University of Zurich[UZH]

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

Assignment 2, due November 22nd, 2022

Prof. Dr. Marc Paolella

Ilaria Avallone, 18-435-800

Lorena Tassone, 18-700-237

Naina Srivastava, 21-738-117

Question 1

The goal of this exercise is to compute the length of a 90% bootstrap confidence interval (CI), based on B bootstrap replications, and check whether the interval contains the true Expected Shortfall (ES). In this first exercise, the true data distribution and the one assumed for the parametric bootstrap are both the same, namely the Student t.

In a first step, we calculate the true ES based on the three true and beforehand fixed parameters (namely location=0, scale=1 and degrees of freedom=4). Then, in a FOR loop we simulate "rep" repetitions of an IID T-length sequence of the Student t distribution to obtain the data generating process (DGP) of an IID location-scale Student t. For each of those "rep" repetitions, as said before, we calculate a bootstrap 90% confidence interval, based on B bootstrap replications. The bootstrap is performed for both the parametric and the non-parametric bootstrap.

In the parametric case, we need to first compute the Maximum Likelihood Estimation (MLE) of the three parameters loc, scale and df, corresponding to an IID Student t, based on the simulated data set. The simulated ES is then computed using a t distributed sample set based on those estimated parameter values, which is adapted by scale and location. We compute the α-quantile (α=0.1) to get the Value-at-Risk (VaR) and from that we get the simulated ES by taking the mean of all values lower-equal the VaR. In this case of the parametric bootstrap, to get the sample set, we use an assumed distribution. In this case the true t distribution is assumed.
The MLE parameter estimation is computed in two ways: once using the codes provided in the book "Fundamental Statistical Inference" by Marc S. Paolella, and once using MATLAB's built-in *mle* function. Like this results and its accuracy can be compared. Regarding the book method, we take advantage of the *tlikmax0* and *tloglik* functions (Program Listing 4.5 in the book "Fundamental Statistical Inference") to maximize the log-likelihood of the IID Student's t model using the true data distribution (*data*) and the true parameters *initvec*=[df loc scale].
For both methods we compute the simulated ES based on the bootstrap values, the length of the CI and we check whether the interval contains the beforehand computed true ES.

For the non-parametric bootstrap we just sample "rep" times, with replacement, from the true data set which was previously simulated. In this case we take advantage of the Program Listing 2.9 in the book "Fundamental Statistical Inference". In this case of the non-parametric bootstrap, there is no need to assume a distribution. As done before, we then compute the simulated ES, the length of the CI and check whether the interval contains the true ES.

Eventually, to get comparable performance estimates, for each method we compute the mean CI length and the coverage ratio of the true ES.

The code of the procedure described above is presented in Listing 1, where we loop over different sample sizes.

Listing 1: Computing the length of a bootstrap 90% confidence interval based on B bootstrap replications, and checking whether the interval contains the true ES - Student t distribution

```
% Parameters
df = 4; loc = 0; scale = 1;
alpha = 0.1;
c01 = tinv(alpha , df);        % left tail quantile c
rep = 1000;                    % #Repititions
T_samples = [250 500 2000];    % Sample size
B = 1000;                      % #Bootstrap samples
rand('twister',6);             % set seed value to replicate results
initvec = [df loc scale];      % for parametric MLE

% Initializing vectors
ESvec_nonparam = zeros(B, 1);
ESvec_param_book = zeros(B, 1);
ESvec_param_matlab = zeros(B, 1);
ci_length_nonparam = zeros(rep, length(T_samples));
ci_length_param_book = zeros(rep, length(T_samples));
ci_length_param_matlab = zeros(rep, length(T_samples));
ci_trueES_nonparam = zeros(rep, length(T_samples));
ci_trueES_param_book = zeros(rep, length(T_samples));
ci_trueES_param_matlab = zeros(rep, length(T_samples));

% True ES for student t
truec = loc+scale*c01;
ES_01_analytic = -tpdf(c01,df)/tcdf(c01,df)*(df+c01^2)/(df-1);
trueES = loc+scale*ES_01_analytic;

% Simulating "rep" repetitions of an IID T-length sequence of Student t
% For each rep compute bootstrap 90% CI based on B bootstrap
```

```
replications
for t = 1:length(T_samples)
    for r = 1:rep
        % Generating data points from student t distribution
        data=loc+scale*trnd(df,T_samples(t),1);


        %% PARAMETRIC BOOTSTRAP %%%
        % 1. Using book MLE code
        mle_param_book = tlikmax0(data, initvec);

        % Computing simulated ES using estimated MLE parameters
        for b=1:B
            param_bs_sample_book = mle_param_book(2)+mle_param_book(3)*trnd(mle_param_book(1),T_samples(t),1);
            VaR_param_book = quantile(param_bs_sample_book, alpha);
            temp_book = param_bs_sample_book(param_bs_sample_book<=VaR_param_book);
            ESvec_param_book(b) = mean(temp_book);
        end

        % Computing length of CI
        ci_param_book = quantile(ESvec_param_book,[alpha/2 1-alpha/2]);
        low_param_book = ci_param_book(1); high_param_book = ci_param_book(2);
        ci_length_param_book(r,t) = high_param_book-low_param_book;

        % Checking wether the CI contains the true ES
        ci_trueES_param_book(r,t) = (trueES>low_param_book)&(trueES<high_param_book);

        % 2. Using MATLAB built-in MLE function
        % output: [loc,scale,nu]
        mle_param_matlab = mle(data, 'Distribution', 'tLocationScale');

        % Computing simulated ES using estimated MLE parameters
        for b=1:B
            param_bs_sample_matlab = mle_param_matlab(1)+mle_param_matlab(2)*trnd(mle_param_matlab(3),T_samples(t),1);
            VaR_param_matlab = quantile(param_bs_sample_matlab, alpha);
            temp = param_bs_sample_matlab(param_bs_sample_matlab<=VaR_param_matlab);
            ESvec_param_matlab(b) = mean(temp);
        end

```

```
        % Computing length of CI
        ci_param_matlab = quantile(ESvec_param_matlab, [alpha/2 1-alpha/2]);
        low_param_matlab = ci_param_matlab(1); high_param_matlab = ci_param_matlab(2);
        ci_length_param_matlab(r,t) = high_param_matlab-low_param_matlab;

        % Checking wether the CI contains the true ES
        ci_trueES_param_matlab(r,t) = (trueES>low_param_matlab)&(trueES<high_param_matlab);


        %% NON-PARAMETRIC BOOTSTRAP %%%
        % book code
        % computing simulated ES
        for b=1:B
            ind = unidrnd(T_samples(t),[T_samples(t),1]);
            nonparam_bs_sample=data(ind); % Program Listing 2.9
            VaR_nonpara = quantile(nonparam_bs_sample, alpha);
            temp = nonparam_bs_sample(nonparam_bs_sample<=VaR_nonpara);
            ESvec_nonparam(b) = mean(temp);
        end

        % Computing length of CI
        ci_nonparam = quantile(ESvec_nonparam, [alpha/2 1-alpha/2]);
        low_nonparam = ci_nonparam(1); high_nonparam = ci_nonparam(2);
        ci_length_nonparam(r,t) = high_nonparam-low_nonparam;

        % Checking wether the CI contains the true ES
        ci_trueES_nonparam(r,t) = (trueES>low_nonparam)&(trueES<high_nonparam);
    end
end
```

