Catam 2.4 Simulation of Random Distributions

February 2023

1 Exponential Distribution

1.1 Median

We define the median m by:

$$\int_0^m f(x|\theta)dx = \frac{1}{2}$$

1.1.1 Q1

A quick integral gives

$$1 - e^{-\theta m} = \frac{1}{2}$$

$$\theta = \frac{log(2)}{m}$$

Hence for $g(x|m) = f(x|\theta(m))$

$$g(x|m) = \frac{log(2)}{m} exp(-\frac{log(2)}{m}x)$$

1.1.2 Q2

$$l_n(m) = log \prod_{i=1}^{n} g(x_i|m)$$

The program "Q2.py" was written to complete this question. It can be found in the Programs section. The results are right below:

u	0.718136	0.240098	0.662383	0.928718	0.181605	0.375651
x	1.055276	0.228805	0.904870	2.200929	0.167008	0.392538

Table 1: Computed values of u and x

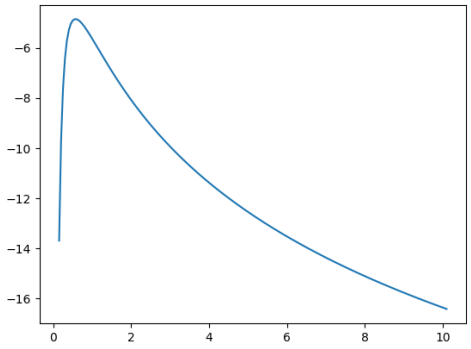

Figure 1: Plot of $l_n(m)$ against m

To derive the true maximum likelihood estimator $\hat{m}_n$ we take derivatives and second derivatives:

$$\frac{\partial l_n(m)}{\partial m} = \sum_{i=1}^{n} \frac{\partial}{\partial m}(log(log(2)) - log(m) - \frac{log(2)}{m}x_i)$$

$$= \sum_{i=1}^{n} -\frac{1}{m} + \frac{log(2)}{m^2}x_i = \frac{1}{m^2}(-nm + log(2)\sum_{i=1}^{n} x_i)$$

So the only stationary point is when $\hat{m}_n = log(2)\bar{X}$, with $\bar{X}$ the mean of x_i. To check this is indeed a maximum

$$\frac{\partial^2 l_n(\hat{m}_n)}{\partial m^2} = \frac{n}{\hat{m}_n^2} - \frac{2log(2)}{\hat{m}_n^3}\sum_{i=1}^{n} x_i = \frac{n}{(log(2)\bar{X})^2}(1-2) < 0$$

Indeed as we know the true mean of an exponential distribution is $\frac{1}{\theta} = \frac{m_0}{log(2)}$ $\hat{m}_n$ will agree with m_0, at least in the limit as $n \to \infty$

1.1.3 Q3

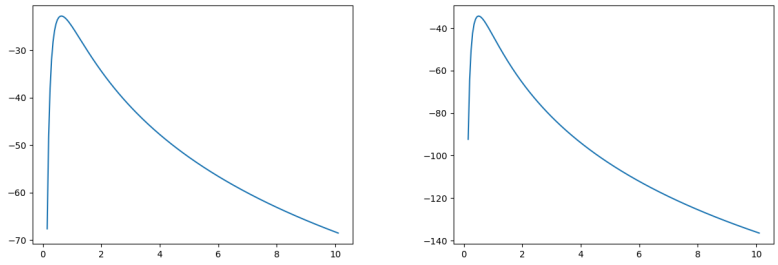

Figure 2: $n = 25, 50$

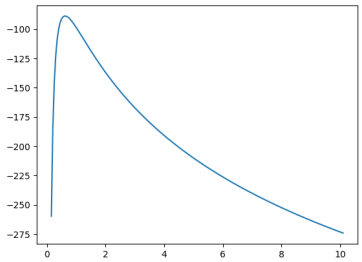

Figure 3: $n = 100$

We note that the approximate shape of the log likelihood functions remains the same up to some translation and scaling on the y axis.

