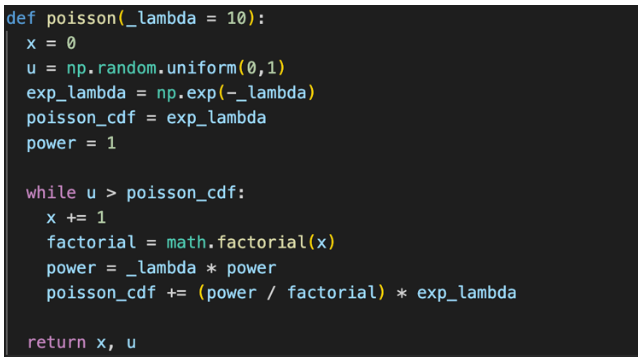
$$F(x) = P(X \le x) = \int_{-\infty}^{x} f(u)du = 1 - e^{-\lambda x}$$

thus, we can generate r.v.:

$$x = F^{-1}(u) = -\frac{\ln(1-u)}{\lambda} = -\frac{\ln(u)}{\lambda}$$

(c) Poisson

Since Poisson is a discrete distribution, we can consider using "Inversion with sequential search", below is the implementation:



```python
def poisson(_lambda = 10):
  x = 0
  u = np.random.uniform(0,1)
  exp_lambda = np.exp(-_lambda)
  poisson_cdf = exp_lambda
  power = 1

  while u > poisson_cdf:
    x += 1
    factorial = math.factorial(x)
    power = _lambda * power
    poisson_cdf += (power / factorial) * exp_lambda

  return x, u
```

when using sequential search in the inversion method, we need to use the recurrence relation. That is, $\frac{P(X=i+1)}{P(X=i)} = \frac{\lambda}{i+1}$, where i ≥ 0.

(d) Chi-Square

By the transformation method. By the definition of Chi-Square distribution, it is the summation of k squared Normal random variables. Now, we know how to generate Normal random variables, we could use the idea of transformation to generate Chi-Square random variables.

To generate a random variable of Chi-Square (df=k), we can do the following step:

(1) We first generate k Normal random variables with U (0, 1) like what we do in (a).

(2) Square of each random variable.

(3) Sum up all of them.

(e) $F_{k,m}$

By "Inverse method",

CDF:

$$F(x;k,m) = \int_{-\infty}^{x} f_X(t)dt = \int_0^x \frac{\Gamma\left[\frac{(k+m)}{2}\right]}{\Gamma\left(\frac{k}{2}\right)\Gamma\left(\frac{m}{2}\right)} (\frac{k}{m})^{\frac{k}{2}} \frac{t^{\frac{k-2}{2}}}{\left[1+\left(\frac{k}{m}\right)t\right]^{\frac{k+m}{2}}} dt$$

where the PDF $f_X(x) = cx^{\frac{k}{2}-1}(1+\frac{k}{m}x)^{-\frac{k+m}{2}}, if\ x \in \mathbb{R}_X$

while $c = (\frac{k}{m})^{\frac{k}{2}} \frac{1}{B(\frac{k}{2},\frac{m}{2})}$ and B is Beta function.

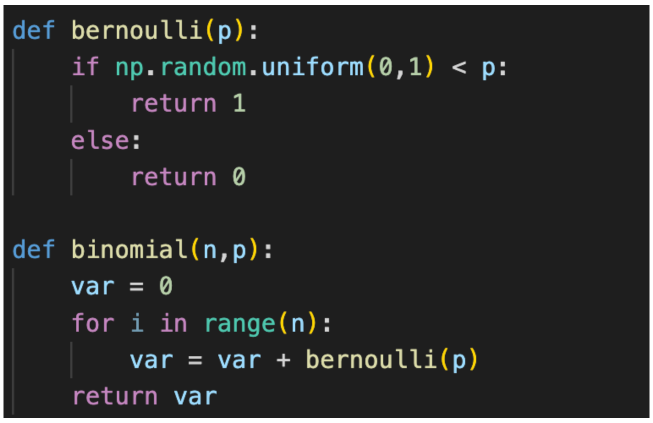
Thus, the r.v. will be:

$$x = F^{-1}(p;k,m) = \{x: F(x;k,m) = p\}$$
$$where\ p = F(x;k,m)$$

(f) Binomial (n, p)

By definition, Binomial r.v. can be calculated by summing up n Bernoulli r.v.

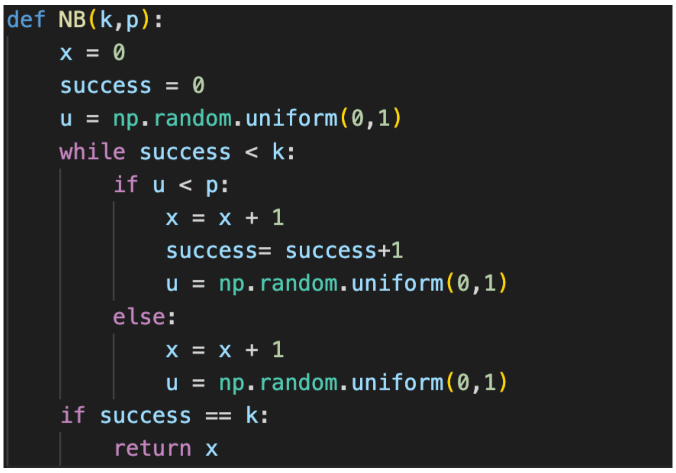
In other words, we need to do Bernoulli trials with probability of success (p) n times and count the number of successful trials.



```python
def bernoulli(p):
    if np.random.uniform(0,1) < p:
        return 1
    else:
        return 0

def binomial(n,p):
    var = 0
    for i in range(n):
        var = var + bernoulli(p)
    return var
```

(g) Negative Binomial

Below is the implementation:


```
def NB(k,p):
    x = 0
    success = 0
    u = np.random.uniform(0,1)
    while success < k:
        if u < p:
            x = x + 1
            success= success+1
            u = np.random.uniform(0,1)
        else:
            x = x + 1
            u = np.random.uniform(0,1)
    if success == k:
        return x
```

x: the total number of trials.

success: the number of success trials.

If success reaches k (the designated k-th success), then the function will return the total number of trials.

(h) Dirichlet $(\alpha_1, \dots \alpha_k)$

Let Y be a random vector with components following a standard gamma distribution. The Dirichlet distribution is a distribution over vectors x that fulfill the conditions $x_i > 0 \; and \; \sum_{i=1}^{k} x_i = 1$.

Draw k independent random samples $y_1, y_2, \dots y_k$ from gamma distribution.

Then $x_i = \frac{1}{\sum_{j=1}^{k} y_j} y_i$ is Dirichlet-distributed.

$$\Rightarrow X = \frac{1}{\sum_{i=1}^{k} Y_i} Y$$

Now that we know the probability mass function of Dirichlet distribution, we use the CDF of Dirichlet distribution and repeat the variable generating skills like we did in (c) to generate discrete random variables.

2. **Sampling Problem.** *(10 points)*

(a) Estimate σ^2 using importance sampling with standardized weights

We can estimate the expectation by a known and easily sampled distribution.

$$E_p\big(\phi(X)\big) = \int \phi(x)p(x)dx = \int \phi(x)\frac{p(x)}{q(x)}q(x)\,dx = E_q(\phi(X)w(X))$$

$$\theta_{IS} = \frac{1}{n}\sum_{i} \phi(x_i)\frac{p(x_i)}{q(x_i)}$$

Now we can sample x from distribution $q(x)$ as shown above,

where $w(x) = \frac{p(x)}{q(x)}$ is the sampling weight and θ_{IS} is the estimator.

Suppose:

$$\phi(X) = e^{(-2(x-3)^2)} \;,$$

$$p(x) = \lambda e^{-\lambda x}, if \; x \geq 0 \; (exponential \; distribution)$$

and we already know that:

$$q(x) = e^{-\frac{|x|^3}{3}}$$

so,

$$w(x) = \frac{p(x)}{q(x)} = \lambda\, e^{\frac{|x|^3}{3} - \lambda x}$$

Hence,

$$Var(\theta_{IS}) = \frac{1}{n}(E_p\big(w(x)\phi^2(x)\big) - E_p(\phi(x))^2$$

(b) Repeat the estimation using rejection sampling

Suppose:

$$p(x) = \lambda e^{-\lambda x},\ if\ x \geq 0\ (exponential\ distribution) \longrightarrow \text{ target distribution.}$$

We propose to use:

$$q(x) = e^{-\frac{|x|^3}{3}}.$$

We want to sample from $p(x)$ using a proposal PMF $q(x)$ that we can sample.

First, we need to find the boundary of k such that:

$$\frac{p(x)}{kq(x)} = \frac{\lambda e^{-\lambda x}}{ke^{\frac{-|x|^3}{3}}} \leq 1$$

$$= \frac{\lambda}{k}\, e^{\frac{|x|^3}{3} - \lambda x}$$

Owing to $p(x) \leq kq(x) \Longrightarrow \frac{p(x)}{q(x)} \leq k$, the smallest k is $k = \max\frac{p(x)}{q(x)}$.