Listing 2: Function *tlikmax0* that computes MLE

```
function MLE = tlikmax0(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 % An ad hoc way of prevent ing negative values
if c<0.01, c=rand; end % which works , but i s NOT recommended !
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
```

Table 1: Performance results

Bootstrap:	Parametric (book)			Parametric (MATLAB)			Non-Parametric		
Sample size:	250	500	2000	250	500	2000	250	500	2000
Mean CI length	1.005	0.705	0.355	1.004	0.704	0.355	0.932	0.680	0.349
Coverage ratio	0.956	0.965	0.974	0.954	0.962	0.976	0.847	0.879	0.896

The results are shown in Table 1 and they do not reflect our expectations. Since the true data and the one assumed in the parametric bootstrap are the same (both in the MATLAB and in the book option), we expected the parametric bootstrap to work better; however it seems that with an increase in T the non-parametric bootstrap works better. We notice that as the sample size increases, the coverage ratio increases for both the parametric and non-parametric, however for parametric the coverage ratio exceeds 0.90 which is not what the case should be. Since we took a confidence interval of 0.90 to begin with, we expected a coverage ratio of around 0.90. Furthermore, We also see that the mean CI length decreases with an increase in T. This is due to the fact that a higher sample size simulates a more exact distribution.

Question 2

After having done the above computations for the data generating process (DGP) of an IID location-scale Student t, and correctly assuming this distribution for the parametric bootstrap, we now use the non-central Student t as the true data. Hence, our true data is non-central Student t distributed, whereas we continue to assume a "regular" location-scale Student t for the bootstrap.

The non-central t is asymmetric, but our parametric bootstrap uses the regular symmetric Student t as the parametric model. Hence, we expect to see deterioration in the performance of the parametric model, as the asymmetry amount (noncentrality parameter ν) increases. The non-parametric bootstrap on the other hand should do okay, since no assumption regarding the true distribution is made there.

In a first step, we simulate the non-central Student t (NCT) distribution using the function *stdnctpdfln_j* from the book "Fundamental Statistical Inference" (Program Listing 9.2) by Marc S. Paolella. The function computes a direct density approximation (d.d.a.) to the NCT(ν,γ), using the log density. To compare the distribution to the MATLAB built-in function *nctpdf*, we take the exponential of the d.d.a. NCT computed in Listing 3 and plot it in Figure 1 and 2 respectively for df=3 and df=6. We can see that decreasing the noncentrality parameter skews the distribution to the left. Using df=3 however, skews the distribution stronger, which is why we expect results for df=6 to be better.

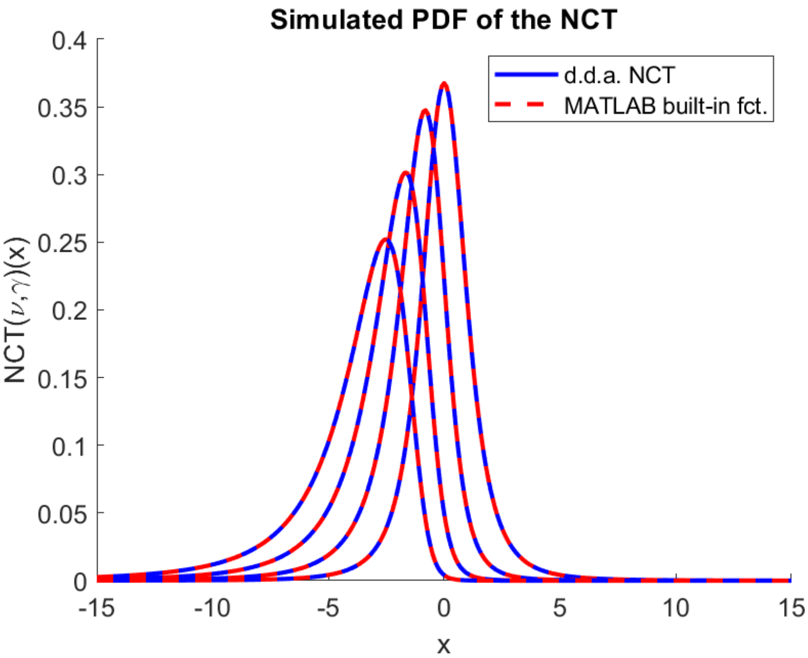

Figure 1: PDF of the non-central Student t (df=3) (from left to right line: mu = -3,-2,-1,0)

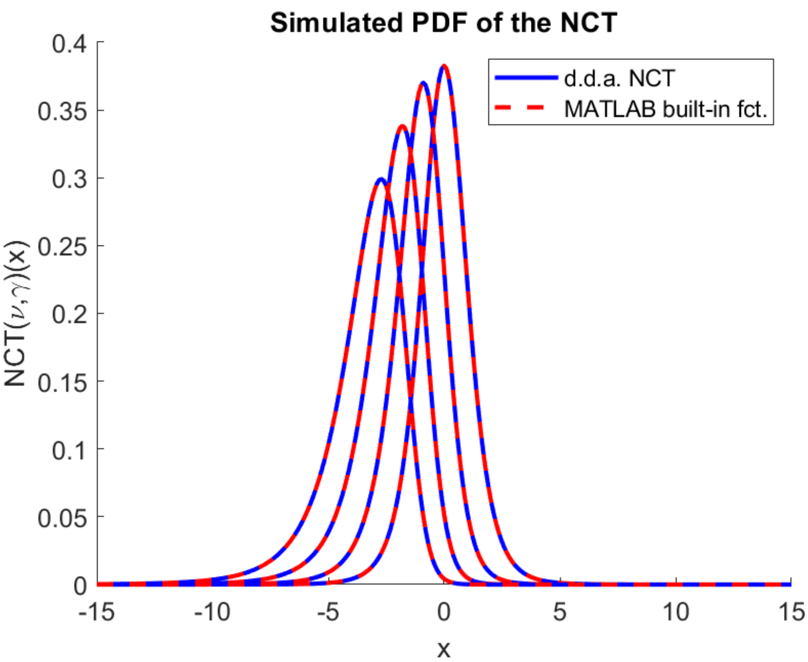

Figure 2: PDF of the non-central Student t (df=6) (from left to right line: mu = -3,-2,-1,0)

Listing 3: Simulation of a non-central t distribution

```
% Parameters
mu = [0 -1 -2 -3];
df = 3; % df = 6
x = (-15:0.1:15)';

for j=1:length(mu)
    % book code
    NCT_sim = exp(stdnctpdfln_j(x, df, mu(j)));

    % MATLAB built in function
    nct = nctpdf(x, df, mu(j));

    hold on,
    plot(x, NCT_sim, 'b','LineWidth', 2)
    plot(x, nct, 'r--','LineWidth', 2)
    xlim([-15 15])
    legend('d.d.a. NCT','MATLAB built-in fct.')
    title('Simulated PDF of the NCT')
    xlabel("x"); ylabel("NCT(\nu,\gamma)(x)")
    set(gca, 'fontsize', 12)

    name = ['Assignment2_ex2_df',int2str(df),'.png'];

    saveas(gcf,name)
    hold off
end
```