1.2 Gamma Distribution

1.2.1 Q4

Let $X, Y \sim Exp(\theta)$ be independent.

$$M_X(\lambda) = \mathbb{E}(e^{\lambda X}) = \int_0^\infty e^{\lambda x}\theta e^{-\theta x}$$

$$M_X(\lambda) = \frac{\theta}{\theta - \lambda}, (\lambda < \theta)$$

As X, Y independent

$$M_{X+Y}(\lambda) = \mathbb{E}(e^{\lambda(X+Y)}) = \mathbb{E}(e^{\lambda X})\mathbb{E}(e^{\lambda Y}) = \frac{\theta^2}{(\theta - \lambda)^2}, (\lambda < \theta)$$

However a simple inspection shows that this agrees precisely with the MGF of $\Gamma(2, \theta)$ and so $X + Y$ is distributed in that way.

1.2.2 Q5

$$F(x) = \int_0^x \theta^2 y e^{-\theta y} dy = [\theta y(-e^{-\theta y})]_0^x - \int_0^x \theta(-e^{-\theta y})dy$$

$$F(x) = -\theta x e^{-\theta x} + 1 - e^{-\theta x} = 1 - (1 + x\theta)e^{-\theta x}$$

[[IS IT INVERTIBLE (ye prolly not)]]

1.2.3 Q6

Finding the MLE

$$l_n(\theta) = log \prod_{i=1}^n f(x_i|\theta)$$

$$\frac{\partial l_n}{\partial \theta} = \sum_{i=1}^n \frac{\partial}{\partial \theta}(2log(\theta) + log(x_i) - \theta x_i) = \frac{2n}{\theta} - \sum_{i=0}^n x_i$$

$$\hat{\theta} = \frac{2}{\bar{X}}$$

Checking it is a maximum

$$\frac{\partial^2 l_n(\hat{\theta})}{\partial \theta^2} = -\frac{2n}{\hat{\theta}^2} < 0$$

1.2.4 Q7

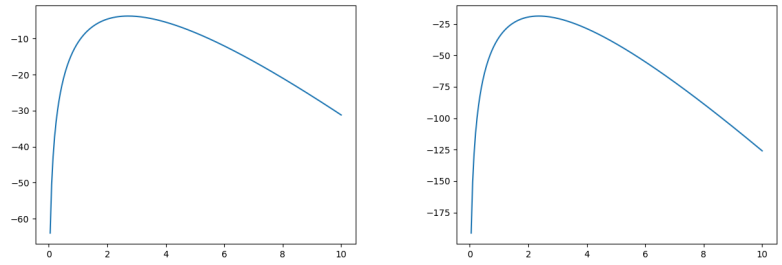

Figure 4: $n = 10,\ 30,\ \hat{\theta} = 2.72039,\ 2.36449$

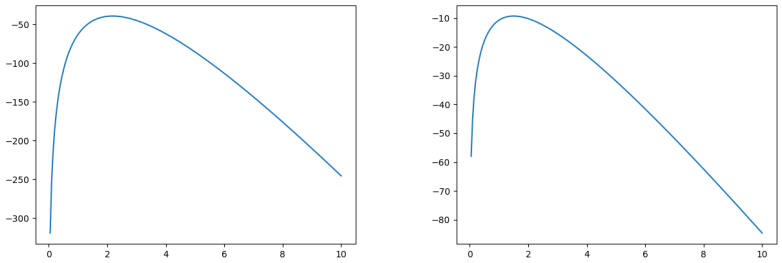

Figure 5: $n = 50, 10,\ \hat{\theta} = 2.18174, 1.50038$

We note that for the first three the shape of the graph appears to be steeper as n increases however that may be an artefact of the MLE being smaller in those cases. Indeed running at $n = 10$ another time has yielded a very steep graph with a low MLE.

