Assignment 2, due 22.11.2022

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

Prof. Dr. Marc Paolella

presented by

Dennis Arend: 21-742-135

Ernest Digore: 21-727-896

Kilian Sennrich: 18-051-060

Abstract

The purpose of the assignment is to explore estimation methods for Student's t distributions and subsequent Expected Shortfall (ES) estimations using parametric and non-parametric bootstraps. Both the central and non-central Student's t distribution are considered. Consequently, we evaluate cases when the model is mis-specified, and test the performance of the two methods of bootstrapping, finding that the latter performs better in these instances.

Contents

1 Question 1

As preparation for further exploration of ES-estimation using Student's t distributions, both parametric and non-parametric bootstrap method to generate samples are utilized. To gain a better understanding of the perks and downsides of the two methods, their results and accuracy are then compared. The underlying data for this comparison is generated based on fixed parameters set as:

$$loc = 1.0, \quad scale = 2.0, \quad \xi = 0.05, \quad df = 4, \quad confidence = 90\%$$

with parameters *loc*, *scale*, *df* and ξ used to calculate the true ES analytically according to Code-Section 1, as well as data samples of the central Student's t distribution. This value will later on be used to calculate the accuracy of the two simulation methods of parametric and non-parametric bootstrap, by checking how often the true ES falls into the 90% confidence interval of the simulated ES samples.

The parametric bootstraps are not calculated on the parameters directly, but on Maximum-Likelihood estimates (MLE) of the underlying data samples instead. The function is shown in Code-Section A.2. These parameter estimates, in combination with Student's t random variates, are subsequently used to generate $Breps = 1000$ parametric bootstraps of size T each. The Es of each of the resulting 1000 bootstraps can then be analyzed using the VaR, given a distribution of ES values for which a confidence interval can be calculated. The non-parametric bootstrap spares itself the step of Maximum-Likelihood estimation, as it *directly* samples from the underlying data. The resulting 1000 bootstraps (each containing T samples, are then processed identical to the parametric bootstraps, calculating individual ES estimates to subsequently create a confidence interval.

The two approaches are executed $reps = 1000$ times, in order to compare a series of confidence intervals, allowing the calculation of the average accuracy of the two approaches, i.e. the probability of a confidence interval containing the "true"/analytical ES. The influence of the sample size parameter T is examined in Tables 1 and 2 comparing results for $T = 250, 500$ and 2000. As to be expected, the accuracy of both methods increases with increasing sample size, while the confidence interval length decreases. The parametric bootstrap approach shows a higher accuracy than the non-parametric bootstrap, since the method is based on the distribution of data generating process (Student's t).

Table 1 and 2 show the results of the two approaches:

	Parametric Bootstrap		
	$T = 250$	$T = 500$	$T = 2000$
CI length	3.83	4.07	2.13
CI accuracy	46.10%	60.40%	70.80%

Table 1: Length and accuracy of the confidence interval generated from a *parametric* bootstrap for different sample sizes T.

	Non-Parametric Bootstrap		
	$T = 250$	$T = 500$	$T = 2000$
CI length	4.53	3.61	2.03
CI accuracy	50.70%	56.90%	65.10%

Table 2: Length and accuracy of the confidence interval generated from a *non-parametric* bootstrap for different sample sizes T.

```
% Question 1
loc=1; scale=2; xi = 0.05; df = 4; confidence = 0.9;
T=[250;500;2000]; rep=1000; Breps = 1000;

for h = 1:size(T,1)
    question1(loc,scale,xi,df,T(h),rep,Breps,confidence);
end


function question1(loc,scale,xi,df,T,rep,Breps,confidence)
fprintf('T: %.0f\n',T)

% True ES
c01 = tinv(xi , df);
ES01 = -tpdf(c01,df)/tcdf(c01,df) * (df+c01^2)/(df-1);
trueES = loc+scale*ES01;
fprintf('True ES: %.4f\n',trueES)

[CI,CI_nonpar] = CI_calculation(loc,scale,xi,df,T,rep,Breps,confidence,0,0,0,0,0,0) ;

% length of the CI's
CI_length = CI(:,2)-CI(:,1);
CI_nonpar_length = CI_nonpar(:,2)-CI_nonpar(:,1);
fprintf('\nthe parametric approach has an average CI length of %.5f\n',mean(CI_length));
fprintf('the non-parametric approach has an average CI length of %.5f\n\n',mean(CI_nonpar_length));

% is the ES in the CI?
CI_hasTrueEs = ((trueES <= CI(:,2)) & (trueES >= CI(:,1)));
CI_nonpar_hasTrueEs = ((trueES <= CI_nonpar(:,2)) & (trueES >= CI_nonpar(:,1)));
fprintf('the parametric CI has the true ES %.2f%% of the time\n',sum(CI_hasTrueEs)/rep*100);
fprintf('the non-parametric CI has the true ES %.2f%% of the
    time\n\n',sum(CI_nonpar_hasTrueEs)/rep*100);
```

Code-Section 1: Calculation of the accuracy of parametric and non-parametric bootstrap approaches for ES estimation. This code uses Code-Section A.1 and Code-Section A.2.