Listing 4: Function *stdnctpdfln_j* that computes the log density and returns the direct approximation to the NCT

```
function pdfln = stdnctpdfln_j(x, df, mu) % Program Listing 9.2
vn2 = (df+1)/2; rho=x.^2;
pdfln = gammaln(vn2) - 1/2*log(pi*df) - gammaln(df/2) - vn2*log1p(rho/df);
if (all(mu == 0)), return, end
idx = (pdfln >= -37);
gcg = mu.^2; pdfln = pdfln - 0.5*gcg; xcg = x .* mu;
term = 0.5*log(2) + log(xcg) - 0.5*log(max(realmin,df+rho));
term(term == -inf) = log(realmin); term(term == +inf) = log(realmax); maxiter = 1e4; k = 0;
logterms = gammaln((df+1+k)/2) - gammaln(k+1) - gammaln(vn2) + k*term; fractions = real (exp( logterms ) ) ; logsumk = log ( fractions ) ;
while (k < maxiter)
k = k + 1;
logterms = gammaln((df+1+k)/2) - gammaln(k+1) - gammaln(vn2) + k*term(
```

```
    idx); fractions = real(exp(logterms-logsumk(idx)));
logsumk(idx) = logsumk(idx) + log1p(fractions);
idx(idx) = (abs(fractions) > 1e-4); if (all(idx == false)), break, end
end
pdfln = real (pdfln+logsumk) ;
end
```

In a next step, we compute the true Expected Shortfall of the NCT distribution in two ways in Listing 4: via simulation and via integral definition.

First, since the non-central t may be expressed in terms of a Normal (z) and a Chi-square (χ^2) distribution, we take a random sample of both distribution and apply the following calculation to get a random sample of the non-central t:

$$NCT = \frac{z}{\sqrt{\chi^2/\nu}} \tag{1}$$

Next, we multiply the NCT random sample by a scale factor and add a location parameter to make it more general. We then compute the α-quantile (α=0.1) to get the Value-at-Risk (VaR) and from that we get the simulated ES by taking the mean of all values lower-equal the VaR.
To compute the ES using the integral definition, we first solve for the α-quantile and then do numeric integration of the NCT density multiplied by x.

Listing 5: Expected Shortfall of the NCT: via simulation and via integral definition

```
% Parameters
alpha = 0.1; loc=0; scale=1; n = 10^6;
mu = 0;            % [0 -1 -2 -3]
df = 3;            % [3 6]

%%% ES via simulation
% Compute Normal and Chi-squared random sample
norm = normrnd(mu,1,n,1); chi2 = chi2rnd(df,n,1);

% Compute NCT random sample
nct_rnd_sample = loc+scale*(norm./sqrt(chi2/df));

% Compute ES
VaR = quantile(nct_rnd_sample, alpha);
temp_2 = nct_rnd_sample(nct_rnd_sample <= VaR);
ES_simulated = mean(temp_2);

%%% ES via integral definition
c01_nct = nctinv(alpha , df, mu);
ES_01_nct = @(x) x.*nctpdf(x, df, mu);
```

```
ES_nct_int= integral(ES_01_nct, -Inf, c01_nct)/alpha;
```

The results for different degrees of freedom (df) and noncentrality parameters (mu) are presented in Table 2 and 3 below. As expected, the results are quite similar. For further calculations, we decide to set the ES via integral definition as the true ES of the NCT distribution, since it is not based on simulation and hence, is more accurate.

Table 2: Expected Shortfall using different methods and parameters (df=3)

	mu=0	mu=-1	mu=-2	mu=-3
ES via simulation	-2.9083	-5.3684	-8.2611	-11.4369
ES via integral definition	-2.9108	-5.3622	-8.2637	-11.4254

Table 3: Expected Shortfall using different methods and parameters (df=6)

	mu=0	mu=-1	mu=-2	mu=-3
ES via simulation	-2.1891	-3.6994	-5.3853	-7.1868
ES via integral definition	-2.1872	-3.7007	-5.3840	-7.1893

Finally, as done in Question 1, for each repetition, we calculate a parametric and non-parametric bootstrap 90% confidence interval, based on B bootstrap replications. Again, we compute for each "rep" the length of the CI and check whether the true ES is in it or not. The computations shown in Listing 5 are run multiple times for different values of degrees of freedom (df), noncentrality parameters (mu) and looped over different sample sizes. The mean CI length and mean coverage ratio of these computations are summarized in Table 4 for df = 3 and Table 5 for df = 6.

Listing 6: Computing the length of a bootstrap 90% confidence interval based on B bootstrap replications, and checking whether the interval contains the true ES

```
% Parameters
alpha = 0.1;
loc = 0; scale = 1;
rep = 1000;                    % #Repititions
T_samples = [100 500 2000];    % Sample size
B = 1000;                      % #Bootstrap samples
df = 3                         % [3 6]
mu = 0;                        % [0 -1 -2 -3]
rand('twister',6)              % set seed value to replicate results
initvec = [df loc scale];      % for parametric MLE

% Initializing vectors
```

```
ESvec_nonparam = zeros(B, 1);
ESvec_param_book = zeros(B, 1);
ESvec_param_matlab = zeros(B, 1);
ci_length_nonparam = zeros(rep, length(T_samples));
ci_length_param_book = zeros(rep, length(T_samples));
ci_length_param_matlab = zeros(rep, length(T_samples));
ci_trueES_nonparam = zeros(rep, length(T_samples));
ci_trueES_param_book = zeros(rep, length(T_samples));
ci_trueES_param_matlab = zeros(rep, length(T_samples));

% True ES for NCT (calculated in listing above)
trueES = ES_nct_int;

% Simulating "rep" repetitions of an IID T-length sequence of NCT
% For each rep claculate bootstrap 90% CI based on B bootstrap
    replications
for t = 1:length(T_samples)
    for r = 1:rep
        % Generating data points from NCT distribution
        norm=normrnd(mu,1,T_samples(t),1);
        chi2=chi2rnd(df,T_samples(t),1);
        data = loc+scale*(norm./sqrt(chi2/df));


        %%% PARAMETRIC BOOTSTRAP %%%
        % 1. Using Book MLE code
        mle_param_book = tlikmax0(data, initvec);

        % Computing simulated ES using estimated MLE parameters
        for b=1:B
            param_bs_sample_book = mle_param_book(2)+mle_param_book(3)*
                trnd(mle_param_book(1),T_samples(t),1);
            VaR_param_book = quantile(param_bs_sample_book, alpha);
            temp_book = param_bs_sample_book(param_bs_sample_book<=
                VaR_param_book);
            ESvec_param_book(b) = mean(temp_book);
        end

        % Computing length of CI
        ci_param_book = quantile(ESvec_param_book,[alpha/2 1-alpha/2]);
        low_param_book = ci_param_book(1); high_param_book =
            ci_param_book(2);
        ci_length_param_book(r,t) = high_param_book-low_param_book;

        % Checking whether the CI contains the true ES
        ci_trueES_param_book(r,t) = (trueES>low_param_book)&(trueES<
```

```
            high_param_book);

    % 2. Using MATLAB built-in MLE function:
    % output: [loc,scale,nu]
    mle_param_matlab = mle(data, 'Distribution', 'tLocationScale');

    % Computing simulated ES using estimated MLE parameters
    for b=1:B
        param_bs_sample_matlab = mle_param_matlab(1)+mle_param_matlab(2)*trnd(mle_param_matlab(3),T_samples(t),1);
        VaR_param_matlab = quantile(param_bs_sample_matlab, alpha);
        temp = param_bs_sample_matlab(param_bs_sample_matlab<=VaR_param_matlab);
        ESvec_param_matlab(b) = mean(temp);
    end

    % Computing length of CI
    ci_param_matlab = quantile(ESvec_param_matlab, [alpha/2 1-alpha/2]);
    low_param_matlab = ci_param_matlab(1); high_param_matlab = ci_param_matlab(2);
    ci_length_param_matlab(r,t) = high_param_matlab-low_param_matlab;

    % Checking whether the CI contains the true ES
    ci_trueES_param_matlab(r,t) = (trueES>low_param_matlab)&(trueES<high_param_matlab);


    %%% NON-PARAMETRIC BOOTSTRAP %%%
    % Book code
    % computing simulated ES
    for b=1:B
        ind = unidrnd(T_samples(t),[T_samples(t),1]);
        nonparam_bs_sample=data(ind);
        VaR_nonpara = quantile(nonparam_bs_sample, alpha);
        temp_nonparam = nonparam_bs_sample(nonparam_bs_sample<=VaR_nonpara);
        ESvec_nonparam(b) = mean(temp_nonparam);
    end

    % Computing length of CI
    ci_nonparam = quantile(ESvec_nonparam, [alpha/2 1-alpha/2]);
    low_nonparam = ci_nonparam(1); high_nonparam = ci_nonparam(2);
    ci_length_nonparam(r,t) = high_nonparam-low_nonparam;
```

```
        % Checking whether the CI contains the true ES
        ci_trueES_nonparam(r,t) = (trueES>low_nonparam)&(trueES<
            high_nonparam);
    end
```