Define: $R(x) = \frac{\lambda}{k}\, e^{\frac{|x|^3}{3} - \lambda x}$

Then,

$$\log R(x) = \log\frac{\lambda}{k} + \frac{|x|^3}{3} - \lambda x$$

$$log' R(x) = x|x| - \lambda$$

Solving the equation: $log' R(x) = 0$, we get $x = \sqrt{\lambda}$.

which gives $k = \lambda e^{-\lambda\sqrt{\lambda}}$.

3. π Estimation *(20 points)*

(a) Implement this method for *h*=1 and *n* = 2000. What is the effect of increasing *n*? What is the effect of increasing and decreasing *h*? Please comment.

h\n	20000	25000	30000	40000
1	3.438	3.448	3.437	3.449
0.5	3.468	3.472	3.447	3.455
0.2	3.380	3.330	3.340	3.303
2	3.143	3.127	3.149	3.150

3	3.134	3.148	3.135	3.133

When we fix *h*, increasing or decreasing *n* does not affect the estimate very much. However, when we fix *n*, increasing *h* could drive the estimate closer to 3.14, which is closer to the accurate π.

Python

```python
def pi_estimate(n, h):
    x = [0]
    y = [0]
    t = 0
    while t <n - 1:
        xt = np.random.uniform(-h, h)
        yt = np.random.uniform(-h, h)
        try_x = xt + x[t]
        try_y = yt + y[t]
        if -1 <= try_x <= 1 and -1 <= try_y <= 1:
            x.append(try_x)
            y.append(try_y)
            t += 1
            #print(len(x))
    return x, y

def cal_pi(x, y, n):
    in_circle = 0
    for i in range(len(x)):
        if x[i]**2 + y[i]**2 <= 1:
            in_circle += 1
    pi = 4 * in_circle / n
    return pi


h_list = [1, 0.5, 0.2, 2, 3]
n_list = [20000, 25000, 30000, 40000]
for h in h_list:
    for n in n_list:
        x, y = pi_estimate(n, h)
        pi = cal_pi(x, y, n)
        print("n = ", n, "h = ", h, "pi = ", pi)
```

(b) Explain why this method is flawed. Using the same method to generate candidates, develop the correct approach by referring to the Metropolis-Hastings ratio.

By using this method, the probability of getting a sample near the edge is much less than getting a sample around the center, for if we get a sample out of range, we will resample. The idea of leveraging Metropolis-

Hastings (MH) ratio is that when we move to the next point (x, y), we set it as the new center and repeat the uniform sampling by $x \pm h, y \pm h$. Afterward, we compute the ratio $R = \frac{P(x^*)Q(x^t|x^*))}{P(x^t)Q(x^*|x^t)}$. Since the probability of any sample in uniform distribution is the same, we can take it as a constant. Hence, $\frac{Q(x^t|x^*)}{Q(x^*|x^t)} = 1$. Furthermore, $P(x^t) = \frac{1}{2h} I(-1 \leq x^* \leq 1)$. When $-1 \leq x^* \leq 1$, $P(x^*) = 1$. $P(x^*) = 0$ elsewhere. As a result, when the new sample locates within the target area, we will accept the new sample for $R = 1$. On the other hand, when the new sample falls out of range, we will reject the new point for $R = 0$, and stay at the same location. With MH ratio, if we receive a sample around the edge which is highly likely to breach the boundary, the point will tend to stay near the edge and therefore balance the probability between the center area and the edge area. The distribution will behave more similar to the uniform distribution.

(c) Implement your approach from Part (b) and calculate $\hat{\pi}$.

h\n	**20000**	**25000**	**30000**	**40000**
1	3.113	3.129	3.145	3.158
0.5	3.150	3.132	3.136	3.147
0.2	3.189	3.069	3.142	3.125
2	3.137	3.14	3.136	3.145
3	3.113	3.113	3.141	3.150

In comparison with the table in (a), the calculated results oscillate around 3.1 with smaller variance than the results in (a). However, the trend is similar that when h < 1, the estimated π is farther away from 3.14. When h > 1, the estimated π is relatively close to 3.14. Moreover, as n increases, however the size of h changes, the estimated π remain stable and quite close to 3.14.

Python

```python
def mh_pi_estimate(n, h):
    x = [0]
    y = [0]
    t = 0
    while t < n - 1:
        xt = np.random.uniform(x[t] - h, x[t] + h)
        yt = np.random.uniform(y[t] - h, y[t] + h)
        try_x = xt
        try_y = yt
        #print(try_x)
        #print(try_y)
        if -1 <= try_x <= 1:
            x.append(try_x)
            #y.append(try_y)
        else:
            x.append(x[t])
            #y.append(y[t])
        if -1 <= try_y <= 1:
            y.append(try_y)
        else:
            y.append(y[t])
        t += 1
        #print(x[t], y[t])
            #print(len(x))
    return x, y

h_list = [1, 0.5, 0.2, 2, 3]
n_list = [20000, 25000, 30000, 40000]
for h in h_list:
    for n in n_list:
        x, y = mh_pi_estimate(n, h)
        pi = cal_pi(x, y, n)
        print("n = ", n, "h = ", h, "pi = ", pi)
```

4. **Coal-Mining Disaster Analysis** *(35 points)*

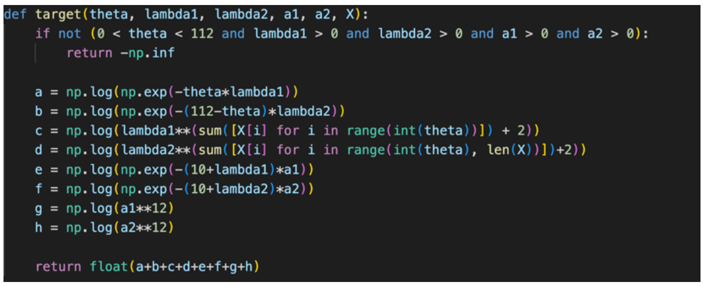
(a) Estimate the posterior mean and provide a histogram each for θ, λ_1 and λ_2.

The idea is that we employ Metropolis-Hastings's algorithm to generate random sample to estimate posterior means and plot histograms for θ, λ_1 and λ_2.

Target Function:

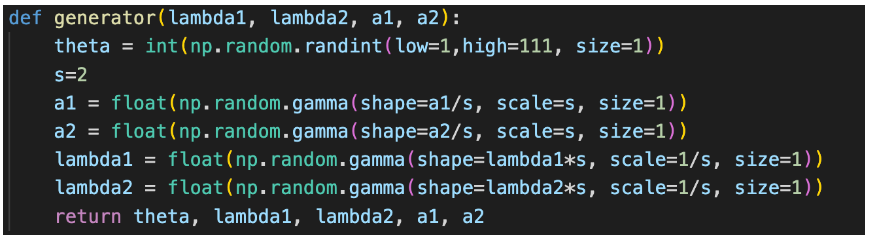
By Bayes' Theorem, we could obtain the posterior distribution function:

$$\begin{aligned}\pi(\theta, \lambda_1, \lambda_2, \mathrm{a}_1, \mathrm{a}_2 | \mathrm{X}) &\propto \mathrm{e}^{-\theta\lambda_1} \lambda_1^{\sum_{i=1}^{\theta} X_i} \mathrm{e}^{-(112-\theta)\lambda_2} \lambda_2^{\sum_{i=\theta+1}^{112} X_i} a_1{}^3 \lambda_1{}^2 \mathrm{e}^{-\mathrm{a}_1\lambda_1} a_2{}^3 \lambda_2{}^2 \mathrm{e}^{-\mathrm{a}_2\lambda_2} \mathrm{a}_1{}^9 \mathrm{e}^{-10\mathrm{a}_1} \mathrm{a}_2{}^9 \mathrm{e}^{-10\mathrm{a}_2} \\ &\propto e^{-\theta\lambda_1-(112-\theta)\lambda_2-(10+\lambda_1)a_1-(10+\lambda_2)a_2} \lambda_1^{\sum_{i=1}^{\theta} X_i+2} \lambda_2^{\sum_{i=\theta+1}^{112} X_i+2} a_1{}^{12} a_2{}^{12}\end{aligned}$$



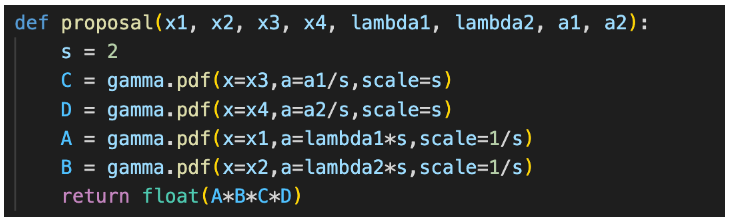
```
def target(theta, lambda1, lambda2, a1, a2, X):
    if not (0 < theta < 112 and lambda1 > 0 and lambda2 > 0 and a1 > 0 and a2 > 0):
        return -np.inf

    a = np.log(np.exp(-theta*lambda1))
    b = np.log(np.exp(-(112-theta)*lambda2))
    c = np.log(lambda1**(sum([X[i] for i in range(int(theta))]) + 2))
    d = np.log(lambda2**(sum([X[i] for i in range(int(theta), len(X))])+2))
    e = np.log(np.exp(-(10+lambda1)*a1))
    f = np.log(np.exp(-(10+lambda2)*a2))
    g = np.log(a1**12)
    h = np.log(a2**12)

    return float(a+b+c+d+e+f+g+h)
```