We also note a similar effect to Question 3 of scaling upon the y axis as n increases.

1.2.5 Q8

The program for this question is called "Q8.py" and can be found in the Programs section.

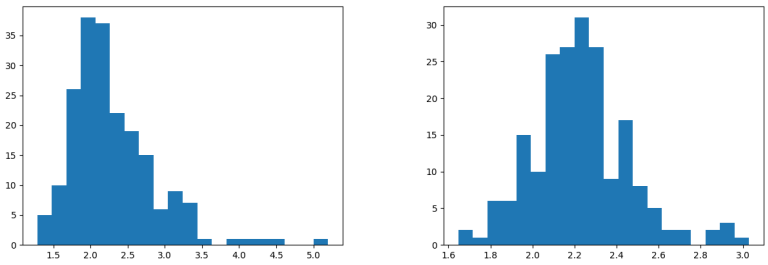

Figure 6: $n = 10, 50$

We note that the variance of the distribution is far less in the $n = 50$ case. This makes sense as we expect the mean squared error to scale proportionally to $\frac{1}{n}$.

2 Normal Distribution

2.1 Normal Distribution

2.1.1 Q9

First let us fully express $f(v, \phi)$, noting that $\mathbf{1}$ is an indicator function

$$f(v,\phi) = \frac{1}{2\pi}\mathbf{1}(\phi \in [0, 2\pi]) \times \frac{1}{2}e^{-\frac{v}{2}}\mathbf{1}(v \geq 0)$$

$$x = \mu_1 + \sigma\sqrt{v}\ cos(\phi),\ \ y = \mu_2 + \sigma\sqrt{v}\ sin(\phi)$$

$$\phi = arctan\frac{y-\mu_2}{x-\mu_1},\ \ v = \frac{1}{\sigma^2}[(x-\mu_1)^2 + (y-\mu_2)^2]$$

We quickly note that the map $(v, \phi) \to (x, y)$ is a bijection simply by considering coordinates about (μ_1, μ_2)

$$f(v(x,y), \phi(x,y)) = \frac{1}{4\pi}exp(-\frac{1}{2\sigma^2}[(x-\mu_1)^2 + (y-\mu_2)^2])\mathbf{1}(-\infty < x, y < \infty)$$

$$|J| = \begin{vmatrix} \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \\ \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} \end{vmatrix} = \begin{vmatrix} \frac{2}{\sigma^2}(x-\mu_1) & \frac{2}{\sigma^2}(y-\mu_2) \\ -\frac{y-\mu_2}{(x-\mu_1)^2+(y-\mu_2)^2} & \frac{x-\mu_1}{(x-\mu_1)^2+(y-\mu_2)^2} \end{vmatrix} = \frac{2}{\sigma^2}$$

This demonstrates that X, Y are independent random variables distributed by $N(\mu_1, \sigma^2), N(\mu_2, \sigma^2)$ respectively.

2.1.2 Q10

A program called "Q10.py" has been written to perform this task, found in the Programs section.

To construct a confidence interval of μ for a sample of size n with $X_i \sim N(\mu, 1)$ IID. A confidence interval are functions $A(x), B(x)$ such that:

$$\mathbb{P}(A(x) \leq \mu \leq B(x)) = 1 - \alpha$$

In this scenario we take $\alpha = 0.2$. Note $\bar{X} \sim N(\mu, \frac{1}{n})$.

$$\mathbb{P}(\Phi(0.1) \leq \sqrt{n}(\bar{X} - \mu) \leq \Phi(0.9)) = 0.8$$

Writing $\Phi(0.9) = z^*$.

$$\mathbb{P}(-z^* \leq \sqrt{n}(\bar{X} - \mu) \leq z^*) = 0.8$$

$$\mathbb{P}(\bar{X} - \frac{z^*}{\sqrt{n}} \leq \mu \leq \bar{X} + \frac{z^*}{\sqrt{n}}) = 0.8$$

$$A(x) = \bar{X} - \frac{z^*}{\sqrt{n}},\ \ B(x) = \bar{X} + \frac{z^*}{\sqrt{n}}$$

2.1.3 Q11

The program for this question is called "Q11.py" and found in the Programs section. The table of results can be found in the Tables section.