In the case of the noncentrality parameter, mu, being 0, the NCT corresponds to the "regular" Student t distribution, resulting in the same setting as in Question 1. Hence, we expect the parametric bootstrap to perform better than the non-parametric.The performance should increase with the sample size, since we get a better estimation of the true distribution like this. As we can see in Table 4 and 5, the results for df=3 and df=6 achieve the most accurate coverage ratio for a sample size of 100. Increasing it actually makes the coverage ratio diverge more from 0.9. This is not what we expected, as we discussed already in Question 1.

Since our true data has a non-central t distribution, but we wrongly assume a "regular" t distribution for our parametric bootstrap, we expect the parametric bootstrap to perform worse than the non-parametric, in the case where mu≠0. This is due to the fact that when computing the non-parametric bootstrap, no wrongly assumed distribution is being used in the computation. Additionally, we expect a degradation in performance as the asymmetry amount (absolute mu) increases. Further, we anticipate the parametric coverage ratio to decrease when increasing the sample size. Increasing the sample size generates a more accurate simulation of the true data. Hence, since we wrongly assume a "regular" t distribution the difference between the true and assumed distributions increases with the sample size. The above described expectation are proven to be correct as shown in Table 4 and 5.
Also, as described before, we expect the results for df=6 to be better, since, as shown in plot 1 and 2, they are skewed less. Also this can be shown in the following result tables.
Lastly, we can see that the parametric method from the book performs slightly better in both df cases.

Table 4: Performance results (df=3)

			df = 3						
Bootstrap:	Parametric (book)			Parametric (MATLAB)			Non-Parametric		
Sample size:	100	500	2000	100	500	2000	100	500	2000
				mu = 0					
Mean CI length	2.302	1.040	0.532	2.306	1.038	0.531	1.901	1.004	0.509
Coverage ratio	0.900	0.959	0.979	0.898	0.961	0.977	0.747	0.836	0.889
				mu = -1					
Mean CI length	2.785	1.313	0.668	2.790	1.309	0.668	2.997	1.604	0.843
Coverage ratio	0.611	0.268	0.003	0.604	0.264	0.003	0.739	0.848	0.875
				mu = -2					
Mean CI length	4.289	2.037	1.064	4.296	2.037	1.064	4.405	2.392	1.285
Coverage ratio	0.558	0.172	0.001	0.549	0.176	0.001	0.752	0.838	0.878
				mu = -3					
Mean CI length	6.194	2.989	1.588	6.193	2.989	1.586	6.022	3.210	1.722
Coverage ratio	0.552	0.217	0	0.549	0.209	0.001	0.745	0.839	0.87

Table 5: Performance results (df=6)

			df = 6						
Bootstrap:	Parametric (book)			Parametric (MATLAB)			Non-Parametric		
Sample size:	100	500	2000	100	500	2000	100	500	2000
				mu = 0					
Mean CI length	1.078	0.492	0.251	1.081	0.492	0.250	0.964	0.482	0.247
Coverage ratio	0.908	0.953	0.965	0.903	0.954	0.964	0.769	0.860	0.897
				mu = -1					
Mean CI length	1.210	0.558	0.279	1.216	0.550	0.273	1.322	0.654	0.331
Coverage ratio	0.719	0.381	0.009	0.714	0.405	0.007	0.811	0.865	0.891
				mu = -2					
Mean CI length	1.492	0.682	0.340	1.496	0.681	0.338	1.728	0.852	0.436
Coverage ratio	0.573	0.115	0	0.563	0.114	0	0.788	0.868	0.875
				mu = -3					
Mean CI length	1.855	0.858	0.434	1.852	0.858	0.433	2.115	1.087	0.557
Coverage ratio	0.497	0.054	0	0.492	0.057	0	0.791	0.868	0.892

Question 3

In this exercise, we repeat Question 2 but we use simulations from the Stable Paretian Distribution as our true data. We wish to see the effect of model misspecification i.e. when we have symmetry but the tail shape is different.
To start with, we simulate a Stable Paretian distribution with two alpha values namely 1.6 and 1.8. Its beta is set to 0, scale=1 and location=0. We simulate data using the *stabgen* function which generates Stable Paretian distribution data points based on the provided parameters. Through this we obtain an estimate of the Stable Paretian distribution using the kernel density function. We then plot the distributions to compare with the Student t.

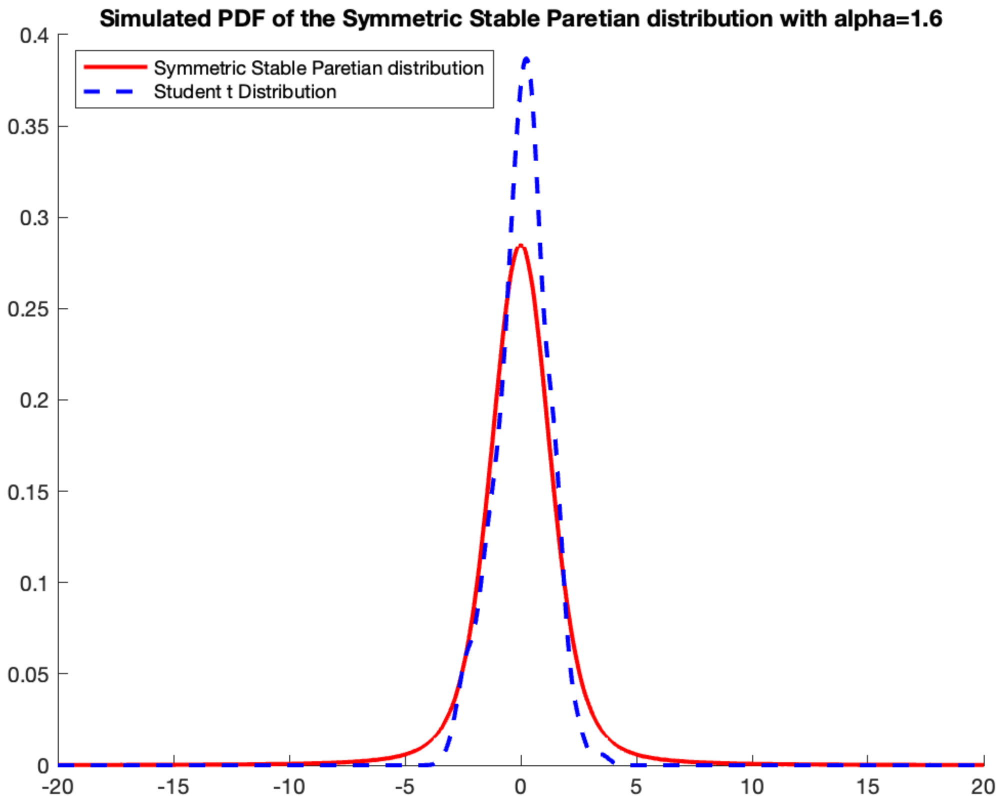

Figure 3: PDF of the Stable Paretian Distribution (alpha=1.6)