Sample Generator:



```python
def generator(lambda1, lambda2, a1, a2):
    theta = int(np.random.randint(low=1,high=111, size=1))
    s=2
    a1 = float(np.random.gamma(shape=a1/s, scale=s, size=1))
    a2 = float(np.random.gamma(shape=a2/s, scale=s, size=1))
    lambda1 = float(np.random.gamma(shape=lambda1*s, scale=1/s, size=1))
    lambda2 = float(np.random.gamma(shape=lambda2*s, scale=1/s, size=1))
    return theta, lambda1, lambda2, a1, a2
```

Proposal Function:



```python
def proposal(x1, x2, x3, x4, lambda1, lambda2, a1, a2):
    s = 2
    C = gamma.pdf(x=x3,a=a1/s,scale=s)
    D = gamma.pdf(x=x4,a=a2/s,scale=s)
    A = gamma.pdf(x=x1,a=lambda1*s,scale=1/s)
    B = gamma.pdf(x=x2,a=lambda2*s,scale=1/s)
    return float(A*B*C*D)
```

Generate θ, λ_1 and λ_2

We generate $X_1, \ldots, X_{112}$ with the given information of $\theta,\ \lambda_1,\ \lambda_2$in problem 4.

The $\theta,\ \lambda_1,\ \lambda_2$we generated are summarized as follows:

$(\theta,\ \lambda_1,\ \lambda_2)$ = (85, 2.76, 3.88)

Python

```python
#%% 4.a
theta = np.random.randint(low=1,high=111, size=1)
a1 = np.random.gamma(shape=10,scale=0.1,size=1)
a2 = np.random.gamma(shape=10,scale=0.1,size=1)
lambda1 = np.random.gamma(shape=3, scale= 1/a1, size=1)
lambda2 = np.random.gamma(shape=3, scale= 1/a2, size=1)
X1 = np.random.poisson(lam = lambda1, size=theta)
X2 = np.random.poisson(lam = lambda2, size=112-theta)
X = np.concatenate((X1,X2),axis=None)


#%% 4.a

def generator(lambda1, lambda2, a1, a2):
    theta = int(np.random.randint(low=1,high=111, size=1))
    s=2
    a1 = float(np.random.gamma(shape=a1/s, scale=s, size=1))
    a2 = float(np.random.gamma(shape=a2/s, scale=s, size=1))
    lambda1 = float(np.random.gamma(shape=lambda1*s, scale=1/s, size=1))
    lambda2 = float(np.random.gamma(shape=lambda2*s, scale=1/s, size=1))
    return theta, lambda1, lambda2, a1, a2

def target(theta, lambda1, lambda2, a1, a2, X):
    if not (0 < theta < 112 and lambda1 > 0 and lambda2 > 0 and a1 > 0 and a2 > 0):
        return -np.inf

    a = np.log(np.exp(-theta*lambda1))
    b = np.log(np.exp(-(112-theta)*lambda2))
    c = np.log(lambda1**(sum([X[i] for i in range(int(theta))]) + 2))
    d = np.log(lambda2**(sum([X[i] for i in range(int(theta), len(X))])+2))
    e = np.log(np.exp(-(10+lambda1)*a1))
    f = np.log(np.exp(-(10+lambda2)*a2))
    g = np.log(a1**12)
    h = np.log(a2**12)

    return float(a+b+c+d+e+f+g+h)

def proposal(x1, x2, x3, x4, lambda1, lambda2, a1, a2):
    s = 2
    C = gamma.pdf(x=x3,a=a1/s,scale=s)
    D = gamma.pdf(x=x4,a=a2/s,scale=s)
    A = gamma.pdf(x=x1,a=lambda1*s,scale=1/s)
    B = gamma.pdf(x=x2,a=lambda2*s,scale=1/s)
    return float(A*B*C*D)

#%%

N = 500000
Y = np.zeros([N,5])
V = np.zeros([N,5])
accept = 1
r=0
theta_v0, lambda1_v0, lambda2_v0, a1_v0, a2_v0 = generator(lambda1, lambda2, a1, a2)
V[0] = [theta_v0, lambda1_v0, lambda2_v0, a1_v0, a2_v0]
Y[0] = V[0]

for i in range(1,N):
    epsilon = 1e-300
    theta_v, lambda1_v, lambda2_v, a1_v, a2_v = generator(lambda1=Y[i-1,1],lambda2=Y[i-
1,2],a1=Y[i-1,3],a2=Y[i-1,4])
    V[i] = [theta_v, lambda1_v, lambda2_v, a1_v, a2_v]
    ratio_up = np.exp(target(V[i,0], V[i,1], V[i,2], V[i,3], V[i,4],
X))*proposal(x1=Y[i-1,1], x2=Y[i-1,2], x3=Y[i-1,3], x4=Y[i-1,4], lambda1=V[i,1],
lambda2=V[i,2],a1=V[i,3],a2=V[i,4])
```

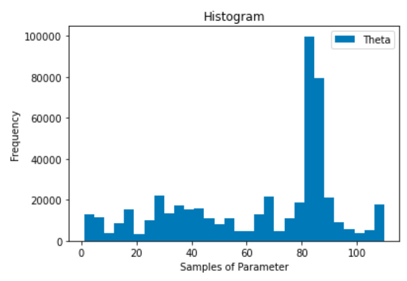
```
    ratio_down = np.exp(target(Y[i-1,0], Y[i-1,1], Y[i-1,2], Y[i-1,3], Y[i-1,4],
X))*proposal(x1=V[i,1], x2=V[i,2], x3=V[i,3], x4=V[i,4], lambda1=Y[i-1,1], lambda2=Y[i-
1,2],a1=Y[i-1,3],a2=Y[i-1,4])
    log_ratio = np.log(ratio_up)-np.log(ratio_down)
    rho = min(log_ratio,1)
    U = float(np.random.uniform(low=0,high=1,size=1))
    if np.log(U) <= rho:
        Y[i] = V[i]
        accept = accept+1
    else:
        Y[i] = Y[i-1]

print('Posterior mean of theta:',Y[:,0].mean())
print('Posterior mean of lambda1:',Y[:,1].mean())
print('Posterior mean of lambda2:',Y[:,2].mean())
```

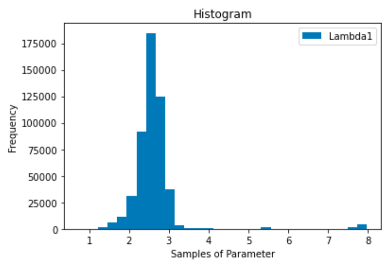
Estimates of Posterior Means

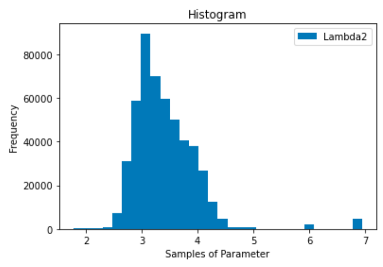
```
Posterior mean of theta: 64.41028
Posterior mean of lambda1: 2.6190533016021686
Posterior mean of lambda2: 3.411394941412507
```

After we generate random samples of each parameter, we calculate the sample means of each random sample. The results of each estimate are listed above.

