University of Zurich

Department of Banking and Finance

03SM22MO0103: Statistical Foundations for Finance (Mathematical and Computational Statistics with a View Towards Finance and Risk Management)

Take Home Assignment 1

Submitted by:
Azamat Zhaksylykov 23-745-664
Nikolai Kurlovich 23-743-412
Ishaan Bhondele 23-745-862

Contents

Thursday 9th November, 2023

Problem 1

Code Walkthrough

We would first like to explain supplementary functions and tools that we used. We first start with the KD-test. Here we fit the distribution and directly compute the relevant statistic. One small point here is that we try to vectorize things as much as possible and avoid using for-loops in order to get better performance.

```
function out = kd(data)
    n = length(data);
    sorted = sort(data);
    para = tlikmax(data, [7, 1 , 2]);
    if para(1) > 0
        degr = round(para(1));
    else
        degr = 2;
    end

    F_emp = ((1:n) - 0.375)/(n + 0.25);
    F_fit = tcdf((sorted - para(2))/para(3), degr)';
    out = max(abs(F_emp - F_fit));
end
```

The part where we manually make sure that our degrees of freedom are non-negative becomes important in the later stages when we proceed to power simulations. There, as we sample from a distribution different from t, our MLE fitting algorithm sometimes produces nonsensical values for degrees of freedom (which makes sense as our data doesn't come from t-distribution), so we clip all these negative values at 1. The same procedure is done in the code for AD-test which is shown below.

```
function out = anderson(data)
    n = length(data);
    sorted = sort(data);
    para = tlikmax(data, [7, 1 , 2]);

    X = (sorted - para(2)) / para(3);
    if para(1) > 0
        degr = round(para(1));
    else
        degr = 2;
    end
    X_flip = flip(X);
    S = sum((2*(1:n)' - 1)./n .* (log(tcdf(X, degr)) + log(1 - tcdf(X_flip,
        (degr)))));
    out = - n - S;
end
```

In both cases we needed to fit our data to a location-scale t-distribution, For that, we utilized the code made by our professor and mentioned in chapter 2 of our statistics book.

```
function MLE = tlikmax(x, initvec)
    tol = 1e-5;
    opts = optimset ('Disp' , 'none' , 'LargeScale' , 'Off' , ...
    'TolFun' , tol , 'TolX' , tol , 'MaxIter', 200) ;
    MLE = fminunc(@(param) tloglik(param, x), initvec , opts) ;
end

function ll = tloglik(param, x)
    v = param(1) ; mu = param(2); c = param(3) ;
        if v<0.01 , v = rand; end
        if c<0.01 , c = rand; end
```

```
    K=beta (v/2 , 0.5 ) * sqrt(v) ; z=(x-mu) / c ;
    ll = -log (c) -log (K) - (( v+1) /2) * log (1 + ( z .^2) / v ) ; ll = -
        sum(ll ) ;
end
```

Also, for the second part of this problem we needed to be able to generate Laplace random variables. There are different ways how this can be accomplished. The way we employ here, however, utilizes the fact that $\log(U_1/U_2) \sim \text{Laplace}(0,1)$ where $U_i \sim \text{Unif}(0,1)$.

```
function x = randlap(m, n)
    u1 = rand(m, n);
    u2 = rand(m, n);

    x = log(u1./u2);
    x = bsxfun(@minus, x, mean(x));
    x = bsxfun(@rdivide, x, std(x));
    end
```

Now let us explain the core part of the code that is used to produce these results. First, we are running a loop that covers all possible parameter combinations that we are interested in.

```
for signif = [0.10, 0.05, 0.01]
    for sample_size = [30, 60, 100]
        for degrees_of_freedom = [3, 6, 12]
```

As explained in the problem, we first want to determine cutoff-values. This is done directly via simulation. 10000 times we create a random sample, compute relevant statistics based on it and once this is done we take $(100-\alpha)$-th percentile.

```
rng(shuffle);
runs = 10000;
stats_ad = zeros(runs, 1);
stats_kd = zeros(runs, 1);
for j = 1:runs
    data = location + scale * trnd(degrees_of_freedom, sample_size, 1);
    stats_kd(j) = kd(data);
    stats_ad(j) = anderson(data);
end
cutoff_kd = prctile(stats_kd, 100*(1 - signif));
cutoff_ad = prctile(stats_ad, 100*(1 - signif));
fprintf('alpha = %.2f, size = %d, df = %d, KD cutoff = %.3f\n', signif,
   sample_size, degrees_of_freedom, cutoff_kd);
fprintf('alpha = %.2f, size = %d, df = %d, AD cutoff = %.3f\n', signif,
   sample_size, degrees_of_freedom, cutoff_ad);
```