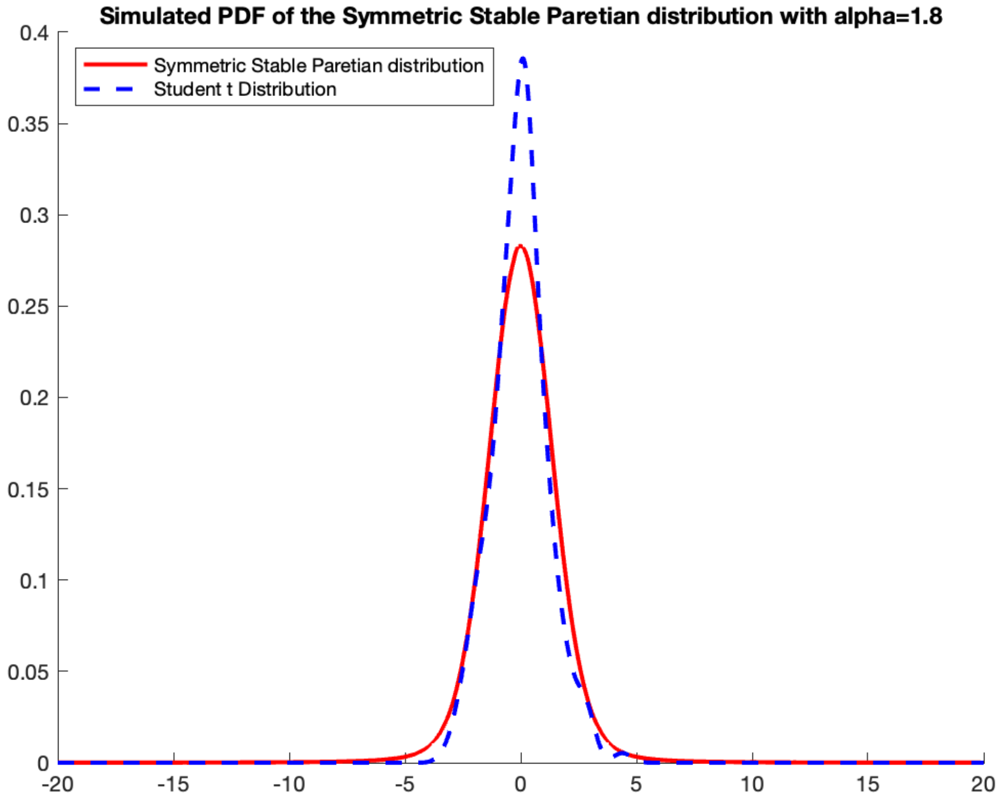

Figure 4: PDF of the Stable Paretian Distribution (alpha=1.8)

We calculate the theoretical Expected Shortfall based on Stoyanov et al. as well as a simulation of the ES. We then do exactly what we did for Question 2 in which we ran a bootstrap with B=1000 with different values of sample size=100, 500, 2000, and different degrees of freedom, df = 3 and 6. For every repetition we calculate the mean confidence interval length and whether the true Expected Shortfall, which we calculated using Stoyanov et al., lies within the confidence interval. We performed this for parametric and non-parametric bootstrap methods and also compare the parametric book method to the built-in MATLAB code. The code provided in Listing 6 is for df=3 and alpha=1.6. We ran the code multiple times for df=[3 6] and alpha (a1) =[1.6 1.8] to get all the values.

Listing 7: Computing the length of a bootstrap 90% confidence interval based on B bootstrap replications, and checking whether the interval contains the true ES

```
% Parameters
a1 = 1.6;                        % [1.6  1.8];
b = 0;  c = 1;  d = 0;  n = 1e7;
xvec = -20:0.01:20;
alpha = 0.1;                     % for the confidence interval
method = 1;
seed = 2;                        % to replicate the results
```

```
T = 200;
loc = 0; scale = 1;             % location and scale for student's t
rep = 1000;                     % #Repetitions
alpha = 0.1;
df = 3;                         % [3 6]
T_samples = [100 500 2000];     % Sample sizes
B = 1000;                       % #Bootstrap samples
initvec = [df loc scale];       % for parametric MLE


% Creating the kernel density estimate by generating 1e6 simulated IID
% variates from the stable distribution and simulating them.
eststab = stabgen(n,a1,b,c,d,seed);
[f,cd] = ksdensity(eststab,xvec);
figure, hold on, plot(cd, f , 'r-', 'linewidth', 2)

% t distribution
t_dist = trnd(df,T,1);
[f2,cd2] = ksdensity(t_dist,xvec);
plot(cd2, f2 , 'b--', 'linewidth', 2)

% adjusting the plot
xlim([-20 20])
legend('Simulated PDF of the Symmtric Stable Paretian distribution', '
    Location', 'NorthWest')
title('Simulated PDF of the Symmetric Stable Paretian distribution')
hold off


% Theoretical ES based on a tail probability alpha using the Stoyanov
    et al. result
[Theo_ES, Theo_VaR] = asymstableES(alpha , a1, b, c, d, method);

% Simulation of ES
nobs = 10^6;
P = stabgen(nobs, a1, b, c, d, seed);
VaR = quantile(P, alpha);
Plo=P(P<=VaR);
ES_simulated = mean(Plo);


% Initializing vectors
ESvec_nonparam = zeros(B, 1);
ESvec_param_book = zeros(B, 1);
ESvec_param_matlab = zeros(B, 1);
ci_length_nonparam = zeros(rep, length(T_samples));
```

```
ci_length_param_book = zeros(rep, length(T_samples));
ci_length_param_matlab = zeros(rep, length(T_samples));
ci_trueES_nonparam = zeros(rep, length(T_samples));
ci_trueES_param_book = zeros(rep, length(T_samples));
ci_trueES_param_matlab = zeros(rep, length(T_samples));


% True ES for student t
trueES = Theo_ES;


% Simulate "rep" repetitions of an IID T-length sequence of Stable
    Paretian
% For each rep claculate bootstrap CI 90% based on B bootstrap
    replications
for t = 1:length(T_samples)
    for r = 1:rep
        %%% generating data points from student t distribution
        data = stabgen(T_samples(t), a1, 0, scale, loc);

        %%% PARAMETRIC BOOTSTRAP %%%
        % 1. Using book MLE code
        mle_param_book = tlikmax0(data, initvec);

        % Computing simulated ES using estimated MLE parameters
        for b=1:B
            param_bs_sample_book = mle_param_book(2)+mle_param_book(3)*
                trnd(mle_param_book(1),T_samples(t),1);
            VaR_param_book = quantile(param_bs_sample_book, alpha);
            temp_book = param_bs_sample_book(param_bs_sample_book<=
                VaR_param_book);
            ESvec_param_book(b) = mean(temp_book);
        end

        % computing length of CI
        ci_param_book = quantile(ESvec_param_book,[alpha/2 1-alpha/2]);
        low_param_book = ci_param_book(1); high_param_book =
            ci_param_book(2);
        ci_length_param_book(r,t) = high_param_book-low_param_book;

        % checking wether the CI contains true ES
        ci_trueES_param_book(r,t) = (trueES>low_param_book)&(trueES<
            high_param_book);

        % 2. Using MATLAB built-in MLE function
        % output: [loc,scale,nu]
```

```
        mle_param_matlab = mle(data, 'Distribution', 'tLocationScale');

        % Computing simulated ES using estimated MLE parameters
        for b=1:B
            param_bs_sample_matlab = mle_param_matlab(1)+
                mle_param_matlab(2)*trnd(mle_param_matlab(3),T_samples(
                t),1);
            VaR_param_matlab = quantile(param_bs_sample_matlab, alpha);
            temp = param_bs_sample_matlab(param_bs_sample_matlab<=
                VaR_param_matlab);
            ESvec_param_matlab(b) = mean(temp);
        end

        % computing length of CI
        ci_param_matlab = quantile(ESvec_param_matlab, [alpha/2 1-alpha
            /2]);
        low_param_matlab = ci_param_matlab(1); high_param_matlab =
            ci_param_matlab(2);
        ci_length_param_matlab(r,t) = high_param_matlab-
            low_param_matlab;

        % checking wether the CI contains true ES
        ci_trueES_param_matlab(r,t) = (trueES>low_param_matlab)&(trueES
            <high_param_matlab);


        %%% NON-PARAMETRIC BOOTSTRAP %%%
        % book code
        % computing simulated ES
        for b=1:B
            ind = unidrnd(T_samples(t),[T_samples(t),1]);
            nonparam_bs_sample=data(ind);
            VaR_nonpara = quantile(nonparam_bs_sample, alpha);
            temp_nonparam = nonparam_bs_sample(nonparam_bs_sample<=
                VaR_nonpara);
            ESvec_nonparam(b) = mean(temp_nonparam);
        end

        % computing length of CI
        ci_nonparam = quantile(ESvec_nonparam, [alpha/2 1-alpha/2]);
        low_nonparam = ci_nonparam(1); high_nonparam = ci_nonparam(2);
        ci_length_nonparam(r,t) = high_nonparam-low_nonparam;

        % checking wether the CI contains true ES
        ci_trueES_nonparam(r,t) = (trueES>low_nonparam)&(trueES<
            high_nonparam);
```

```
    end
end
```

