Part A: In-class assignment:
(a)Generate a random sample of size 500 from Gamma (2,3),where we assume alpha=2 is fixed and beta is to be estimated. This is your raw data.
(b)Use the Newton-Raphson method to obtain the MLE for beta. Compare the estimate with the MME for beta. (see the note for the MME)
(c)Plot the log likelihood over the sample space for band indicate the location of the MLE in the plot.

Steps for coding:

The prompt asked us to generate a random sample of size 500 from Gamma(2,3), so I planned on using gamrnd(2,3,500,1). Next, I chose my arbitrary beta0 to equal 1.5 for this sample run, but I am planning on seeing how it works with other values before submission. The Score vector is much simpler in this case because of the constant alpha. Continuing on from the notes, we obtained the values $U(\beta) = -\frac{2n}{\beta_0} + \frac{\sum x_i}{\beta_0^2}$ and $I(\beta) = \frac{2n}{\beta_0}$. These are going to be used for the Newton-Raphson Method. The MME from the notes will be used to find beta as well, and, although it is not required, the built-in MLE function will also used for observation purposes. Afterwards, the plot will be created.

Code:

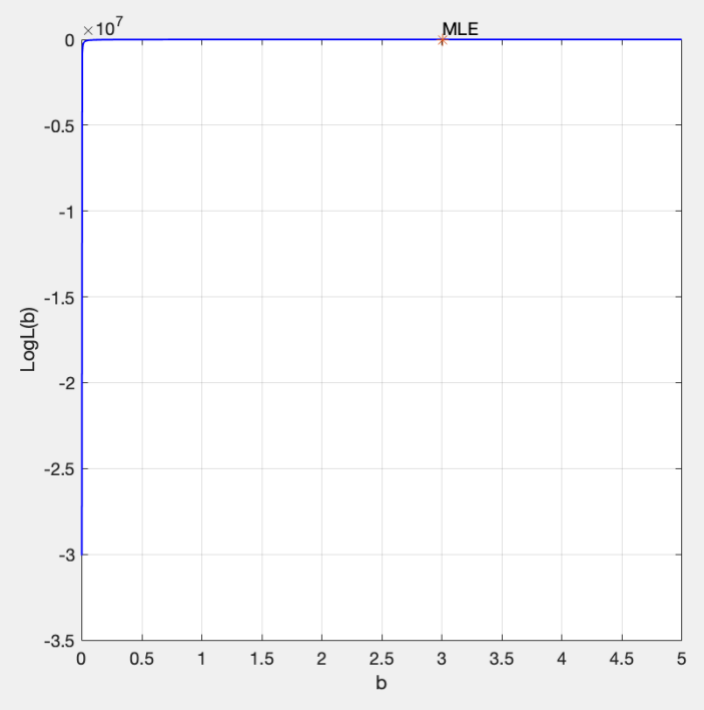
```
clc, clear

% Part A, generating 500 gamma(2,3) samples
r = gamrnd(2, 3, 500, 1);
n = length(r);
mu = mean(r);
var = var(r);

% Part B
% Assigning fixed alpha0 and chosen beta0
% alpha0 = 2;
beta0 = 1.5;

%Finding Information matrix and Score vector
Io = [2*n/beta0^2]
Uo = [-2*n/beta0 + sum(r)/beta0^2]


iter = 1; % Assigning initial iteration count

fprintf('-----------------------------------------------------\n')
fprintf('\nUsing Newton-Raphson method, the MLE for beta = %1.4f\n',...
    beta0)
beta= beta0 + Io \ Uo;
fprintf('Iteration count is at %d\n', iter)
while ( abs(beta-beta0) > 0.0001 )
    beta0=beta;
    Io = [2*n/beta0^2];
    Uo = [-2*n/beta0 + sum(r)/beta0^2];

    beta=beta0 + Io \ Uo;
```

```
    fprintf('\nUsing Newton-Raphson method, the MLE for beta = %1.4f\n',...
        beta)
    iter=iter+1;
    fprintf('Iteration count is at %d\n', iter)
end;

% MLE function
fprintf('-----------------------------------------------------\n')
mlethetahat=mle('gamma',r);
fprintf('\nUsing the MLE function, alpha = %1.4f and beta = %1.4f\n', ...
    mlethetahat(1), mlethetahat(2))

% Estimating MME for beta
fprintf('\n-----------------------------------------------------\n')

m2=(1/n)*sum(r.^2);
MME_beta = (m2-mu^2)/mu;
fprintf('\nUsing the MME from notes, beta = %1.4f\n', MME_beta)

fprintf('\n-----------------------------------------------------\n')

% Part C
b = 0:0.0001:5;
logL = inline('-2*n*log(b)+sum(log(r))-sum(r)./b', 'n', 'b', 'r');

y = logL(n, b, r);
plot(b, y, 'b','LineWidth',1), hold on

[max, loc]=max(y);
bhat=b(loc)

plot(bhat, max,'*'), hold off
axis square
text(bhat,max,
'MLE','VerticalAlignment','bottom','HorizontalAlignment','left')

line([bhat bhat], [-2 max], 'Color','red','LineStyle',':','LineWidth',1.75)
grid, xlabel('b'), ylabel('LogL(b)')
```