Now that this is done, we need to verify that these cutoff values make sense. For this, we compute the size of the relevant tests. For that, we draw samples, compute relevant statistics and determine whether our null would be rejected or not. It is important to note that we reset the random generation seed to make sure that we are not basically generating exact same samples that we used to construct those cutoff values. This is done in the first line of code below.

```
rng(shuffle);
runs = 10000;
rej_kd = 0;
rej_ad = 0;
for j = 1:runs
    data = location + scale * trnd(degrees_of_freedom, sample_size, 1);

    if kd(data) > cutoff_kd
        rej_kd = rej_kd + 1;
    end
```

```
    if anderson(data) > cutoff_ad
        rej_ad = rej_ad + 1;
    end
end
fprintf('alpha = %.2f, size = %d, df = %d, SIZE KD = %.3f\n', signif,
    sample_size, degrees_of_freedom, rej_kd / runs);
fprintf('alpha = %.2f, size = %d, df = %d, SIZE AD = %.3f\n', signif,
    sample_size, degrees_of_freedom, rej_ad / runs);
```

Next we are interested in determining the power of these tests. For that, we take some other distribution as our true distribution and via simulations observe how often we would reject the null hypothesis. We first do it for the case when the true distribution is assumed to be normal. Things are quite straightforward here since matlab has all the required functions to make sampling from it as easy as possible.

```
rng(shuffle);
runs = 5000;
rej_kd = 0;
rej_ad = 0;
for j = 1:runs
    data = location + scale * normrnd(sample_size, 1);

    if kd(data) > cutoff_kd
        rej_kd = rej_kd + 1;
    end
    if anderson(data) > cutoff_ad
        rej_ad = rej_ad + 1;
    end
end

fprintf('alpha = %.2f, size = %d, POWER KD (nrm) = %.3f\n', signif,
    sample_size, rej_kd / runs);
fprintf('alpha = %.2f, size = %d, POWER AD (nrm) = %.3f\n', signif,
    sample_size, rej_ad / runs);
```

Now we are interested in computing power for the case when the true distribution is assumed to be Laplacian.

```
rng(shuffle);
runs = 5000;
rej_kd = 0;
rej_ad = 0;
for j = 1:runs
    data = randlap(sample_size, 1);

    if kd(data) > cutoff_kd
        rej_kd = rej_kd + 1;
    end
    if anderson(data) > cutoff_ad
        rej_ad = rej_ad + 1;
    end
end

fprintf('alpha = %.2f, size = %d, POWER KD (lap) = %.3f\n', signif,
    sample_size, rej_kd / runs);
fprintf('alpha = %.2f, size = %d, POWER AD (lap) = %.3f\n', signif,
    sample_size, rej_ad / runs);
```

Obtained results

We first list the computed cutoff values for each parameter combination. For each significance - sample size pair we have 3 degrees of freedom parameter values.

	α								
df	0.10			0.05			0.01		
	3	6	12	3	6	12	3	6	12
30	0.111	0.115	0.119	0.122	0.127	0.132	0.149	0.156	0.162
60	0.081	0.084	0.089	0.088	0.093	0.096	0.102	0.111	0.117
100	0.063	0.066	0.070	0.069	0.073	0.077	0.081	0.088	0.091

Table 0.0.1: KD-statistic cutoff values

and we have a similar table for AD-values:

	α								
df	0.10			0.05			0.01		
	3	6	12	3	6	12	3	6	12
30	0.551	0.542	0.545	0.678	0.630	0.655	1.023	0.926	0.895
60	0.537	0.519	0.543	0.646	0.619	0.644	0.914	0.849	0.899
100	0.545	0.512	0.537	0.665	0.617	0.649	0.943	0.888	0.896

Table 0.0.2: AD-statistic cutoff values

Now we verify these values by running (with a new seed) additional simulations and determine size for each test. First, for KD-test we have:

	α								
df	0.10			0.05			0.01		
	3	6	12	3	6	12	3	6	12
30	0.099	0.098	0.111	0.053	0.050	0.055	0.012	0.012	0.010
60	0.104	0.099	0.092	0.055	0.049	0.056	0.012	0.009	0.010
100	0.096	0.109	0.093	0.046	0.046	0.049	0.012	0.009	0.010