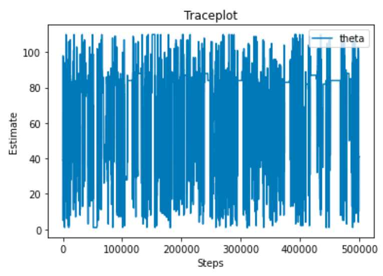
Histograms of θ, λ_1, λ_2

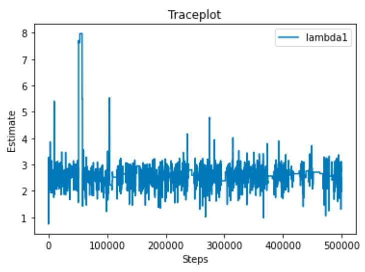


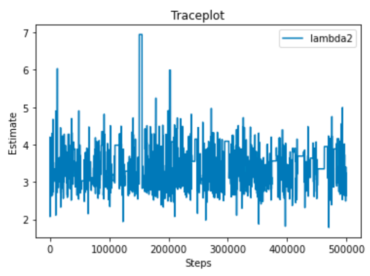




Traceplots of θ, λ_1, λ_2





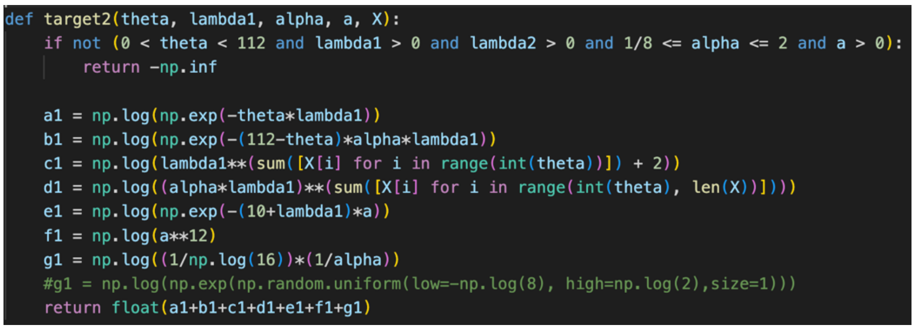


(b) Consider a modification that $\lambda_2 = \alpha\lambda_1$. Assume the same discrete uniform prior for θ and the same prior $\lambda_1 \parallel a \sim Gamma(3, a)$, $a \sim Gamma(10, 10)$. In addition, assume $\log(\alpha) \sim Uniform\left(\log\left(\frac{1}{8}\right), \log 2\right)$. Estimate the posterior mean and provide a histogram each for θ, λ_1, and λ_2.

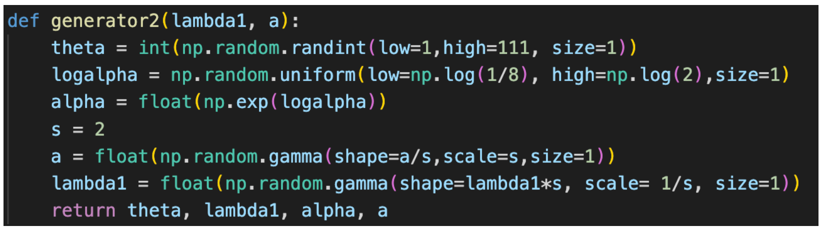
Target Function:

By Bayes' Theorem, we could obtain the posterior distribution function:

$$\pi(\theta,\lambda_1,\alpha,\mathrm{a}|\mathrm{X}) \propto e^{-\theta\lambda_1-(112-\theta)\lambda_2}{\lambda_1}^{\Sigma_{i=1}^{\theta}X_i+2}(\alpha\lambda_1)^{\Sigma_{i=\theta+1}^{112}X_i}(\frac{1}{\log 16}\frac{1}{\alpha})a^{12}e^{-(10+\lambda_1)a}$$


```
def target2(theta, lambda1, alpha, a, X):
    if not (0 < theta < 112 and lambda1 > 0 and lambda2 > 0 and 1/8 <= alpha <= 2 and a > 0):
        return -np.inf

    a1 = np.log(np.exp(-theta*lambda1))
    b1 = np.log(np.exp(-(112-theta)*alpha*lambda1))
    c1 = np.log(lambda1**(sum([X[i] for i in range(int(theta))]) + 2))
    d1 = np.log((alpha*lambda1)**(sum([X[i] for i in range(int(theta), len(X))])))
    e1 = np.log(np.exp(-(10+lambda1)*a))
    f1 = np.log(a**12)
    g1 = np.log((1/np.log(16))*(1/alpha))
    #g1 = np.log(np.exp(np.random.uniform(low=-np.log(8), high=np.log(2),size=1)))
    return float(a1+b1+c1+d1+e1+f1+g1)
```

Sample Generator:


```
def generator2(lambda1, a):
    theta = int(np.random.randint(low=1,high=111, size=1))
    logalpha = np.random.uniform(low=np.log(1/8), high=np.log(2),size=1)
    alpha = float(np.exp(logalpha))
    s = 2
    a = float(np.random.gamma(shape=a/s,scale=s,size=1))
    lambda1 = float(np.random.gamma(shape=lambda1*s, scale= 1/s, size=1))
    return theta, lambda1, alpha, a
```

α Function:

Since $\lambda_2 = \alpha\lambda_1$, $\log\alpha \sim U\left(\log\frac{1}{8}, \log 2\right)$, we need to derive the probability density function for α.

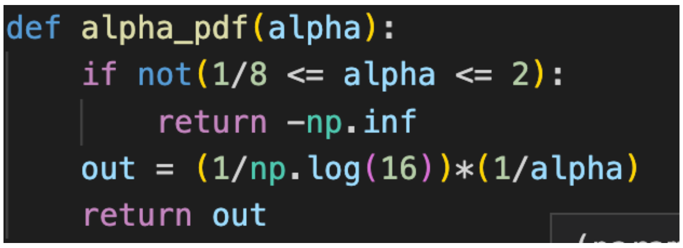
$$X = log\alpha \sim U\left(log(\frac{1}{8}), log2\right)$$

$$\alpha = Y = e^X,\ \tfrac{1}{8} \leq y \leq 2$$

$$\begin{aligned} P[Y \leq y] &= P[e^X \leq y] \\ &= P[X \leq \log y] \\ &= \int_{\log\frac{1}{8}}^{\log y} \frac{1}{\log 16} du \\ &= \frac{log8y}{log16} \end{aligned}$$

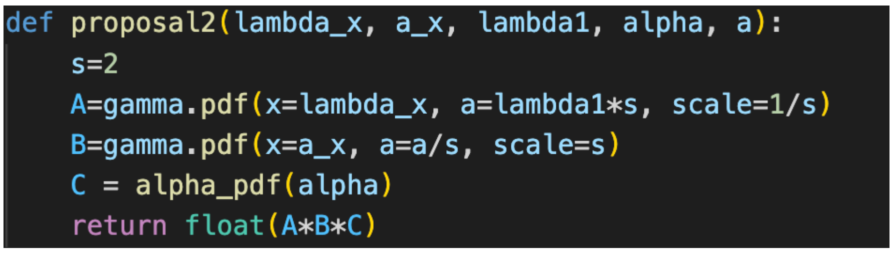
$$\Longrightarrow P[Y = y] = \frac{1}{\log 16} \cdot \frac{1}{\alpha}$$

$$f_Y(y) = \frac{1}{\log 16} \cdot \frac{1}{y} \Longrightarrow f_\alpha(\alpha) = \frac{1}{\log 16} \cdot \frac{1}{\alpha}$$



```python
def alpha_pdf(alpha):
    if not(1/8 <= alpha <= 2):
        return -np.inf
    out = (1/np.log(16))*(1/alpha)
    return out
```

Proposal Function:



```python
def proposal2(lambda_x, a_x, lambda1, alpha, a):
    s=2
    A=gamma.pdf(x=lambda_x, a=lambda1*s, scale=1/s)
    B=gamma.pdf(x=a_x, a=a/s, scale=s)
    C = alpha_pdf(alpha)
    return float(A*B*C)
```

Generate $\theta, \lambda_1, \lambda_2$
The $\theta, \lambda_1, \lambda_2, a$ we generated are summarized as follows: $(\theta, \lambda_1, \lambda_2) = (64, 2.58, 0.84)$
Python

```python
logalpha = np.random.uniform(low=np.log(1/8), high=np.log(2),size=1)
alpha = np.exp(logalpha)
theta = np.random.randint(low=1,high=111, size=1)
a = np.random.gamma(shape=10,scale=0.1,size=1)
lambda1 = np.random.gamma(shape=3, scale= 1/a, size=1)
lambda2 = alpha*lambda1
X1 = np.random.poisson(lam = lambda1, size=theta)
X2 = np.random.poisson(lam = lambda2, size=112-theta)
X = np.concatenate((X1,X2),axis=None)


#%%


def generator2(lambda1, a):
    theta = int(np.random.randint(low=1,high=111, size=1))
    logalpha = np.random.uniform(low=np.log(1/8), high=np.log(2),size=1)
    alpha = float(np.exp(logalpha))
    s = 2
    a = float(np.random.gamma(shape=a/s,scale=s,size=1))
    lambda1 = float(np.random.gamma(shape=lambda1*s, scale= 1/s, size=1))
    return theta, lambda1, alpha, a
```