Listing 8: Function *stabgen* to generate Stable Paretian data points

```
function x=stabgen (nobs,a,b,c,d,seed)
if nargin<3, b=0; end, if nargin<4, c=1; end
if nargin<5, d=0; end, if nargin<6, seed=rand;end
z=nobs ;
rand('twister',seed),
V=unifrnd(-pi/2,pi/2,1,z);
rand('twister',seed+42),
W=exprnd(1,1,z);
if a==1
    x=(2/pi)*((( pi/2)+b*V).*tan(V)-b*log((W.*cos(V)) ./(( pi/2)+b*V)))
        ;
    x=c*x+d-(2/ pi )*d*log (d)*c*b;
else
    Cab=atan(b*tan(pi*a/2))/(a); Sab=(1+b^2*(tan((pi*a)/2))^2)^(1/(2*a)
        );
    A=(sin(a*(V+Cab))) ./((cos(V)).^(1/a));
    B0=(cos(V-a*(V+Cab)))./W; B = (abs(B0)).^((1-a)/a) ;
    x=Sab*A.*(B.*sign(B0)); x=c*x+d;
end
```

Listing 9: Function *asymstableES* to calculate the Expected Shortfall using Stoyanov et al.

```
function [ES, VaR] = asymstableES (xi, a, b, scale, mu, method)
if nargin < 3 , b=0; end, if nargin < 4 , mu=0; end
if nargin < 5 , scale=1; end, if nargin < 6 , method = 1; end

% Get q , the quantile from the S(0 ,1) distribution
opt=optimset('Display', 'off', 'TolX', 1e-6);
q=fzero(@stabcdfroot, -6, opt, xi, a, b); VaR=mu+scale*q;
if (q == 0)
    t0 = (1/a)*atan(b*tan(pi*a/2));
    ES = ((2*gamma((a-1)/a))/(pi-2*t0))*(cos(t0)/cos(a*t0)^(1/a));
return;
end
if (method==1), ES=(scale*Stoy(q, a, b)/xi)+mu;
else ES=(scale*stabletailcomp(q, a, b)/xi)+mu;
end

function diff = stabcdfroot(x, xi, a, b)
if exist('stableqkcdf.m', 'file'), F = stableqkcdf (x, [a, b], 1); %
    Nolan routine
```

```
else [~, F] = asymstab(x, a, b); %get the cdf of the asymmetric stable
end
diff = F - xi;

function tailcomp = stabletailcomp(q, a, b)
% direct integration of x*f(x) and use of asymptotic tail behavior
K = (a/pi)*sin(pi*a/2)*gamma(a)*(1-b); % formula for K_
ell = -120; M = ell; display = 0; term1 = K*(-M)^(1-a)/(1-a);
%term3 = quadl(@stableCVARint, ell, q, 1e-5, display, a, b);
term3 = integral(@(x) stableCVARint(x, a, b), ell, q);
tailcomp = term1 + term3;

function [g] = stableCVARint(x, a, b)
if exist('stableqkpdf.m', 'file'), den = stableqkpdf(x,[a, b], 1);
else den = asymstab(x, a, b)';
end
g = x.*den;
```

Table 6 shows the computed theoretical ES using the two methods and different alpha values for the Stable Paretian distribution. We can see they are quite similar.

Table 6: Expected Shortfall using different methods and parameters

	alpha=1.6	alpha=1.8
ES via Stoyanov et al.	–4.2911	-3.1433
ES via Simulation	-4.2574	-3.1336

Table 7: Performance results (df=3)

				df=3					
Bootstrap:	Parametric (book)			Parametric (MATLAB)			Non-Parametric		
Sample size:	100	500	2000	100	500	2000	100	500	2000
					a1= 1.6				
Mean CI length	3.599	1.618	0.837	3.593	1.621	0.836	4.130	2.652	1.777
Coverage ratio	0.772	0.751	0.505	0.774	0.749	0.496	0.591	0.703	0.773
					a1=1.8				
Mean CI length	1.783	0.828	0.414	1.785	0.827	0.414	1.858	1.192	0.759
Coverage ratio	0.823	0.868	0.810	0.827	0.863	0.811	0.651	0.739	0.793