Reading off the data from the table we find that we reject 7 times in 25. Which is close to the expected value of 5.

2.1.4 Q12

With $n = 50, \mu = 4$ we construct a confidence interval of 80%, we would expect the confidence interval to not contain μ 10 times.

$$\mathbb{E}(\sum_{i=1}^{50} \mathbf{1}(\mu \notin C)) = \sum_{i=1}^{50} \mathbb{E}(\mathbf{1}(\mu \notin C)) = \sum_{i=1}^{50} \mathbb{P}(\mu \notin C) = 50 * 0.2 = 105$$

2.2 Chi Squared Distribution

2.2.1 Q13

A program called "Q13.py" has been written to generate the histograms for this question, it can be found in the Programs section.

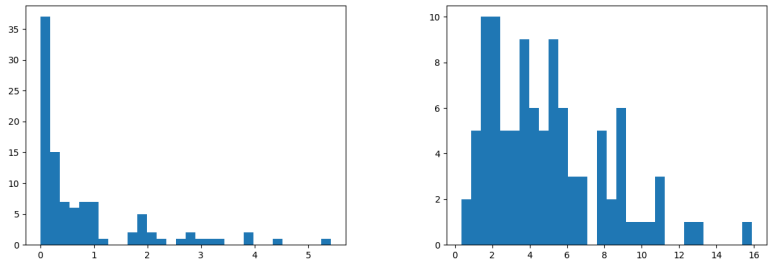

Figure 7: $n = 100$ of χ_1^2, χ_5^2

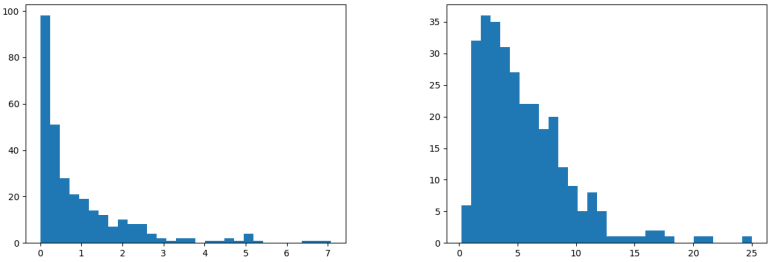

Figure 8: $n = 300$ of χ_1^2, χ_5^2

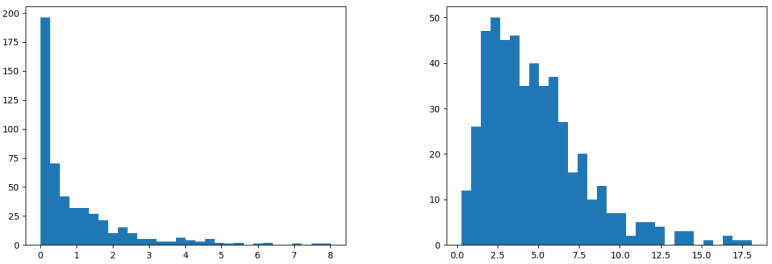

Figure 9: $n = 500$ of χ_1^2, χ_5^2

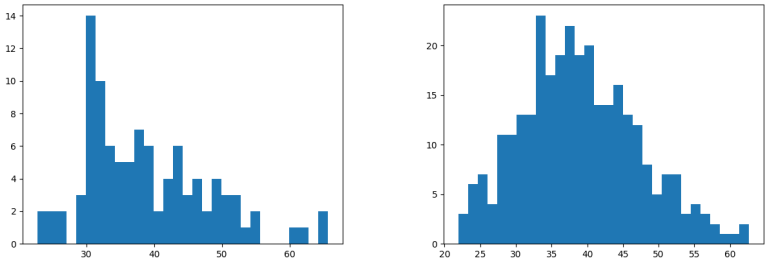

Figure 10: $n = 100, 300$ of χ^2_{40}

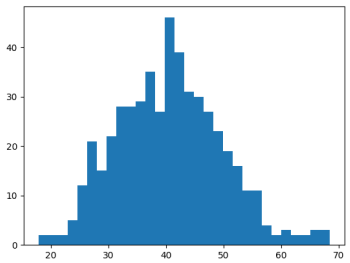

Figure 11: $n = 500$ of χ^2_{40}

We note that for χ^2_k where $k = 1$ we have an asymptotic behaviour at 0 which lines up with the distribution we expect, being $\Gamma(1/2, 1/2)$. We also see that for $k > 1$, both the mean and the variance of the distribution increases which can be easily seen by examining the scaling of the x axis in the $k = 5, 40$ cases.

3 Tables

Sample mean	Lower Bound	Upper Bound	Accepted
-0.014533411410052375	-0.14253341141005238	0.11346658858994763	1
-0.06468712115034685	-0.19268712115034686	0.06331287884965316	1
0.14584866386813833	0.017848663868138326	0.27384866386813833	0
0.0055490506628591	-0.1224509493371409	0.1335490506628591	1
0.03317109443865838	-0.09482890556134163	0.16117109443865837	1
-0.07239737980925601	-0.200397379809256	0.055602620190743995	1