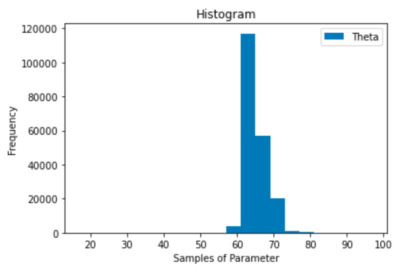
```
def target2(theta, lambda1, alpha, a, X):
    if not (0 < theta < 112 and lambda1 > 0 and lambda2 > 0 and 1/8 <= alpha <= 2 and a 
> 0):
        return -np.inf

    a1 = np.log(np.exp(-theta*lambda1))
    b1 = np.log(np.exp(-(112-theta)*alpha*lambda1))
    c1 = np.log(lambda1**(sum([X[i] for i in range(int(theta))]) + 2))
    d1 = np.log((alpha*lambda1)**(sum([X[i] for i in range(int(theta), len(X))])))
    e1 = np.log(np.exp(-(10+lambda1)*a))
    f1 = np.log(a**12)
    g1 = np.log((1/np.log(16))*(1/alpha))
    #g1 = np.log(np.exp(np.random.uniform(low=-np.log(8), high=np.log(2),size=1)))
    return float(a1+b1+c1+d1+e1+f1+g1)


def alpha_pdf(alpha):
    if not(1/8 <= alpha <= 2):
        return -np.inf
    out = (1/np.log(16))*(1/alpha)
    return out

def proposal2(lambda_x, a_x, lambda1, alpha, a):
    s=2
    A=gamma.pdf(x=lambda_x, a=lambda1*s, scale=1/s)
    B=gamma.pdf(x=a_x, a=a/s, scale=s)
    C = alpha_pdf(alpha)
    return float(A*B*C)

#%%

N = 200000
Y = np.zeros([N,4])
V = np.zeros([N,4])
accept = 1
theta_v0, lambda1_v0, alpha_v0, a_v0 = generator2(lambda1, a)
V[0] = [theta_v0, lambda1_v0, alpha_v0, a_v0]
Y[0] = V[0]


for i in range(1,N):
    theta_v, lambda1_v, alpha_v, a_v = generator2(lambda1=Y[i-1,1],a=Y[i-1,3])
    V[i] = [theta_v, lambda1_v, alpha_v, a_v]
    ratio_up = np.exp(target2(V[i,0], V[i,1], V[i,2], V[i,3],
X))*proposal2(lambda_x=Y[i-1,1], a_x=Y[i-1,3],lambda1=V[i,1],alpha=V[i,2],a=V[i,3])
    ratio_down = np.exp(target2(Y[i-1,0], Y[i-1,1], Y[i-1,2], Y[i-1,3],
X))*proposal2(lambda_x=V[i,1], a_x=V[i,3],lambda1=Y[i-1,1],alpha=Y[i-1,2],a=Y[i-1,3])
    log_ratio = np.log(ratio_up)-np.log(ratio_down)
    rho = min(log_ratio,1)
    U = float(np.random.uniform(low=0,high=1,size=1))
    if np.log(U) <= rho:
        Y[i] = V[i]
        accept = accept+1
    else:
        Y[i] = Y[i-1]


print('Posterior mean of theta:',Y[:,0].mean())
print('Posterior mean of lambda1:',Y[:,1].mean())
print('Posterior mean of lambda2:',(Y[:,1]*Y[:,2]).mean())
```

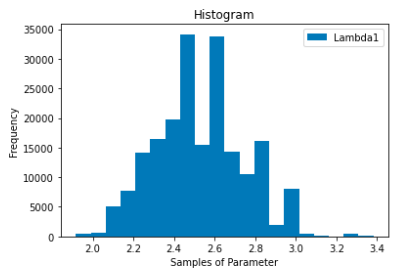
Estimates of Posterior Means

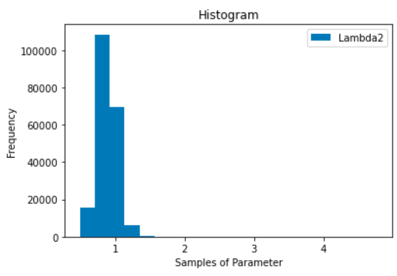
```
Posterior mean of theta: 63.816035
Posterior mean of lambda1: 2.5329241710986556
Posterior mean of lambda2: 0.8753092656740102
```

After we generate random samples of each parameter, we calculate the sample means of each random sample. The results of each estimate are listed above.

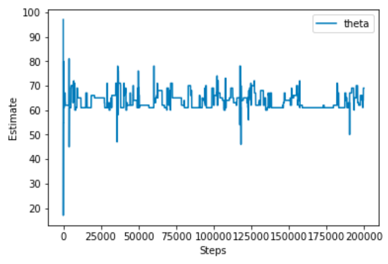
Histogram of $\theta,\ \lambda_1,\ \lambda_2$

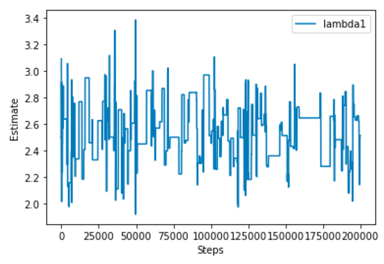


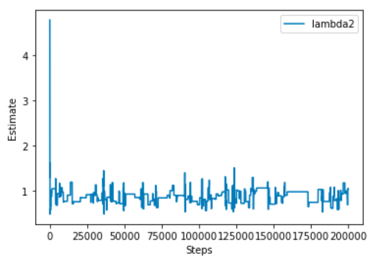




Traceplots of $\theta,\ \lambda_1,\ \lambda_2$





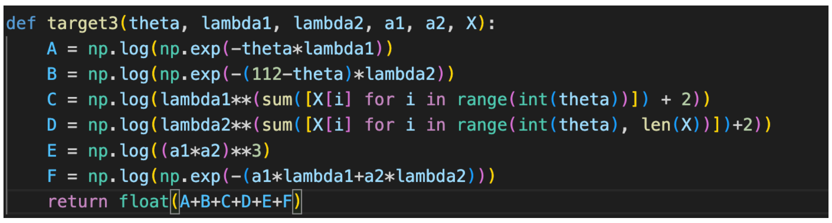


(c) Consider a modification that priors $\lambda_i \parallel a_i \sim Gamma(3, a_i)$, and $a_i \sim Uniform(0, 100)$ independently for $i = 1, 2$ instead. Assume the same discrete uniform prior for θ. Estimate the posterior mean and provide a histogram each for θ, λ_1, and λ_2.

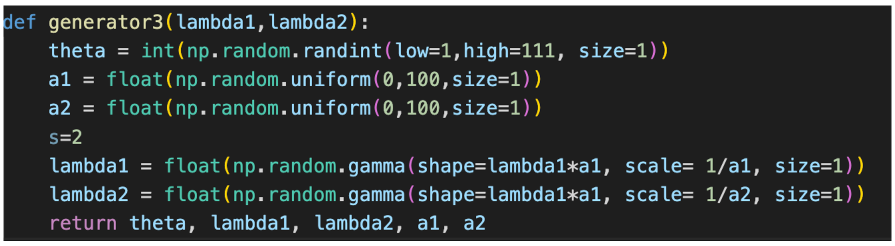
Target Function:

By Bayes' Theorem, we could obtain the posterior distribution function:

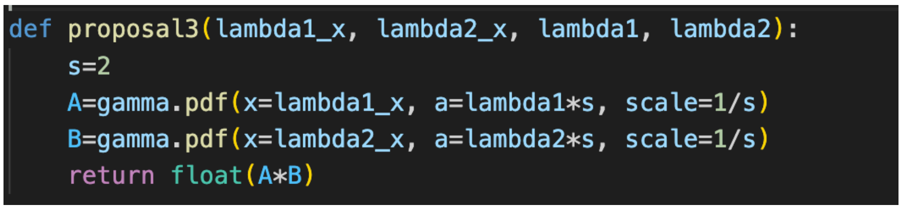
$$\begin{aligned}\pi(\theta, \lambda_1, \lambda_2, \mathrm{a}_1, \mathrm{a}_2|X) &\propto \mathrm{e}^{-\theta\lambda_1}{\lambda_1}^{\sum_{i=1}^{\theta} X_i}\mathrm{e}^{-(112-\theta)\lambda_2}{\lambda_2}^{\sum_{i=\theta+1}^{112} X_i}{a_1}^3{\lambda_1}^2\mathrm{e}^{-\mathrm{a}_1\lambda_1}{a_2}^3{\lambda_2}^2\mathrm{e}^{-\mathrm{a}_2\lambda_2} \\ &\propto \mathrm{e}^{-\theta\lambda_1-(112-\theta)\lambda_2}{\lambda_1}^{\sum_{i=1}^{\theta} X_i+2}{\lambda_2}^{\sum_{i=\theta+1}^{112} X_i+2}(a_1a_2)^3\mathrm{e}^{-(a_1\lambda_1+a_2\lambda_2)}\end{aligned}$$