Table 0.0.3: Size verification for KD-test

and for AD-test we have:

	α								
df	0.10			0.05			0.01		
	3	6	12	3	6	12	3	6	12
30	0.102	0.093	0.106	0.048	0.050	0.052	0.011	0.010	0.012
60	0.103	0.105	0.106	0.053	0.051	0.049	0.012	0.012	0.009
100	0.100	0.098	0.087	0.048	0.044	0.050	0.011	0.009	0.011

Table 0.0.4: Size verification for AD-test

As we can see, for both tests these results are where we expect them to be, so we have some confidence in the numbers obtained. However, they could be more precise if we ran our simulation for more trials. Unfortunately, this would require much more computation time, much more than we could afford, so we went with a more limited estimate.

We now proceed to results concerning power of these tests. We first report our results for power when the alternative is assumed to be a normal distribution.

	α		
df	0.10	0.05	0.01
30	0.630	0.617	0.571
60	0.995	0.993	0.993
100	1.000	1.000	0.999

Table 0.0.5: Power for KD test(alternative is normal)

and for AD test (still when alternative is normal)

	α		
df	0.10	0.05	0.01
30	0.522	0.462	0.380
60	0.976	0.948	0.767
100	0.995	0.941	0.902

Table 0.0.6: Power for AD test(alternative is normal)

For the case when alternative is Laplace we have:

	α		
df	0.10	0.05	0.01
30	0.089	0.047	0.011
60	0.115	0.057	0.011
100	0.136	0.062	0.014

Table 0.0.7: Power for KD test(alternative is Laplace)

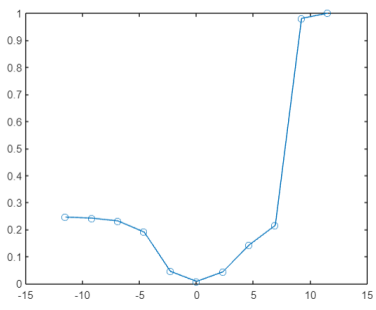
and for AD test we have:

	α		
df	0.10	0.05	0.01
30	0.111	0.066	0.021
60	0.147	0.085	0.025
100	0.174	0.107	0.027

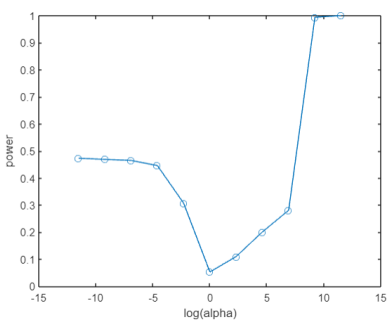
Table 0.0.8: Power for AD test(alternative is Laplace)

Bonus: Power plots for Normal-Inverse Gaussian

As it can be seen, normal and Laplace do not really have a shape parameter that we could vary to obtain different values of power for our tests. For that reason, now we would like to use Normal-inverse Gaussian distribution as our alternative and vary its alpha parameter to obtain power plots. We take μ and β to be equal to 0 and fix δ at 1. α is a positive parameter which we take to be equal to 10^i for $i \in \{-5, -4, \ldots, 4, 5\}$. We fix our sample size at 100 and significance at 0.05. We use cutoff values obtained previously. Note that x-axis is in $\log(\alpha)$



KD-test power



AD-test power

It is interesting that power seems to be distributed around $\alpha = 1$. However, we do not perfect symmetry either - while for $\alpha = 10^5$ we get power equal to one for pretty much both tests, at $\alpha = 10^{-5}$ we only get 0.3 and 0.5 for KD and AD tests respectively. We didn't vary sample size or significance, since their impact on the results is widely known and is consistent across different distributions. Thus, we were only interested in varying α to discover this kind of power plot shape.

The code to make these two plots overall resembles the code we used to compute power previously:

```
signif = 0.05;
sample_size = 100;
alphas_pows = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
alphas = 10.^alphas_pows;
kd_pows = zeros(length(alphas), 1);
ad_pows = zeros(length(alphas), 1);
s = 1;
for alpha = alphas
    rng(shuffle);
    runs = 1000;
    rej_kd = 0;
    rej_ad = 0;
    for j = 1:runs
        data = nigrnd(alpha, 0, 0, 1, sample_size, 1);

        if kd(data) > 0.093
            rej_kd = rej_kd + 1;
        end
        if anderson(data) > 0.619
            rej_ad = rej_ad + 1;
        end
    end
    kd_pows(s) = rej_kd/runs;
    ad_pows(s) = rej_ad/runs;

    s = s+1;
    fprintf('size = %d, alph = %f, POWER KD (NIG) = %.3f\n', sample_size,
       alpha, rej_kd / runs);
    fprintf('size = %d, alph = %f, POWER AD (NIG) = %.3f\n', sample_size,
       alpha, rej_ad / runs);
end

plot(log(alphas), ad_pows, o-);
xlabel('log(alpha)')
ylabel('power')
```