-0.27344378100294014	-0.40144378100294015	-0.14544378100294014	0
0.10431990844737647	-0.02368009155262353	0.2323199084473765	1
-0.06286855681783832	-0.19086855681783832	0.06513144318216169	1
0.033826449569032135	-0.09417355043096787	0.16182644956903214	1
0.005683099316335965	-0.12231690068366403	0.13368309931633596	1
0.04155241693904463	-0.08644758306095537	0.16955241693904463	1
-0.12708889507605198	-0.255088895076052	0.0009111049239480251	1
0.05999347466652258	-0.06800652533347742	0.18799347466652258	1
0.13529700399264535	0.007297003992645351	0.26329700399264533	0
0.03544049798659528	-0.09255950201340472	0.16344049798659527	1
-0.13126691906392074	-0.25926691906392074	-0.003266919063920737	0
-0.19193943233153127	-0.31993943233153127	-0.06393943233153127	0
-0.15994821477905963	-0.28794821477905963	-0.031948214779059625	0
-0.011744357029184646	-0.13974435702918464	0.11625564297081535	1
0.10246880505547758	-0.025531194944522426	0.23046880505547757	1
0.060277440833894774	-0.06772255916610523	0.18827744083389478	1
0.0846247134351084	-0.043375286564891605	0.21262471343510841	1
0.13922199572135624	0.011221995721356237	0.26722199572135624	0
0.04275659379183149	-0.0852434062081685	0.1707565937918315	1

Table 2: Data for Q11

4 Programs

4.1 Q2.py

```
import matplotlib.pyplot as plt
import numpy as np
import random

n = 100
theta0 = 1.2


u = [random.uniform(0,1) for i in range(n)]
x = [0 for i in range(n)]

for i in range(n):
    x[i] = -np.log(1-u[i])/theta0



def g (x, m):
    return ( np.log(2) / m) * np.exp(- np.log(2) * x / m )

def getProd (x, m, n):
    prod = 1
    for i in range(n):
        prod *= g(x[i], m)

    return prod


m = [0.05*(i+3) for i in range(200)]

y = [ np.log(getProd(x, m[i], n)) for i in range(200)]

plt.plot(m, y)
plt.show()


print(u)
print(x)
```