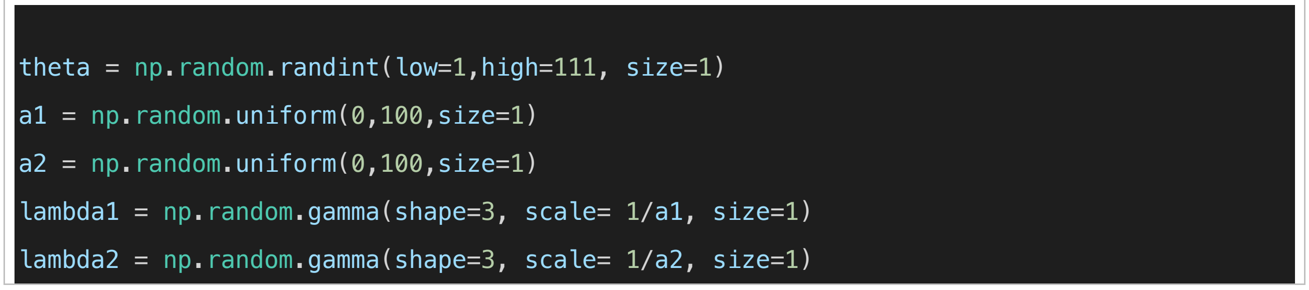

```
def target3(theta, lambda1, lambda2, a1, a2, X):
    A = np.log(np.exp(-theta*lambda1))
    B = np.log(np.exp(-(112-theta)*lambda2))
    C = np.log(lambda1**(sum([X[i] for i in range(int(theta))]) + 2))
    D = np.log(lambda2**(sum([X[i] for i in range(int(theta), len(X))])+2))
    E = np.log((a1*a2)**3)
    F = np.log(np.exp(-(a1*lambda1+a2*lambda2)))
    return float(A+B+C+D+E+F)
```

Sample Generator:


```
def generator3(lambda1,lambda2):
    theta = int(np.random.randint(low=1,high=111, size=1))
    a1 = float(np.random.uniform(0,100,size=1))
    a2 = float(np.random.uniform(0,100,size=1))
    s=2
    lambda1 = float(np.random.gamma(shape=lambda1*a1, scale= 1/a1, size=1))
    lambda2 = float(np.random.gamma(shape=lambda1*a1, scale= 1/a2, size=1))
    return theta, lambda1, lambda2, a1, a2
```

Proposal Function:


```
def proposal3(lambda1_x, lambda2_x, lambda1, lambda2):
    s=2
    A=gamma.pdf(x=lambda1_x, a=lambda1*s, scale=1/s)
    B=gamma.pdf(x=lambda2_x, a=lambda2*s, scale=1/s)
    return float(A*B)
```

Generate θ, λ_1, λ_2
The θ, λ_1, λ_2, a we generated are summarized as follows: $(\theta,\ \lambda_1,\ \lambda_2) = (59, 0.292, 0.12)$
Python


```
theta = np.random.randint(low=1,high=111, size=1)
a1 = np.random.uniform(0,100,size=1)
a2 = np.random.uniform(0,100,size=1)
lambda1 = np.random.gamma(shape=3, scale= 1/a1, size=1)
lambda2 = np.random.gamma(shape=3, scale= 1/a2, size=1)
```

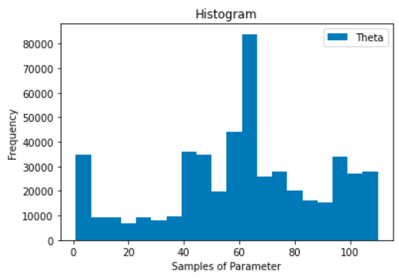
```python
X1 = np.random.poisson(lam = lambda1, size=theta)
X2 = np.random.poisson(lam = lambda2, size=112-theta)
X = np.concatenate((X1,X2),axis=None)

#%%

def generator3(lambda1,lambda2):
    theta = int(np.random.randint(low=1,high=111, size=1))
    a1 = float(np.random.uniform(0,100,size=1))
    a2 = float(np.random.uniform(0,100,size=1))
    s=2
    lambda1 = float(np.random.gamma(shape=lambda1*a1, scale= 1/a1, size=1))
    lambda2 = float(np.random.gamma(shape=lambda1*a1, scale= 1/a2, size=1))
    return theta, lambda1, lambda2, a1, a2

def target3(theta, lambda1, lambda2, a1, a2, X):
    A = np.log(np.exp(-theta*lambda1))
    B = np.log(np.exp(-(112-theta)*lambda2))
    C = np.log(lambda1**(sum([X[i] for i in range(int(theta))]) + 2))
    D = np.log(lambda2**(sum([X[i] for i in range(int(theta), len(X))])+2))
    E = np.log((a1*a2)**3)
    F = np.log(np.exp(-(a1*lambda1+a2*lambda2)))
    return float(A+B+C+D+E+F)

def proposal3(lambda1_x, lambda2_x, lambda1, lambda2):
    s=2
    A=gamma.pdf(x=lambda1_x, a=lambda1*s, scale=1/s)
    B=gamma.pdf(x=lambda2_x, a=lambda2*s, scale=1/s)
    return float(A*B)


#%%

N = 500000
Y = np.zeros([N,5])
V = np.zeros([N,5])
accept = 1
theta_v0, lambda1_v0, lambda2_v0, a1_v0, a2_v0 = generator3(lambda1,lambda2)
V[0] = [theta_v0, lambda1_v0, lambda2_v0, a1_v0, a2_v0]
Y[0] = V[0]

for i in range(1,N):
    theta_v, lambda1_v, lambda2_v, a1_v, a2_v = generator3(lambda1=Y[i-1,1],lambda2=Y[i-
1,2])
    V[i] = [theta_v, lambda1_v, lambda2_v, a1_v, a2_v]
    ratio_up = np.exp(target3(V[i,0], V[i,1], V[i,2], V[i,3], V[i,4],
X))*proposal3(lambda1_x=Y[i-1,1],lambda2_x=Y[i-1,2],lambda1=V[i,1],lambda2=V[i,2])
    ratio_down = np.exp(target3(Y[i-1,0], Y[i-1,1], Y[i-1,2], Y[i-1,3], Y[i-1,4],
X))*proposal3(lambda1_x=V[i,1],lambda2_x=V[i,2],lambda1=Y[i-1,1],lambda2=Y[i-1,2])
    log_ratio = np.log(ratio_up)-np.log(ratio_down)
    rho = min(log_ratio,1)
    U = float(np.random.uniform(low=0,high=1,size=1))
    if np.log(U) <= rho:
        Y[i] = V[i]
        accept = accept+1
    else:
        Y[i] = Y[i-1]


print('Posterior mean of theta:',Y[:,0].mean())
print('Posterior mean of lambda1:',Y[:,1].mean())
print('Posterior mean of lambda2:',Y[:,2].mean())
```

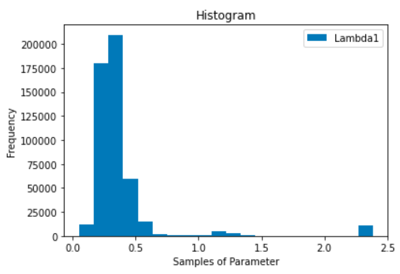
Estimates of Posterior Means

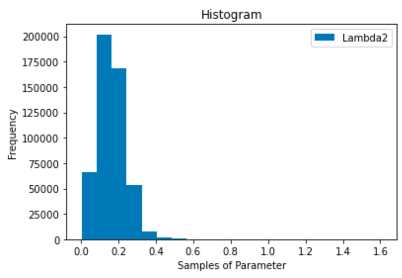
```
Posterior mean of theta: 61.332592
Posterior mean of lambda1: 0.3799976151752662
Posterior mean of lambda2: 0.16213960124116325
```

After we generate random samples of each parameter, we calculate the sample means of each random sample. The results of each estimate are listed above.

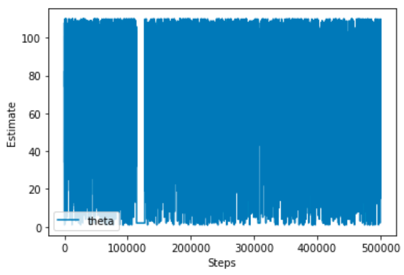
Histogram of $\theta,\ \lambda_1,\ \lambda_2$

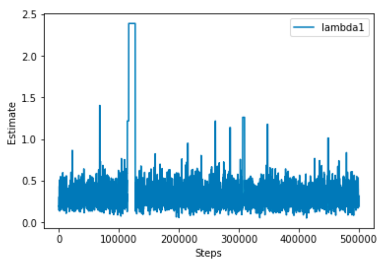


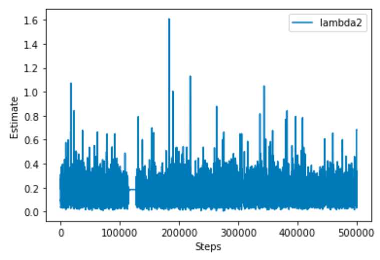




Traceplots of $\theta,\ \lambda_1,\ \lambda_2$







(d) Using the standard setup, derive the conditional distributions necessary to carry out Gibbs sampling for the change-point model.