Table 8: Performance results (df=6)

			df=6						
Bootstrap:	Parametric (book)			Parametric (MATLAB)			Non-Parametric		
Sample size:	100	500	2000	100	500	2000	100	500	2000
				a1= 1.6					
Mean CI length	3.433	1.622	0.838	3.433	1.622	0.836	4.053	2.507	1.761
Coverage ratio	0.766	0.753	0.518	0.769	0.753	0.504	0.583	0.688	0.766
				a1=1.8					
Mean CI length	1.765	0.833	0.414	1.769	0.835	0.413	2.121	1.205	0.725
Coverage ratio	0.814	0.886	0.798	0.81	0.859	0.803	0.631	0.769	0.806

As expected, the parametric bootstrap performs in both cases worse than the non-parametric. Again, as described in Question 2, this is due to the fact that the parametric distribution uses the wrongly assumed t-distribution for its computations, whereas the true distribution is the Stable Paretian. The non-parametric on the other hand, does not use an assumed distribution and hence, is more accurate but still not sufficiantly precise since we are not closely reaching the 0.9 coverage ratio.
As we can see from the tables above, the coverage ratio for both parametric bootstrap methods decreases with an increase in the sample size T from 100 to 2000. Whereas the coverage ratio for non-parametric bootstrap method increases with an increase in the sample size. We assume that this is due to the fact that with an increase in sample size, the simulation of the true distribution, which is the Stable Paretian, becomes more exact and gets further away from the wrongly assumed t-distribution used for the parametric bootstrap. This lowers the coverage ratio for the parametric. For the non-parametric case, the coverage ratio increases with an increase in the sample size as it does not use the assumed distribution.

If we compare the results for a1=1.6 and 1.8, we can see that all three methods perform better for the higher stability parameter a1=1.8 for the Stable Paretian. This comes from the fact that with a higher stability parameter, the Stable Paretian is more similar to the t-distribution, as we can also see in Figure 3. Comparing Table 7 and 8 shows that there is no big difference when using 3 or 6 degresse of freedom.

Question 4

In this last exercise we basically repeat the idea of Question 1, by using the NCT for both the true data generating process and also for the assumed distribution for the parametric bootstrap.

After setting all the necessary parameters, we compute, as previously done in the setting of Question 2, the simulation of the NCT distribution taking advantage of the function *stdnctpdfln_j*. Afterwards, to compare the distribution to the MATLAB built-in function *nctpdf*, we take again the exponential of the d.d.a NCT.

In the second step, we compute the true ES of the NCT distribution both via simulation and via integral definition, as previously performed in Question 2.
Finally, as done in the previous questions, for each repetition, we calculate a parametric and non parametric bootstrap 90% confidence interval, based on B bootstrap replications. We compute again for each repetition, both the length of the CI and we check whether the interval contains the true ES.
As done before, for the parametric bootstrap we need to compute the MLE to obtain the estimated parameters. We again use the *tlikmax0* function. Since this function estimates parameters for the regular t distribution, we modify it such that it estimates parameters for the location-scale NCT. Hence, for the input vector *initvec* we add a fourth parameter mu. We call the function *tlikmax0_modified* and it can be found in Listing 12. Additionally, we allow the function to take on two possible different ways of evaluating the pdf, namely via the d.d.a NCT (method='ddanct') discussed before and via the MATLAB built-in pdf (method='matlab'). Finally, we proceed as done in the previous questions.

As in Question 1, we expect again the parametric bootstrap to perform better than the non-parametric, since we now assume the correct distribution.

Listing 10: Computing the length of a bootstrap 90% confidence interval based on B bootstrap replications, and checking whether the interval contains the true ES - Non-central Student t distribution

```
% Parameters
alpha = 0.1;
loc = 0; scale = 1;
x = (-20:0.1:20)';
n = 10^6;
rep = 1000;                    % #Repititions
T_samples = [100 500 2000];    % Sample size
```

```
B = 1000;                     % #Bootstrap samples
seed = 2;                     % set seed value to replicate results
initvec = [df loc scale mu];  % for parametric MLE
method = 'ddanct';            % ddanct or matlab
mu = 0;                       % [0 -1 -2 -3]
df = 3;                       % [3 6]

% Initializing vectors
ESvec_nonparam = zeros(B, 1);
ESvec_param_book = zeros(B, 1);
ESvec_param_matlab = zeros(B, 1);
ci_length_nonparam = zeros(rep, length(T_samples));
ci_length_param_book = zeros(rep, length(T_samples));
ci_length_param_matlab = zeros(rep, length(T_samples));
ci_trueES_nonparam = zeros(rep, length(T_samples));
ci_trueES_param_book = zeros(rep, length(T_samples));
ci_trueES_param_matlab = zeros(rep, length(T_samples));


% computing pdf of the location-zero, scale-one NCT
% Use stdnctpdfln_j: Program Listing 9.2
nct_book = exp(stdnctpdfln_j(x, df, mu));

% MATLAB built in function
nct_matlab = nctpdf(x, df, mu);

figure, plot(x, nct_book, 'b','LineWidth', 2)
hold on, plot(x, nct_matlab, 'r--','LineWidth', 2)
xlim([-20 20])
legend('Book code','MATLAB built-in')
title('Simulated PDF of the NCT')
hold off

% ES via simulation
norm=normrnd(mu,1,n,1); chi2=chi2rnd(df,n,1);
nct_rnd_sample = loc+scale*(norm./sqrt(chi2/df));
VaR = quantile(nct_rnd_sample, alpha);
temp_2 = nct_rnd_sample(nct_rnd_sample <= VaR);
ES_simulated = mean(temp_2);

% ES using integral definition of NCT
c01_nct = nctinv(alpha , df, mu);
ES_01_nct = @(x) x.*nctpdf(x, df, mu);
ES_nct_int= integral(ES_01_nct, -Inf, c01_nct)/alpha;

% True ES for student t
```

```
trueES = ES_nct_int;


% Simulate "rep" repetitions of an IID T-length sequence of NCT
% For each rep claculate bootstrap CI 90% based on B bootstrap replications
for t = 1:length(T_samples)
    for r = 1:rep
        % Generating data points from NCT distribution
        norm=normrnd(mu,1, T_samples(t), 1); chi2=chi2rnd(df, T_samples(t), 1);
        data = loc+scale*(norm./sqrt(chi2/df));


        %%% PARAMETRIC BOOTSTRAP %%%
        % 1. Using Book MLE code (method='ddanct') and 2. using MATLAB built-in MLE function (method='matlab')
        mle_param_book = tlikmax0_modified(data, initvec, method);

        % Computing simulated ES using estimated MLE parameters
        for b=1:B
            rnd_norm = normrnd(mle_param_book(4),1, T_samples(t), 1); rnd_chi2=chi2rnd(mle_param_book(1), T_samples(t), 1);
            rnd_nct = rnd_norm./sqrt(rnd_chi2/mle_param_book(1));
            param_bs_sample_book = mle_param_book(2)+mle_param_book(3)*rnd_nct;
            VaR_param_book = quantile(param_bs_sample_book, alpha);
            temp_book = param_bs_sample_book(param_bs_sample_book<=VaR_param_book);
            ESvec_param_book(b) = mean(temp_book);
        end

        % Computing length of CI
        ci_param_book = quantile(ESvec_param_book,[alpha/2 1-alpha/2]);
        low_param_book = ci_param_book(1); high_param_book = ci_param_book(2);
        ci_length_param_book(r,t) = high_param_book-low_param_book;

        % Checking whether the CI contains the true ES
        ci_trueES_param_book(r,t) = (trueES>low_param_book)&(trueES<high_param_book);


        %%% NON-PARAMETRIC BOOTSTRAP %%%
        % Book code: Program Listing 2.9
        for b=1:B
```

```
            ind = unidrnd(T_samples(t),[T_samples(t),1]);
            nonparam_bs_sample=data(ind);
            VaR_nonpara = quantile(nonparam_bs_sample, alpha);
            temp_nonparam = nonparam_bs_sample(nonparam_bs_sample<=VaR_nonpara);
            ESvec_nonparam(b) = mean(temp_nonparam);
        end

        % Computing length of CI
        ci_nonparam = quantile(ESvec_nonparam, [alpha/2 1-alpha/2]);
        low_nonparam = ci_nonparam(1); high_nonparam = ci_nonparam(2);
        ci_length_nonparam(r,t) = high_nonparam-low_nonparam;

        % Checking whether the CI contains the true ES
        ci_trueES_nonparam(r,t) = (trueES>low_nonparam)&(trueES<high_nonparam);
    end
end
```