The only thing interesting here is the nigrnd function. Here we use inverse gaussian to compute normal inverse gaussian.

```
x = invgrnd(delta, sqrt(alpha^2 - beta^2), m, n);
y = normrnd(0, 1, m, n);
r = sqrt(x) .* y + mu * ones(m, n) + beta * x;
```

and invgrnd function is a series of daunting computations.

```
function R = invgrnd(delta, gamma, M, N)
    V = chi2rnd(1, M, N);
    x1 = delta / gamma * ones(M, N) + 1 / (2 * gamma^2) * (V + sqrt(4 *
       gamma * delta * V + V.^2));
    x2 = delta / gamma * ones(M, N) + 1 / (2 * gamma^2) * (V - sqrt(4 *
       gamma * delta * V + V.^2));
    Y = unifrnd(0, 1, M, N);
    p1 = (delta * ones(M, N)) ./ (delta * ones(M, N) + gamma * x1);
```

```
    p2 = ones(M, N) - p1;
    C = (Y < p1);
    R = C .* x1 + (ones(M, N) - C) .* x2;
end
```

We need to make explicit, however, that this code is heavily inspired by the work published by Allianz, Group Risk Controlling in 2008.

Problem 2

Student's T

Program that computes the student's T distribution with a given DOF by the inversion formula

We first filter out positive user inputs by the following simple piece of code.

```
%Checks whether integer is positive or negative
XX = input('please enter DOF : ');;

 if XX > 0
     DOF = XX

 else
    disp(' please enter a positive number and try again')
 end
```

Now we define a function to compute the Students's T distribution by the inversion formula. The formula we used was as follows: $f_X(x) = \frac{1}{2\pi}\int_{-\infty}^{-\infty} e^{-itx}\phi_X(t)\,dt$
For a given Degree of Freedom (v), we computed the characteristic function using the formula as follows: $\phi_X(t:v) = \frac{K_{\frac{v}{2}}(v^{\frac{1}{2}}|t|)^{\frac{v}{2}}(v^{\frac{1}{2}}|t|)^{\frac{1}{2}}}{\gamma(v/2)2^{\frac{v}{2}-1}}$
As you can see in the function below, we can input the Degree of Freedom(v), the x values range & number of values. The output is a set of x values and their respective y values. When we plot these, we get a student's T distribution.

```
%calculate student T by characteristic function inversion formula
function [X ,f] = my_student_t(v, min_x, max_x, n)

f = zeros(n, 1);
X = linspace(min_x, max_x, n);
for i = 1:n
    x = X(i);
    %calculating characteristic function for Student T: Given DOF(v)
    % from Time Series book pg 736
    phi = @(t) (besselk(v/2, (sqrt(v) .* abs(t))) .* ((sqrt(v) .* abs(t))
        .^(v/2)))/(2.^(v/2 - 1) .* gamma(v/2));
    %defining function to intergrate for inversion theorem
    pdf_t = @(t, x) (1/(2 .* pi)) .* exp((-1i) .* t .* x ) .* phi(t);
    %integrating t from -inf to inf
    f(i) = integral(@(t) pdf_t(t, x), -inf,inf);
end
```

We then call this function and give it some input values. The in-built *tpdf* function is used to calculate the exact density. The Probability Densities are plotted on the same graph to compare them.

```
[X, y1_cal] = my_student_t(3, -3, 3, 500); %calling self defined function
   to calculate students T from -3,3, 500 values with DOF 3
[X2, y2_cal] = my_student_t(30, -6, 6, 500); %calling self defined function
    to calculate students T from -6,6, 500 values with DOF 30
```

```
x = [-3:.1:3];
x2 = [-6:.1:6];
y1 = tpdf(x,3); %using inbuilt function
y2 = tpdf(x2, 30);

figure;
plot(X,y1_cal,'Color','black','LineStyle','-')
hold on
plot(x,y1,'Color','red','LineStyle','--','LineWidth', 2.0)
plot(x2,y2,'Color','blue','LineStyle','-')
plot(X2,y2_cal,'Color','green','LineStyle','--')
xlabel('Observation')
ylabel('Probability Density')
legend({'calculated: DOF = 3','simulated: DOF = 3', 'calculated: DOF = 30',
    'simulated: DOF = 30' })
hold off
```