By Bayes' Theorem,

$$\pi(\theta,\lambda_1,\lambda_2,a_1,a_2|X) \propto \lambda_1^{\sum_{i=1}^{\theta} X_i+2} e^{-(\theta+a_1)\lambda_1} \lambda_2^{\sum_{i=\theta+1}^{112} X_i+2} e^{-(112-\theta+a_2)\lambda_2} a_1^{12} e^{-10a_1} a_2^{12} e^{-10a_2}$$

We can derive the following full conditional distributions:

$$\mathrm{P}\left[\lambda_1|\theta,\lambda_2,a_1,a_2,X\right] = \frac{(\theta+a_1)^{\sum_{i=1}^{\theta} X_i+3}}{\Gamma\left(\sum_{i=1}^{\theta} X_i+3\right)} \lambda_1^{\sum_{i=1}^{\theta} X_i+2} e^{-(\theta+a_1)\lambda_1}$$

$$\mathrm{P}\left[\lambda_2|\theta,\lambda_1,a_1,a_2,X\right] = \frac{(112-\theta+a_2)^{\sum_{i=\theta+1}^{112} X_i+3}}{\Gamma\left(\sum_{i=\theta+1}^{112} X_i+3\right)} \lambda_2^{\sum_{i=\theta+1}^{112} X_i+2} e^{-(112-\theta+a_2)\lambda_2}$$

$$\mathrm{P}\left[a_1|\theta,\lambda_1,\lambda_2,a_2,X\right] = \frac{(10+\lambda_1)^{13}}{\Gamma(13)} a_1^{12} e^{-a_1(10+\lambda_1)}$$

$$\mathrm{P}\left[a_2|\theta,\lambda_1,\lambda_2,a_1,X\right] = \frac{(10+\lambda_2)^{13}}{\Gamma(13)} a_2^{12} e^{-a_2(10+\lambda_2)}$$

$$\mathrm{P}\left[\theta^*|\lambda_1,\lambda_2,a_1,a_2,X\right] = \frac{e^{\theta^*(\lambda_2-\lambda_1)}\left(\frac{\lambda_1}{\lambda_2}\right)^{\sum_{i=1}^{\theta^*} X_i}}{\sum_{\theta=1}^{111} e^{\theta(\lambda_2-\lambda_1)}\left(\frac{\lambda_1}{\lambda_2}\right)^{\sum_{i=1}^{\theta} X_i}}$$

(e) Implement the Gibbs sampler. Use a suite of convergence diagnostics to evaluate the convergence and mixing of your sampler.

Generate $\theta,\ \lambda_1,\ \lambda_2$

The $\theta,\ \lambda_1,\ \lambda_2$ we generated are summarized as follows:

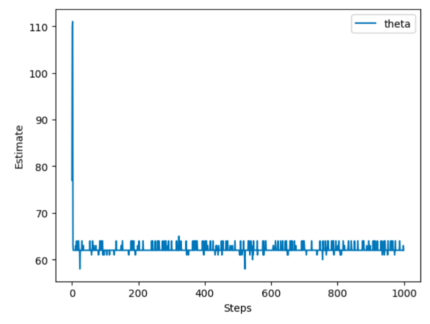
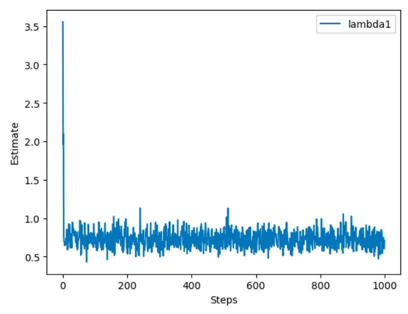
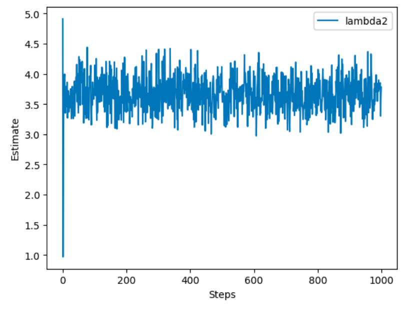
$(\theta,\ \lambda_1,\ \lambda_2) = (62, 0.657, 3.918)$

Python

```python
theta = np.random.randint(low=1,high=111, size=1)
a1 = np.random.gamma(shape=10,scale=0.1,size=1)
a2 = np.random.gamma(shape=10,scale=0.1,size=1)
lambda1 = np.random.gamma(shape=3, scale= 1/a1, size=1)
lambda2 = np.random.gamma(shape=3, scale= 1/a2, size=1)
X1 = np.random.poisson(lam = lambda1, size=theta)
X2 = np.random.poisson(lam = lambda2, size=112-theta)
X = np.concatenate((X1,X2),axis=None)

#%% Posterior Theta Generator
```

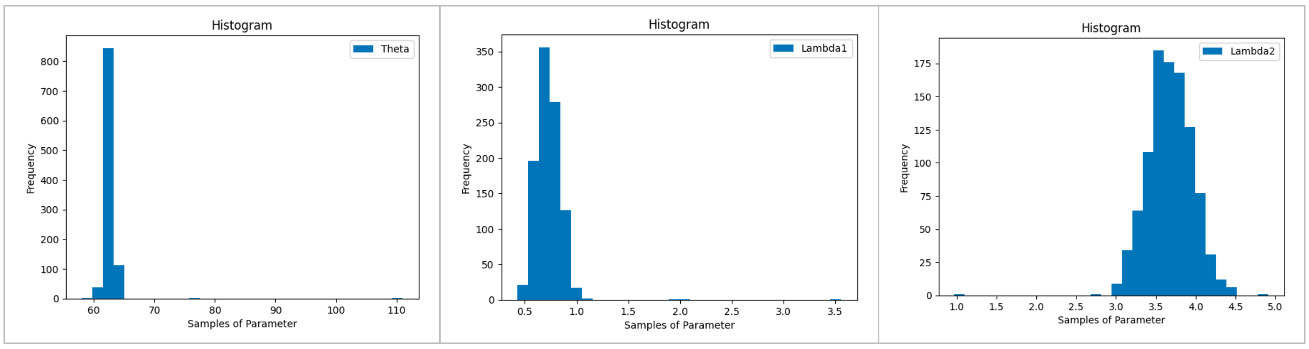
```python
class Theta(object):
    def __init__(self, lambda1, lambda2, X):
        self.lambda1 = lambda1
        self.lambda2 = lambda2
        self.X = X

    def theta_up(self, theta):
        A = np.exp(theta*(self.lambda2 - self.lambda1))
        B = (self.lambda1/self.lambda2)**(sum([self.X[i] for i in range(int(theta))]))
        return float(A*B)

    def theta_pmf(self, theta):
        A = self.theta_up(theta)
        B = sum([self.theta_up(theta = i) for i in range(1,112)])
        out = A/B
        return float(out)

    def theta_cdf(self, theta):
        out = sum([self.theta_pmf(theta = i) for i in range(1,theta+1)])
        return out

    def generator(self):
        U = np.random.uniform(low=0, high=1, size=1)
        t = 1
        while self.theta_cdf(t) < U:
            t = t + 1
        return t


#%% Gibbs Sampler

theta_y0 = int(np.random.randint(low=1,high=111, size=1))
a1_y0 = float(np.random.gamma(shape=10,scale=0.1,size=1))
a2_y0 = float(np.random.gamma(shape=10,scale=0.1,size=1))
lambda1_y0 = float(np.random.gamma(shape=3, scale= 1/a1, size=1))
lambda2_y0 = float(np.random.gamma(shape=3, scale= 1/a2, size=1))
ThetaGenerator = Theta(lambda1, lambda2, X)
ThetaGenerator.generator()

N = 1000
Y = np.zeros([N, 5])
Y[0] = [theta_y0, lambda1_y0, lambda2_y0, a1_y0, a2_y0]
```

```
for t in range(1, N):
    ThetaGenerator = Theta(lambda1 = Y[t-1,1], lambda2 = Y[t-1,2], X = X)
    Y[t,0] = ThetaGenerator.generator()
    Y[t,1] = np.random.gamma(shape=(sum([X[i] for i in range(int(Y[t,0]))])+3), scale =
1/(Y[t,0]+Y[t-1,3]), size=1)
    Y[t,2] = np.random.gamma(shape=(sum([X[i] for i in range(int(Y[t,0]), len(X))])+3),
scale=1/(112-Y[t,0]+Y[t-1,4]), size=1)
    Y[t,3] = np.random.gamma(shape = 13, scale = 1/(10+Y[t,1]), size=1)
    Y[t,4] = np.random.gamma(shape = 13, scale = 1/(10+Y[t,2]), size=1)


#%% Traceplots in 4.e

plt.plot(range(N), Y[:,0], label='theta')
# plt.plot(range(N), Y[:,1], label='lambda1')
# plt.plot(range(N), Y[:,2], label='lambda2')
plt.ylabel("Estimate")
plt.xlabel("Steps")
plt.legend()
plt.show()
```

Traceplots of θ, λ_1, λ_2		
		

(f) Construct density histogram and a table of summary statistics for the approximate posterior distributions of θ, λ1 and λ2

Table of Summary Statistics				
Parameter	Sample Mean	Sample Median	Min	Max
θ	62.32	62	58	111
λ_1	0.73	0.72	0.43	3.55
λ_2	3.67	3.66	0.97	4.91
Histogram of θ, λ_1, λ_2				



5. Unbiased Cross-Validation *(15 points)*

(a) Show that $UCV(h)$ can be simplified into $A+B+C$, where $A = \frac{1}{2nh\sqrt{\pi}}$ and $B = \frac{1}{2n(n-1)h\sqrt{\pi}} \sum_{i=1}^{n} \sum_{j\neq i} exp\left\{-\frac{1}{4h^2}\left(X_i - X_j\right)^2\right\}$.