Listing 11: Function *stdnctpdfln_j* that computes the log density and returns the direct approximation to the NCT

```
function pdfln = stdnctpdfln_j(x, df, mu) % Program Listing 9.2
vn2 = (df+1)/2; rho=x.^2;
pdfln = gammaln(vn2) - 1/2*log(pi*df) - gammaln(df/2) - vn2*log1p(rho/df);
if (all(mu == 0)), return, end
idx = (pdfln >= -37);
gcg = mu.^2; pdfln = pdfln - 0.5*gcg; xcg = x .* mu;
term = 0.5*log(2) + log(xcg) - 0.5*log(max(realmin,df+rho));
term(term == -inf) = log(realmin); term(term == +inf) = log(realmax); maxiter = 1e4; k = 0;
logterms = gammaln((df+1+k)/2) - gammaln(k+1) - gammaln(vn2) + k*term; fractions = real (exp( logterms ) ) ; logsumk = log ( fractions ) ;
while (k < maxiter)
k = k + 1;
logterms = gammaln((df+1+k)/2) - gammaln(k+1) - gammaln(vn2) + k*term(idx); fractions = real(exp(logterms-logsumk(idx)));
logsumk(idx) = logsumk(idx) + log1p(fractions);
idx(idx) = (abs(fractions) > 1e-4); if (all(idx == false)), break, end
end
pdfln = real (pdfln+logsumk) ;
end
```

Listing 12: Function *tlikmax0_modified* that computes the MLE

```
function MLE = tlikmax0_modified(x,initvec,method)
tol=1e-5;
opts=optimset('Disp', 'none', 'LargeScale', 'Off', 'TolFun', tol, 'TolX', tol, 'Maxiter',200);
MLE = fminunc(@(param)tloglik_modified(param, x, method), initvec, opts);
end

function ll = tloglik_modified(param,x, method)
v=param(1);      % df
l=param(2);      % loc
c=param(3);      % scale
m=param(4);      % mu


if v<0.01, v=rand; end % Ad hoc way of preventing negative values
if c<0.01, c=rand; end

if method == 'ddanct'
    % Use stdnctpdfln_j: Program Listing 9.2
    z = (x-l)./c;
    ll = -log(c)+stdnctpdfln_j(z, v, m);
elseif method == 'matlab'
    % Use matlab nct pdf fct & take log
    z = (x-l)./c;
    ll = -log(c)+log(nctpdf(z, v, m));
else disp('Invalid method used!')
end
ll = -sum(ll);
end
```

Table 9: Expected Shortfall using different methods and parameters (df=3)

	mu=0	mu=-1	mu=-2	mu=-3
ES via simulation	-2.908	-5.369	-8.2758	-11.4457
ES via integral definition	-2.911	-5.362	-8.2637	-11.4254

Table 10: Expected Shortfall using different methods and parameters (df=6)

	mu=0	mu=-1	mu=-2	mu=-3
ES via simulation	-2.175	-3.696	-5.382	-7.198
ES via integral definition	-2.187	-3.7007	-5.384	-7.189

Table 11: Performance results (df=3)

			df = 3						
Bootstrap:	Parametric (ddanct)			Parametric (matlab)			Non-Parametric		
Sample size:	100	500	2000	100	500	2000	100	500	2000
				mu = 0					
Mean CI length	2.211	1.045	0.527	2.229	1.038	0.533	1.834	0.967	0.506
Coverage ratio	0.827	0.909	0.924	0.842	0.910	0.921	0.735	0.843	0.880
				mu = -1					
Mean CI length	3.608	1.742	0.890	3.700	1.750	0.8922	3.057	1.582	0.853
Coverage ratio	0.838	0.914	0.931	0.840	0.902	0.923	0.783	0.843	0.871
				mu = -2					
Mean CI length	5.723	2.580	1.323	5.799	2.592	1.325	4.463	2.354	1.261
Coverage ratio	0.863	0.910	0.918	0.847	0.930	0.917	0.749	0.834	0.875
				mu = -3					
Mean CI length	7.814	3.585	1.837	7.577	3.495	1.786	5.789	3.237	1.740
Coverage ratio	0.87	0.922	0.939	0.873	0.899	0.932	0.716	0.836	0.860

Table 12: Performance results (df=6)

			df = 6						
Bootstrap:	Parametric (ddanct)			Parametric (matlab)			Non-Parametric		
Sample size:	100	500	2000	100	500	2000	100	500	2000
				mu = 0					
Mean CI length	1.048	0.494	0.249	1.076	0.494	0.249	0.967	0.483	0.248
Coverage ratio	0.831	0.867	0.909	0.846	0.890	0.908	0.796	0.849	0.898
				mu = -1					
Mean CI length	1.460	0.672	0.338	1.434	0.667	0.335	1.316	0.655	0.336
Coverage ratio	0.849	0.867	0.884	0.840	0.887	0.909	0.802	0.879	0.888
				mu = -2					
Mean CI length	1.966	0.890	0.446	1.959	0.874	0.438	1.696	0.864	0.440
Coverage ratio	0.858	0.897	0.915	0.865	0.896	0.898	0.76	0.868	0.897
				mu = -3					
Mean CI length	2.586	1.134	0.562	1.959	1.112	0.557	1.762	1.086	0.551
Coverage ratio	0.905	0.909	0.926	0.865	0.904	0.901	0.799	0.885	0.876

The results displayed in Table 11 and 12 reflect partly what we expected to obtain. Again, in fact, using the true DGP allows a better performance of the parametric bootstrap compared to the non-parametric computation, both in terms of coverage accuracy and length of the CIs. However, for df=3, the accuracy doesn't increase with the sample size, as expected. The most accurate coverage ratio is achieved with a sample size of 500. Increasing it to 2000 further increases the ratio. For the non-parametric however, as expected, the coverage ratio gets more accurate as we increase the sample size. But we can see that the length of the intervals decreases for all methods as we increase the sample size.
For df=6 on the other hand, increasing the sample size to 2000 makes the parametric models for mu=0 and mu=-1 more accurate. For mu=-2 and mu=-3 again the sample size of 500 yields the most accurate performance. This is not quite what we expected.

Further, if we compare the two parametric methods, we can see that there is no obvious outstanding model. We can also observe, that decreasing the noncentrality parameter to a more negative value, leads to an increase of the confidence interval, since we further skew the distribution. Finally, comparing the two tables we can see that for df=3 we get longer CIs. Hence, since the coverage ratio for both df's is similar, the overall performance for df=6 is better.