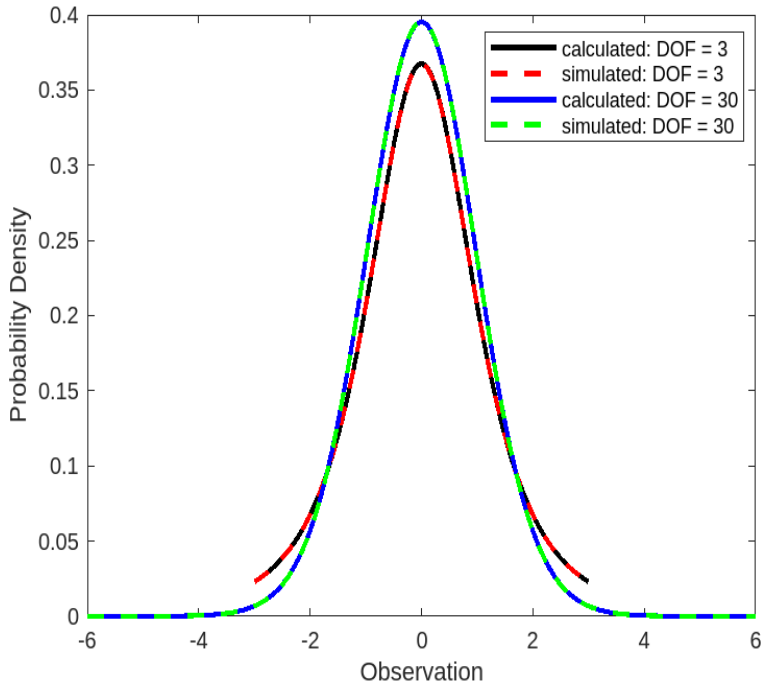


Figure 0.0.1: **Graph of exact density vs calculated by inversion formula**: DOF: 3 & DOF: 6.

As you can see in graph 0.0.1, I have plotted the distributions for two different Degrees of Freedom on the same graph for us to:

- Observe the difference in shape (distribution)
- Observe the difference made by the range of x values(min_x, max_x)

Problem 4

For clarity, we define functions for part A and C. These functions are called and plotted towards the end.

Part A

We use the integral convolution formula to compute the joint density of two independent random variables. Generally, we use this when the MGF of the resulting function is not recognizable. The formula

is as follows: $f_S(s) = \int_{-\infty}^{-\infty} f_1(x) f_2(s-x)\, dx$

```
% summing two independent student T
%using the convolution formula
function [S ,f] = my_student_t_sum(v1, v2, min_x, max_x, n)

f = zeros(n, 1);
% s = x + y
S = linspace(min_x, max_x, n);
for i = 1:n
    s = S(i);
    %define function for students T
    y = @(x, v) tpdf(x,v);
    %according to convolution formula: PDF(x) * PDF(y)
    combined_pdf = @(x) y(x ,v1) .* y(s - x, v2);
    %integrating to get y value for the combined pdf
    f(i) = quadgk(combined_pdf, -inf, inf);
end
```

Part B

Two student T distributions are simulated using the *trnd* function with two different degrees of freedom. The *ksdenstiy* function is used to compute a probability density estimate for two a list of the summed values of the simulation. For a relatively smooth graph we set the number of simulations as 10000. Then the values from the ksdensity function are plotted to obtain a graph. Below is the code for this part:

```
x = [trnd(3, 10000, 1) + trnd(5, 10000, 1)]; % summing two independent
    student T by simulation (Part B)
[f,xi] = ksdensity(x);
```

Part C

This uses the previously defined inversion formula. We calculate the characteristic function of the resulting distribution (ϕ_{X+Y}) by multiplying ϕ_X and ϕ_Y. This is because the resulting MGF can be expressed at the product of two MGFs in the case of independent random variables. The characteristic function is nothing but $\phi_X(t) = M_X(it)$.