$K(z) = \frac{1}{\sqrt{2\pi}} exp\left(-\frac{z^2}{2}\right)$ the Gaussian kernel.

$\hat{f}(x) = \frac{1}{nh}\sum_{j=1}^{n} K\left(\frac{x-X_j}{h}\right) = \frac{1}{nh}\sum_{j=1}^{n} \frac{1}{\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{x-X_j}{h}\right)^2\right)$ the kernel density estimate, where h denotes the width of the kernel, often called kernel bandwidth.

$\hat{f}_{-i}(X_i) = \frac{1}{n-1}\sum_{j=1,i\neq j}^{n} K\left(X - X_j\right) = \frac{1}{(n-1)h}\sum_{j=1,i\neq j}^{n} K\left(\frac{X-X_j}{h}\right)$ leave-one-out cross-validation

$R(\hat{f}) = \int \left[\hat{f}(x)\right]^2 dx$

$$= \frac{1}{n^2h^2}\sum_{i=1}^{n}\int K^2\left(\frac{x-X_i}{h}\right)dx + \frac{1}{n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\int K\left(\frac{x-X_i}{h}\right)\times K\left(\frac{x-X_j}{h}\right)dx$$

$$= \frac{1}{n^2h^2}\sum_{i=1}^{n}\int \frac{1}{2\pi}\exp\left\{-\left(\frac{x-X_i}{h}\right)^2\right\}dx + \frac{1}{n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\int \frac{1}{2\pi}\exp\left\{-\frac{1}{2}\left(\frac{x-X_i}{h}\right)^2 - \frac{1}{2}\left(\frac{x-X_j}{h}\right)^2\right\}dx$$

$$= \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\int \exp\left\{-\left(\frac{x-X_i}{h}\right)^2\right\}dx + \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\int \exp\left\{-\frac{1}{2}\left(\frac{x-X_i}{h}\right)^2 - \frac{1}{2}\left(\frac{x-X_j}{h}\right)^2\right\}dx$$

$$= \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\int \exp\left\{-\left(\frac{x-X_i}{h}\right)^2\right\}dx + \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\int \exp\left\{-\frac{(x-X_i)^2 + \left(x-X_j\right)^2}{\left(\sqrt{2}h\right)^2}\right\}dx$$

$$= \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\int \exp\left\{-\left(\frac{x-X_i}{h}\right)^2\right\}dx + \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\int \exp\left\{-\frac{x^2 - x\left(X_i+X_j\right)}{h^2} - \frac{X_iX_j}{h^2} - \frac{\left(X_i-X_j\right)^2}{\left(\sqrt{2}h\right)^2}\right\}dx$$

$$= \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\int \exp\left\{-\left(\frac{x-X_i}{h}\right)^2\right\}dx + \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\exp\left\{-\frac{\left(X_i-X_j\right)^2}{\left(\sqrt{2}h\right)^2} - \frac{X_iX_j}{h^2}\right\}\int \exp\left\{-\frac{x^2 - x\left(X_i+X_j\right)}{h^2}\right\}dx$$

$$= \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\int \exp\left\{-\left(\frac{x-X_i}{h}\right)^2\right\}dx + \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\exp\left\{-\frac{\left(X_i-X_j\right)^2}{2h^2} - \frac{X_iX_j}{h^2}\right\}\int \exp\left\{-\frac{x^2 - x\left(X_i+X_j\right) + \frac{1}{4}\left(X_i+X_j\right)^2 - \frac{1}{4}\left(X_i+X_j\right)^2}{h^2}\right\}dx$$

$$= \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\int \exp\left\{-\left(\frac{x-X_i}{h}\right)^2\right\}dx + \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\exp\left\{-\frac{\left(X_i-X_j\right)^2}{2h^2} - \frac{X_iX_j}{h^2} + \frac{\left(X_i+X_j\right)^2}{4h^2}\right\}\int \exp\left\{-\frac{\left[x - \frac{1}{2}\left(X_i+X_j\right)\right]^2}{h^2}\right\}dx$$

$$= \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}h\sqrt{\pi} + \frac{1}{2\pi n^2h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n}\exp\left\{-\frac{2\left(X_i-X_j\right)^2}{4h^2} - \frac{4X_iX_j}{4h^2} + \frac{\left(X_i+X_j\right)^2}{4h^2}\right\}h\sqrt{\pi}$$

$$= \frac{nh\sqrt{\pi}}{2\pi n^2 h^2} + \frac{h\sqrt{\pi}}{2\pi n^2 h^2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} \exp\left\{-\frac{2X_i^2 - 4X_iX_j + 2X_j^2 + 4X_iX_j - X_i^2 - 2X_iX_j - X_j^2}{4h^2}\right\}$$

$$= \frac{1}{2nh\sqrt{\pi}} + \frac{1}{2n^2h\sqrt{\pi}}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} \exp\left\{-\frac{X_i^2 + X_j^2 - 2X_iX_j}{4h^2}\right\}$$

$$= \frac{1}{2nh\sqrt{\pi}} + \frac{1}{2n^2h\sqrt{\pi}}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} \exp\left\{-\frac{1}{4h^2}\left(X_i - X_j\right)^2\right\} = A + B$$

Note: $\int_{-\infty}^{\infty} \exp\{-ax^2\}\, dx = \sqrt{\pi/a}$

① $u = \frac{x - X_i}{h},\ du = \frac{1}{h}dx \ \Rightarrow \ \int \frac{1}{2\pi}\exp\left\{-\left(\frac{x - X_i}{h}\right)^2\right\}dx = \int \exp\{-u^2\}\, h\, du = h\sqrt{\pi}$

② $u = \frac{x - \frac{1}{2}(X_i + X_j)}{h},\ du = \frac{1}{h}dx \ \Rightarrow \ \int \exp\left\{-\frac{\left[x - \frac{1}{2}(X_i + X_j)\right]^2}{h^2}\right\}dx = \int \exp\{-u^2\}\, h\, du = h\sqrt{\pi}$

(b) Find the functional form of C

$$C = \frac{2}{n}\sum_{i=1}^{n}\hat{f}_{-i}(X_i) = \frac{2}{n(n-1)h}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} K\left(\frac{X_i - X_j}{h}\right)$$

$$= \frac{2}{n(n-1)h}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} \frac{1}{\sqrt{2\pi}}\exp\left\{-\frac{1}{2}\left(\frac{X_i - X_j}{h}\right)^2\right\}$$

$$= \frac{2}{n(n-1)h\sqrt{2\pi}}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} \exp\left\{-\frac{1}{2h^2}\left(X_i - X_j\right)^2\right\}$$

6. Bootstrapping Practice *(Bonus 10 points)*

(a) Show that $E^*(X^*) = \bar{x}$ and $var^*(\bar{X}^*) = \frac{\hat{\mu}^2}{n}$, where $\hat{\mu}_k = \sum_{i=1}^{n}(x_i - \bar{x})^k$.

The asterisk (*) in the formulas indicates that these are estimations of the mean and variance of the bootstrap estimate of the statistic. $X^* = \{X_1^*, \dots, X_n^*\} \sim \hat{F}$ is the bootstrap sample and $\hat{F}$ the empirical distribution of observed data.

$$E^*(X^*) = E\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{1}{n}\sum_{i=1}^{n} E(x_i) = \frac{1}{n}n\bar{x} = \bar{x}$$

$$var^*(X^*) = E[(X^* - EX^*)^2] = \frac{1}{n}\sum_{i=1}^{n} E[(x_i - \bar{x})^2] = \frac{\hat{\mu}^2}{n},\ \text{where } \hat{\mu}^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2$$

(b) By Taylor Series Expansion, find $E^*\left(R(X^*, \hat{F})\right)$ and $var^*\left(R(X^*, \hat{F})\right)$. You can show the first two terms only of the expansion only.

We generate B bootstrap samples by sampling with replacement from the original sample. Then, $R(X^*, \hat{F})$ is the statistic of interest calculated for the original sample, and $R(X_b^*, \hat{F}_b)$ is the statistic of interest calculated for the bth bootstrap sample. We can approximate the mean and variation of the original sample with the following functions:

$$E^*\left(R(X^*, \hat{F})\right) \approx R(X^*, \hat{F}) + \sum_{b=1}^{B}\left[R(X_b^*, \hat{F}_b) - R(X^*, \hat{F})\right]/B$$

$$Var^*\left(R(X^*, \hat{F})\right) \approx \sum_{b=1}^{B}\left[R(X_b^*, \hat{F}_b) - R(X^*, \hat{F})\right]^2/(B - 1)$$