Output:

```
Io =

  444.4444


Uo =

  669.8726

-----------------------------------------------------

Using Newton-Raphson method, the MLE for beta = 1.5000
Iteration count is at 1
```

```
Using Newton-Raphson method, the MLE for beta = 3.0072
Iteration count is at 2
-----------------------------------------------------

Using the MLE function, alpha = 1.8957 and beta = 3.1726

-----------------------------------------------------

Using the MME from notes, beta = 3.0036

-----------------------------------------------------

bhat =

    3.0072
```



Discussion:

As we saw in class, the Newton-Raphson method is great for finding MLE of a probability distribution function. It is most useful for doing so without having to do any handwritten computation, but rather a repetitive loop on a strong programming compiler, such as MATLAB. Using the gamma random function, we first defined it in the gamrnd(2, 3…), where we said alpha = 2 and beta = 3. We can see that the betahat was successfully estimated to be approximately 3 from out arbitrary input of beta0 = 1.5. Once again, this was a success. Comparing to the MLE found in the function, we also receive a value very close to 3 there. This allowed us to verify that we are indeed attaining the appropriate value. Next, we wanted to look at the MME equation(s) from the notes and we were, once again, successful in attaining a value near beta = 3. I attempted to adjust the intervals for the plot several times, but it kept on erasing

the plot. I adjusted the thickness of the graph, but I was still not successful. However, the MLE is still represented by the asterisk. Therefore, it is still an appropriate graph.

Rene Zamudio (008634559) – STAT 572 – Dr. Sung Kim – Homework 1

Part B: Textbook problems: Chapter 3: #1, 5, 6, 9, 16

Exercise 3.1 Pg 80

Generate 500 random samples from the standard normal distribution for sample sizes of *n* = 2, 15, and 45. At each sample size, calculate the sample mean for all 500 samples. How are the means distributed as *n* gets large? Look at a histogram of the sample means to help answer this question. What is the mean and variance of the sample means for each *n*? Is this what you would expect from the Central Limit Theorem? Here is some MATLAB code to get you started.

For each *n*:

```
% Generate 500 random samples of size n: x = randn(n, 500);
% Get the mean of each sample:
xbar = mean(x);

% Do a histogram with superimposed normal density. % This function is in the
MATLAB Statistics Toolbox. % If you do not have this, then just use the
% function hist instead of histfit.
histfit(xbar);
```