The code for this part is as follows:

```
function [X ,f] = my_student_t_summed_inversion(v1, v2, min_x, max_x, n)
%calcutating student T of two different DOF
f = zeros(n, 1);
X = linspace(min_x, max_x, n);
for i = 1:n
    x = X(i);
    %defining phi function
    phi = @(t,v) (besselk(v/2, (sqrt(v) .* abs(t))) .* ((sqrt(v) .* abs(t))
        .^(v/2)))/(2.^(v/2 - 1) .* gamma(v/2));
    %multiply in case of independent variables
    phi_combined = @(t) phi(t, v1) .* phi(t,v2);
    pdf_t = @(t, x) (1/(2 .* pi)) .* exp((-1i) .* t .* x ) .* phi_combined(
        t);
    f(i) = integral(@(t) pdf_t(t, x), -inf,inf);
end
```

Plotting all functions

Here we call all the functions defined in Parts A, B and C with the same parameters. We can observe the results in figure.

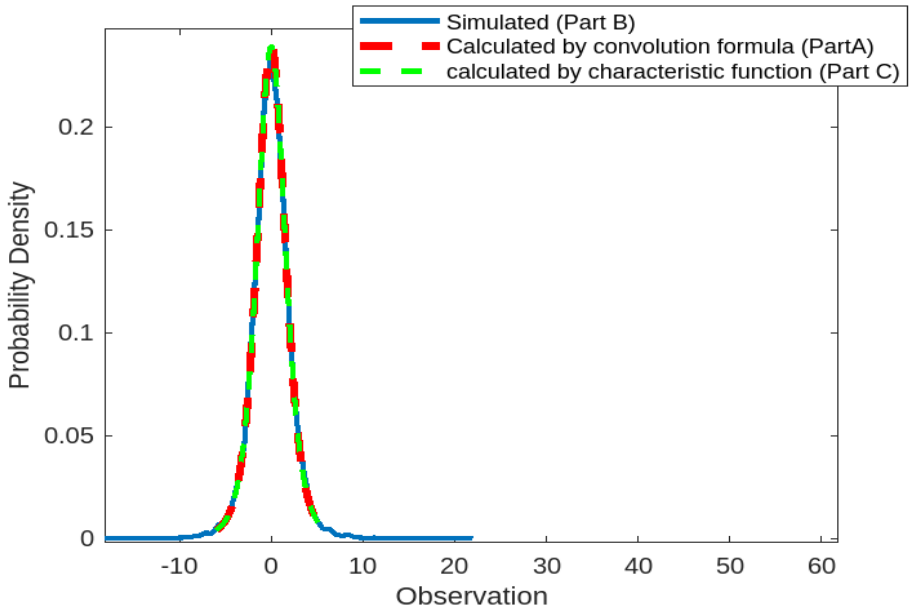


Figure 0.0.2: **Part A vs B vs C** : DOF: 3 & DOF: 5.

Below is the code to call these functions and plot the

```
x = [trnd(3, 10000, 1) + trnd(5, 10000, 1)]; % summing two independent
    student T by simulation (Part B)
[f,xi] = ksdensity(x);
[X_sum, y_cal] = my_student_t_sum(3, 5, -6, 6, 1000); %using convolution
    formula (part A)
[X, y_new] = my_student_t_summed_inversion(3, 5, -6, 6, 1000); %using
    characteristic function inversion formula, by multiplying the
    characteristic


figure
plot(xi,f,'LineWidth', 2.0)
```

```
hold on
plot(X_sum,y_cal,'Color','red','LineStyle','--','LineWidth',  3.0)
plot(X,y_new,'Color','green','LineStyle','--','LineWidth', 2.0)
xlabel('Observation')
ylabel('Probability Density')
legend({'Simulated (Part B)','Calculated by convolution formula (PartA)', '
    calculated by characteristic function (Part C)'})
hold off
```

Problem 5

We computed the Expected Shortfall by replicating the calculation detailed on page 446 of the book for the location-scale Student's t distribution. Where ϕ and Φ respectively denote p.d.f. and c.d.f. of the location-scale Student's t distribution with v degree of freedom, μ location, and σ scale. Assume v>1, and let $K = v^{-1/2}/B(v/2, 1/2)$ and $u = 1 + \frac{1}{v} * (\frac{r-\mu}{\sigma})^2$