4.2 Q7.py

```
import matplotlib.pyplot as plt
import numpy as np
import random

n = 10
t0 = 2.2


u = [random.uniform(0,1) for i in range(n)]
x = [0 for i in range(n)]

def F (x, t):
    return 1 - (1 + x*t) * np.exp(-t * x)

def findX (u, i, uBound, lBound, t): # Function which attempts to find x given u by the bise
    m = (uBound + lBound) / 2
    uNew = F(m, t)

    if (i <= 0): return m

    if (uNew > u):
        return findX( u, i-1, m, lBound, t)
    elif (uNew < u):
        return findX( u, i-1, uBound, m, t)
    else:
        return m

for i in range(n):
    x[i] = findX(u[i], 50, 20, 0, t0)



def f (x, t):
    return t * t * x * np.exp(- t * x)

def getLog (x, t, n):
    S = 1
    for i in range(n):
        S += np.log(f(x[i], t))

    return S


t = [0.05*(i+1) for i in range(200)]
```

```
y = [ getLog(x, t[i], n) for i in range(200)]

print(2*n / sum(x, 0))

plt.plot(t, y)
plt.show()
```

4.3 Q8.py

```
import matplotlib.pyplot as plt
import numpy as np
import random

n = 10
N = 200
t0 = 2.2


def F (x, t):
    return 1 - (1 + x*t) * np.exp(-t * x)

def findX (u, i, uBound, lBound, t): # Function which attempts to find x given u by the bise
    m = (uBound + lBound) / 2
    uNew = F(m, t)

    if (i <= 0): return m

    if (uNew > u):
        return findX( u, i-1, m, lBound, t)
    elif (uNew < u):
        return findX( u, i-1, uBound, m, t)
    else:
        return m



def f (x, t):
    return t * t * x * np.exp(- t * x)

def getLog (x, t, n):
    S = 1
    for i in range(n):
        S += np.log(f(x[i], t))

    return S

data = [0 for i in range(N)]

for j in range(N):
    u = [random.uniform(0,1) for i in range(n)]
    x = [0 for i in range(n)]

    for i in range(n):
```

```
        x[i] = findX(u[i], 50, 20, 0, t0)

    data[j] = 2*n / sum(x, 0)


plt.hist(data, bins = 20)
plt.show()
```

4.4 Q10.py

```
import numpy
import random

def generateNormal (mu, sigma):
    A = random.uniform(0,1)
    B = random.uniform(0,1)

    U = 2*numpy.pi*A
    V = -2 * numpy.log(1-B)

    X = mu + sigma*numpy.sqrt(V)*numpy.cos(U)
    Y = mu + sigma*numpy.sqrt(V)*numpy.sin(U)

    return X, Y


def generateN (n, mu, sigma):
    x = [0 for i in range(n)]
    for i in range(n):
        x[i] = generateNormal(mu, sigma)[0]

    return x
```

4.5 Q11.py

```
import numpy
import Q10

z = 1.28
n = 100

tot = 0

for i in range(500):
    x = Q10.generateN(n, 0, 1)
    S = 0
    for j in range(n):
        S += x[j]
    mean = S/n

    A = mean - z/numpy.sqrt(n)
    B = mean + z/numpy.sqrt(n)

    oz = 1
    if (A > 0): oz = 0
    if (B < 0): oz = 0

    tot += oz

    #print(str(mean) + " & " + str(A) + " & " + str(B) + " & " + str(oz))


print(tot)
```

4.6 Q13.py

```
import matplotlib.pyplot as plt
import Q10

def GenerateNChi(n, k):
    x = [0 for i in range(n)]
    for j in range(n):
        for i in range(k):
            y = Q10.generateNormal(0, 1)[0]
            x[j] += y*y

    return x

def DrawGraph(n, k):
    x = GenerateNChi(n, k)
    plt.hist(x, bins = 30)
    plt.show()

DrawGraph(100, 1)
DrawGraph(300, 1)
DrawGraph(500, 1)

DrawGraph(100, 5)
DrawGraph(300, 5)
DrawGraph(500, 5)

DrawGraph(100, 40)
DrawGraph(300, 40)
DrawGraph(500, 40)
```