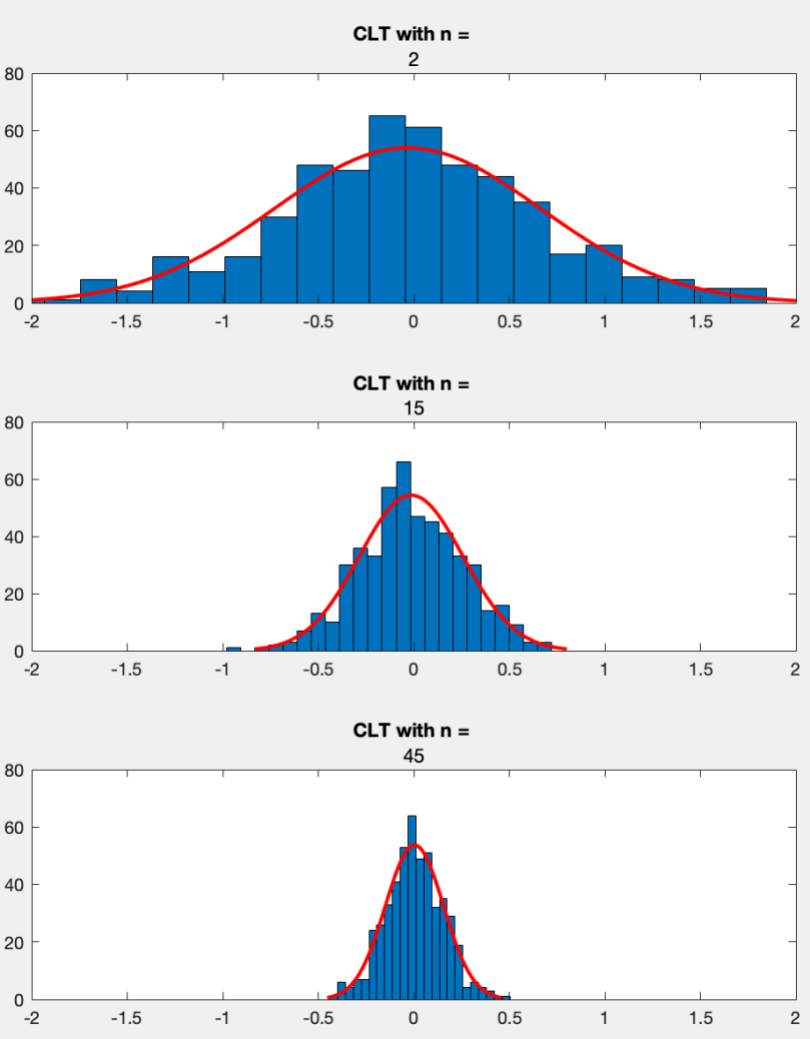
Steps for coding:

The prompt specifically asked for the sample sizes of n = 2, n = 15, and n = 45. I decided to make that into a 1x3 vector and then proceeded to work with each one individually using randn, mean, and histfit. Once I saw that everything was working properly, I decided to make it into a loop to shorten the code.

Code:

```
% Assigning/storing each sample size
n = [2 15 45];

% Create figure.
figure();

% Calculating the mean for each sample size
% and creating histfit
% Decided to make a loop to shorten the code
for i = 1:3
    x = randn(n(i),500);
    xbar = mean(x);
    subplot(3,1,i)
%   hist(xbar)
    histfit(xbar)
    xlim([-2 2])
    title('CLT with n = ', n(i))
end
```

Output:



Discussion:

We can clearly see that as our sample size increases, we are estimating closer to the standard Normal Distribution mean of 0. This helps reinforce the Central Limit Theorem. It is clear that if we continue to increase the sample size, we will continue to see a much narrower bell-shape since the plots are specified to display the interval [-2 2]. I chose to stay consistent with this interval to help emphasize the differences.

Exercise 3.5 Pg 80

Repeat example 3.1 using samples generated from a standard normal distribution. You can use the MATLAB function **`randn`** to generate your samples. What happens to the coefficient of skew- ness and kurtosis as the sample size gets large?

Steps for coding:

The prompt asks us to essentially repeat exercise 3.1, but not entirely. We are allowed to generate our own randn and sample sizes. I chose to use sample sizes that increased from 5 to 1,000,000 in no particular order. I wanted to see how much kurtosis and skewness changed as the sample sizes increase. Once I saw that everything was working properly, I decided to make it into a loop to shorten the code.