$$\begin{aligned}
E[R|R<c] &= \frac{1}{\Phi(c)} \int_{-\infty}^{c} r\phi(r)\, dr \\
&= \frac{K}{\sigma\Phi(c)} \int_{-\infty}^{c} r(1+\frac{1}{v}(\frac{r-\mu}{\sigma})^2)^{-\frac{v+1}{2}}\, dr \\
&= \frac{K\sigma v}{2\Phi(c)} \int_{-\infty}^{1+\frac{1}{v}(\frac{c-\mu}{\sigma})^2} U^{\frac{-(v+1)}{2}}\, dU + \frac{K\sigma v}{2\Phi(c)} \int_{-\infty}^{c} \frac{2\mu}{v\sigma^2}(1+\frac{1}{v}(\frac{-\mu}{\sigma})^2)^{\frac{-(v+1)}{2}}\, dr \\
&= \mu + \frac{K\sigma}{\Phi(c)} \frac{v}{1-v}(1+\frac{1}{v}(\frac{c-\mu}{\sigma})^2)^{1-(\frac{v+1}{2})} \\
&= \mu + \frac{\sigma^2}{\Phi(c)} \frac{v}{1-v}\phi(c)(1+\frac{1}{v}(\frac{c-\mu}{\sigma})^2) \\
&= \mu - \frac{\phi(c)}{\Phi(c)}(\frac{v\sigma^2+(c-\mu)^2}{v-1})
\end{aligned} \tag{0.0.1}$$

The above formula converges to the same result as in the book if $\mu = 0$ and $\sigma = 1$.

Problem 6

We subsequently programmed a function to compute Expected Shortfall, where the input parameters are location-scale Student's t distribution parameters, and the ESlevel, represents the desired tail probability.

```
function ES=TrueES(df, mu, sigma, ESlevel)
    % Set default values
    if nargin<2
        mu=0;
    end
    if nargin<3
        sigma=1.0;
    end
    if nargin<4
        ESlevel=0.05;
    end
    c=tinv(ESlevel,df);

    ES=mu-pdf(tLocationScale,c,mu,sigma,df)*(df*sigma^2+(c-mu)^2)/cdf(
       tLocationScale,c,mu,sigma,df)/(df-1);
end
```

Problem 7

In this section, Expected Shortfall was computed through three distinct approaches: the first method involved the utilization of a theoretical formula, while the second and third methods employed parametric and non-parametric bootstrapping, respectively. Following these calculations, 90 percent confidence intervals were determined for both bootstrapping techniques. To obtain the maximum likelihood estimator for the location-scale Student's t distribution, we employed the built-in MLE function provided by Matlab. The accompanying code for this section is provided below.

```
function [true_ES, CI_p, CI_np]=part7(df, n, B)
    %Set default values
    if nargin<2
        n=500;
    end
    if nargin<3
        B=500;
    end
    options=statset(Display,off);

    %parta
    data=trnd(df,n, 1);
    phat=mle(data,'Distribution','tLocationScale','Start',[mean(data),std(
       data),df],'Options',options);
    true_ES=TrueES(phat(3),phat(1),phat(2), 0.05);

    %partb
    bootstrap_ES_p=zeros(B,1);
    for i =1:B
        bootstrap_data=trnd(df,n, 1);
        phat=mle(bootstrap_data,'Distribution','tLocationScale','Start',[
           mean(bootstrap_data),std(bootstrap_data),df],'Options',options);
        bootstrap_ES_p(i)=TrueES(phat(3),phat(1),phat(2),0.05);
    end
    CI_p=prctile(bootstrap_ES_p,[5,95]);

    %partc
    bootstrap_ES_np=zeros(B,1);
    for i=1:B
        bootstrap_data_np=datasample(data,size(data,1));
        var=quantile(bootstrap_data_np,0.05);
        bootstrap_ES_np(i)=mean(bootstrap_data_np(bootstrap_data_np<=var));
    end
    CI_np=prctile(bootstrap_ES_np,[5,95]);

    %Making a Box-Plot
    boxplot([bootstrap_ES_p,bootstrap_ES_np],'Labels',{'Parametric','Non-
       Parametric'});
    hold on;
    yline(true_ES,'Color','g');
    legend('True Value');

end
```

Below, the box plots are given for df=5 and B=500, with sample sizes n=250 and n=500.