Code:

```
% the sample sizes:
n = [5 100 1000 10000 1000000];

% creating loop to shorten code
for i=1:5
    x = randn(n(i),1); %randn w/ diff n sizes
    xbar = mean(x);
    kurt = cskurtosis(x); %function from Toolbox
    skew = cskewness(x); %function from Toolbox
    fprintf('For n = %d, Kurtosis = %1.5f and Skewness = %2.5f \n',...
        n(i),kurt,skew);
end
```

Output:

```
For n = 5, Kurtosis = 2.72303 and Skewness = 0.91563
For n = 100, Kurtosis = 2.78830 and Skewness = -0.08316
For n = 1000, Kurtosis = 3.11417 and Skewness = -0.06557
For n = 10000, Kurtosis = 3.02897 and Skewness = -0.00466
For n = 1000000, Kurtosis = 2.99355 and Skewness = 0.00034
```

Discussion:

We can clearly see that as our sample size increases, we are seeing less Skewness or outliers as it gets closer to 0. Additionally, the Kurtosis is approaching the value of 3, which is appropriate for the standard Normal Distribution. Both of these results help support Exercise 3.1 and the Central Limit Theorem. It is clear that if we continue to increase the sample size, we will continue to progress Skewness towards 0 and Kurtosis towards 3.

Exercise 3.6 Pg 80

Generate a random sample that is uniformly distributed over the interval (0, 1). Plot the empirical distribution function over the interval (-0.5, 1.5). There is also a function in the Statistics Toolbox called cdfplot that will do this.

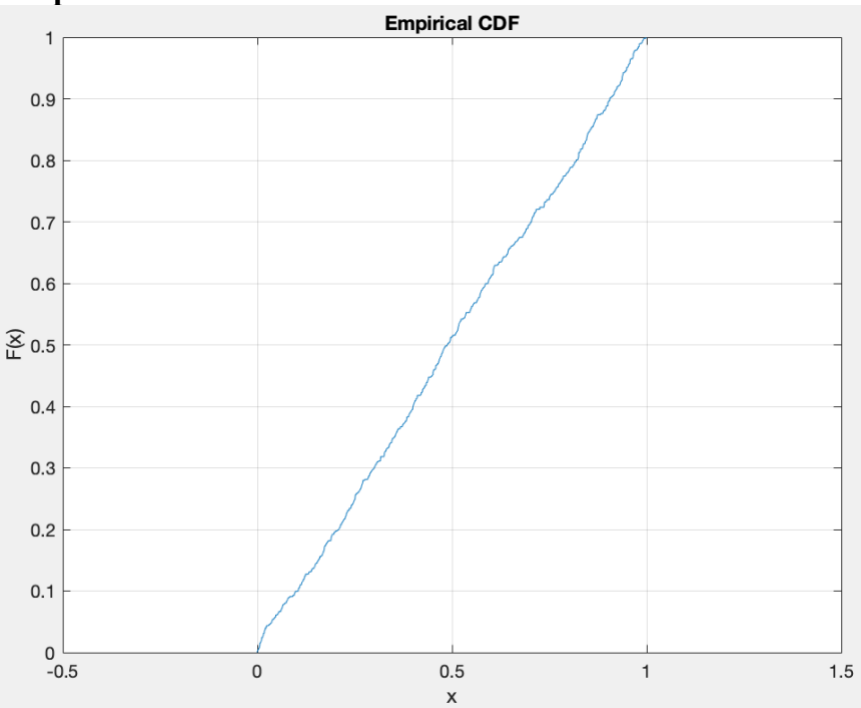
Steps for coding:

The prompt asks us to generate a random sample U(0,1) and to plot the CDF with the given interval (-0.5, 1.5). The interval looks appropriate since it is expected that the CDF plot should end at 1 and appear linear. Since we only have to generate one sample size, I chose to use n = 572, since we are in STAT 572. I then used the suggested function [cdfplot] from the Statistics Toolbox and creating a limit for the appropriate interval.

Code:

```
% rand will generate random values from 0 to 1 (uniform)
x = rand(1,572); % 572 for n (stat 572)

% using cdfplot from Toolbox and setting x interval
cdfplot(x);
xlim([-0.5 1.5]);
```

Output:



Discussion:

As anticipated, our CDF plot goes from F(x)=0 to F(x)=1 in an almost linear pattern. This is because of the almost constant behavior of the uniform distribution for our sample size of 572. The cdfplot function worked great and I will definitely be using this and other of the Statistics Toolbox functions. The interval limit for x (-0.5, 1.5) also worked nicely.

Exercise 3.9 Pg 81

Generate a random sample of size 100 from a uniform distribution (use the MATLAB function rand to generate the samples). What are the sample quantiles for p = 0.33, 0.40, 0.63, 0.90 ? Is this what you would expect from theory?

Steps for coding:

The prompt asks us to generate a random sample of size 100 (uniform). Then, we are to find the sample quantiles for P= 0.33, 0.40, 0.63, 0.90. My initial thoughts are that it will make things much easier if the samples were sorted. Since we are asked to use n = 100, we can simply choose the values in the corresponding indexes that are equal to P*100. For example, the quantile for P = 0.33 should be located at index 33. If the sample size were different, then we would be required to do a little bit more computation.

Code:

```
% generate u(0,1) r.s. of n=100 (ordered)
x  = sort(rand(1,100));

% finding quantiles based on indexes
p = [0.33 0.4 0.63 0.9];
q = [x(33), x(40), x(63), x(90)];

% printing quantiles
for i=1:4
    fprintf('The sample quantile for P = %1.2f is %1.4f \n', p(i),q(i));
end
```

Output:

```
The sample quantile for P = 0.33 is 0.3333
The sample quantile for P = 0.40 is 0.4168
The sample quantile for P = 0.63 is 0.6355
The sample quantile for P = 0.90 is 0.9058
```

Discussion:

For this exercise, the output looks appropriate to what we expect from the theory. Since the random samples were generated from a uniform distribution, we could predict that there should be evenly spread out values from 0 to 1, especially when ordered using the [sort] function. We can see that the sample quantiles for P = 0.33, 0.40, 0.63, 0.90 are close in value to the respective sample quantile values: 0.3333, 0.4168, 0.6355, 0.9058. The randomly generated numbers were nearly all the similar to the hundredth decimal place.

Exercise 3.16 Pg 81

Investigate the bias in the maximum likelihood estimate of the variance that is given in equation 3.28. Generate a random sample from the standard normal distribution. You can use the randn function that is available in the standard MATLAB package. Calculate σˆ2 using equation 3.28 and record the value in a vector. Repeat this process (generate a random sample from the standard normal distribution, estimate the variance, save the value) many times. Once you are done with this procedure, you should have many estimates for the variance. Take the mean of these estimates to get an estimate of the expected value of σˆ 2 . How does this compare with the known value of σ2 = 1 ? Does this indicate that the maximum likelihood estimate for the variance is biased? What is the estimated bias from this procedure?

Steps for coding:

The prompt is asking us to refer to equation 3.28 on page 70 of the textbook, $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$. The equation proves to not be unbiased as shown in the written work below. Thus, we can expect for us to see differences between equation 3.28 and the [var(x)] function in MATLAB, which is also equivalent to s^2 shown below as well. I chose to use n = [5 10 50 100 500]. As n → ∞, $\frac{n-1}{n}$ will essentially equal 1. Therefore, it is clear that the biggest differences will be found in smaller sample sizes. If I chose to use too many big sample sizes, then the averages of the two equations being compared will be more identical. I chose to use a loop to make the code shorter and I also decided to calculate [var(x)], sigmahat^2 (equation 3.28), and s^2 (to verify that we receive identical values to the variance function). As requested in the prompt, I found the averages of [var(x)] and sigmahat^2 to see which is closer to 1, which is the variance of the standard Normal Distribution.

$E(\hat{\sigma}^2) = \sigma^2$?