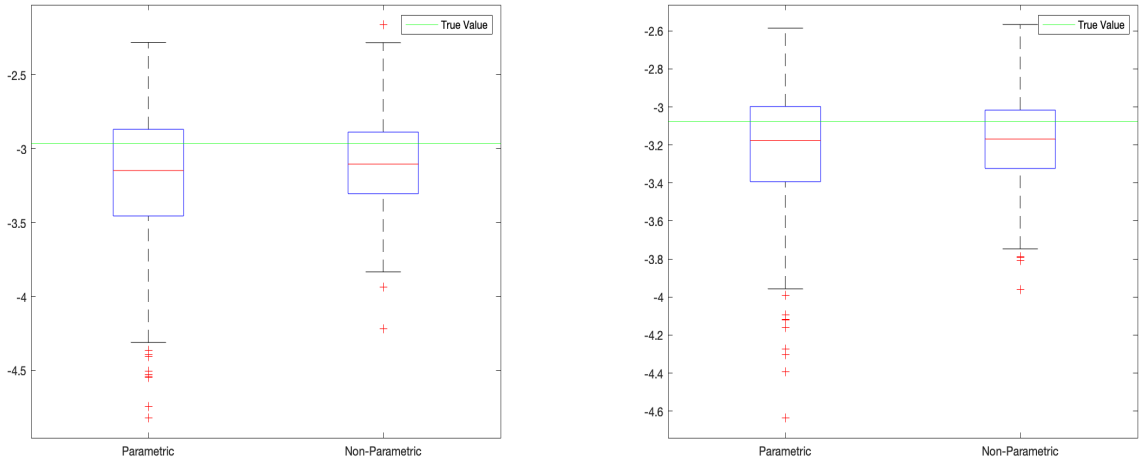


Figure 0.0.3: **Box plot of parametric and non-parametric ES with true value of ES**: n=250 & n=500

Problem 8

In this section, we determined the convergence probabilities by running N simulations and counting the occurrences where the True ES value fell within the 90 percent confidence interval. The convergence probability was computed by dividing this count by N. Additionally, we calculated the lengths of the 90 percent confidence intervals for both parametric and non-parametric methods. The code is given below.

```
function [p_p,p_np,l_p,l_np]=part8(N,df,n,B)
CI_p_l=zeros(N,1);
CI_np_l=zeros(N,1);
s_p=0;
s_np=0;
for i=1:N
    [true_ES, CI_p, CI_np]=part7(df, n, B);
    CI_p_l=max(CI_p)-min(CI_p);
    CI_np_l=max(CI_np)-min(CI_np);
    if min(CI_p)<=true_ES && true_ES <max(CI_p)
        s_p=s_p+1;
    end
    if min(CI_np)<=true_ES && true_ES<=max(CI_np)
       s_np=s_np+1;
    end
end
p_p=s_p/N;
p_np=s_np/N;
l_p=mean(CI_p_l);
l_np=mean(CI_np_l);
end
```

Simulations were conducted for N values of 300, 500, 700, 1000, and 1500. The larger simulations were infeasible due to the time constraints of the project. The results of the coverage probability are given in the table below.

As observed in the graph, the coverage probabilities approach 90 percent as the number of simulations increases. Also, the probability of convergence is consistently higher for the parametric bootstrap method compared to the non-parametric bootstrap method.

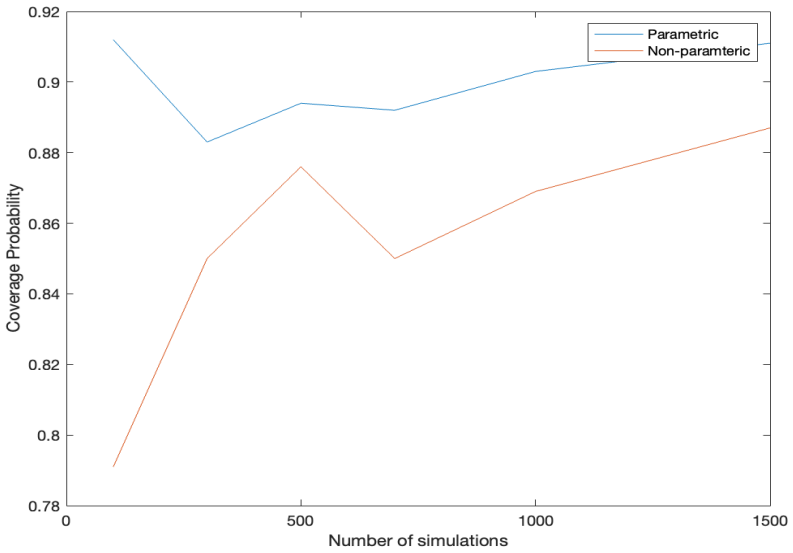


Figure 0.0.4: Coverage probability

The average length of a confidence interval was computed by subtracting the low endpoint from the high endpoint. In the last simulation ($N = 1500$), the average length of the confidence interval for the parametric bootstrap method was 1.30, while for the non-parametric bootstrap method, it was 1.42. As expected, the parametric bootstrap method has a shorter average length for the confidence interval.