$E(\hat{\sigma}^2) = E\left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right] = \frac{1}{n}E\left[\sum_{i=1}^{n}(x_i - \bar{x})^2\right]$

$= \frac{1}{n}E\left[\sum_{i=1}^{n}(x_i^2 - 2x_i\bar{x} + \bar{x}^2)\right]$

$= \frac{1}{n}E\left[\sum_{i=1}^{n}x_i^2 - 2\bar{x}\sum_{i=1}^{n}x_i + n\bar{x}^2\right]$

$= \frac{1}{n}E\left[\sum_{i=1}^{n}x_i^2 - 2n\bar{x}\cdot\bar{x} + n\bar{x}^2\right]$

$= \frac{1}{n}E\left[\sum_{i=1}^{n}x_i^2 - 2n\bar{x}^2 + n\bar{x}^2\right]$

$= \frac{1}{n}E\left[\sum_{i=1}^{n}x_i^2 - n\bar{x}^2\right] = \frac{1}{n}E\left[\sum_{i=1}^{n}x_i^2\right] - E[\bar{x}^2]$

$= \frac{1}{n}\sum_{i=1}^{n}E[x_i^2] - E[\bar{x}^2] = \frac{n}{n}E[x^2] - E[\bar{x}^2]$ $= \mu$

$\sigma^2 = E[x^2] - E[x]^2$ $\quad \sigma_{\bar{x}} = E[\bar{x}^2] - E[\bar{x}]^2$

$E[x^2] = \sigma^2 + \mu^2$ $\quad E[\bar{x}^2] = \sigma_{\bar{x}} + \mu^2$

$= \sigma^2 + \mu^2 - \sigma_{\bar{x}} - \mu^2 = \sigma^2 - \sigma_{\bar{x}}^2 = \sigma^2 - \frac{\sigma^2}{n}$

$= \sigma^2\left(1 - \frac{1}{n}\right) = \sigma^2\left(\frac{n-1}{n}\right)$

Not unbiased since $E[\hat{\sigma}^2] \neq \sigma^2$

Therefore, $\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ (page 71)

Code:

```
% creating n
n = [5 10 50 100 500];

% n samples and mean
for i=1:length(n)
    x = randn(1,n(i));
    xbar = mean(x);
    varx(i) = var(x); % finding variance using MATLAB function
    sighat2(i) = 1/n(i)*sum((x-xbar).^2); % using equation 3.28
    s2(i) = 1/(n(i)-1)*sum((x-xbar).^2); % should be equal to varx
    %   Exp = (n(i)-1)/n(i)*varx
    fprintf('At n = %4.f, var(x)=%1.4f, sigmahat^2=%1.4f, s^2=%1.4f \n', ...
        n(i), varx(i), sighat2(i), s2(i))
end
%print average
fprintf('\nThe average of var(x) = %1.4f \n', mean(varx))
fprintf('The average of sigmahat^2 = %1.4f \n', mean(sighat2))
```

Output:

```
At n =    5, var(x)=0.8758, sigmahat^2=0.7006, s^2=0.8758
At n =   10, var(x)=0.7490, sigmahat^2=0.6741, s^2=0.7490
At n =   50, var(x)=1.3768, sigmahat^2=1.3492, s^2=1.3768
At n =  100, var(x)=0.9859, sigmahat^2=0.9760, s^2=0.9859
At n =  500, var(x)=1.0561, sigmahat^2=1.0540, s^2=1.0561

The average of var(x) = 1.0063
The average of sigmahat^2 = 0.9648
```

Discussion:

For this exercise, the output looks appropriate to what we anticipated. We confirmed that the MATLAB variance function is identical to s^2 that we found. We also confirmed that with smaller sample sizes, we will see more difference in the variance and the equation 3.28, which is not unbiased. Overall, the average of the variance function is closer to 1, than the average of the sigmahat^2 equation (eq 3.28). As mentioned this was anticipated and we showed that making the equation unbiased gave us s^2, which is also equal to the values provided by the variance function. Thus, the variance function was expected to have an average closer to 1.