2 Question 2

As a continuation of Question 1, we replace the central Student‘s t distribution for the *non-central* Student‘s t distribution (NCT). For different non-centrality parameters μ and degrees of freedom df, the goal is to calculate the accuracy and length of confidence intervals using the parametric and non-parametric bootstrapping approach of Code-Section 1. Thus, we ...

1. ... simulate the NCT, using a self-made function instead of Matlab‘s fast and readily-available built-in function *nctrnd*.

2. ... compute the true ES for the NCT using both simulation and numeric integration.

The subsequent comparison of parametric and non-parametric bootstrapping is straight forward and nearly identical to the procedure of Code-Section 1 (and we again use the code from Appendix A.1 for this step), since we (wrongfully, but knowingly) estimate parameters as if working with a central Student‘s t distribution for the parametric bootstrap. We expect the non-parametric bootstrap to perform much better, particularly for cases where the non-centrality parameter μ is not zero. To illustrate the differences between the methods, we once again use multiple values for some parameters, namely:

$$location = 0.0, \quad scale = 1.0, \quad \xi = 0.05, \quad df = [3, 6], \quad \mu = [0, -1, -2, -3]$$
$$confidence = 90\%, \quad T = [100, 500, 2000], \quad rep = 1000, \quad Breps = 1000$$

In order to generate iid. non-central student t random variables, we created our own function that follows the non-central Student’s t distribution using the general definition:

$$T = \frac{X}{\sqrt{\frac{Y}{k}}}, \text{ where } \; X \sim N(\mu, 1) \text{ and } \; Y \sim \; \chi^2(k, \theta) \tag{1}$$

Namely, we generate random variables for both the normal and χ^2 distributions by taking advantage from the fact that a combination of two independently generated random variables results in another random variable. The implementation can be found in the Code-Section 2. To verify that both the built-in function and our custom function generate R.V’s under the same distribution, we sampled 10,000 R.V’s with both the *nctrnd* and *nctrnd_1* (see appendix A.4). The result shows, that the two functions converge nicely, when calculating the mean of the sample after each generation of a new random variate. The results of the expected shortfall for the simulation is shown in Table 3 below:

ES01 using the simulated non-central t

	$\mu = 0$	$\mu =$ -1	$\mu =$ -2	$\mu =$ -3
df = 3	-6.9969	-9.449	-12.215	-15.243
df = 6	-4.0276	-5.084	-6.2266	-7.4379

Table 3: Computation of the ES01 using the simulated non-central t random variables

ES01 using the integral definition of the NCT

	$\mu = 0$	$\mu =$ -1	$\mu =$ -2	$\mu =$ -3
df = 3	-7.0024	-9.4514	-12.2261	-15.2650
df = 6	-4.0325	-5.0874	-6.2290	-7.4466

Table 4: Computation of the ES01 using the integral definition of the NCT

Parametric Bootstrap

T	df	μ	CI length	CI Accuracy
100	3.0	0.0	2.06	54.80%
		-1.0	2.69	52.80%
		-2.0	3.78	48.80%
		-3.0	4.82	32.30%
	6.0	0.0	1.03	55.90%
		-1.0	1.14	59.10%
		-2.0	1.37	53.10%
		-3.0	1.79	65.60%
500	3.0	0.0	0.97	48.20%
		-1.0	1.15	13.60%
		-2.0	1.56	4.20%
		-3.0	2.06	37.20%
	6.0	0.0	0.49	15.60%
		-1.0	0.54	61.60%
		-2.0	0.63	41.70%
		-3.0	0.77	41.60%
2000	3.0	0.0	0.48	27.50%
		-1.0	0.57	6.10%
		-2.0	0.76	0.50%
		-3.0	1.02	0.20%
	6.0	0.0	0.24	33.90%
		-1.0	0.27	4.40%
		-2.0	0.32	1.00%
		-3.0	0.38	4.80%

Non-Parametric Bootstrap

T	df	μ	CI length	CI Accuracy
100	3.0	0.0	1.82	49.00%
		-1.0	3.05	56.60%
		-2.0	4.49	24.50%
		-3.0	6.13	47.30%
	6.0	0.0	1.57	49.80%
		-1.0	1.20	60.20%
		-2.0	1.57	37.10%
		-3.0	2.13	48.30%
500	3.0	0.0	0.95	48.10%
		-1.0	1.59	52.20%
		-2.0	2.37	51.20%
		-3.0	3.20	42.70%
	6.0	0.0	0.47	18.50%
		-1.0	0.64	62.40%
		-2.0	0.85	68.60%
		-3.0	1.08	60.10%
2000	3.0	0.0	0.48	27.90%
		-1.0	0.80	62.50%
		-2.0	1.19	62.60%
		-3.0	1.63	57.70%
	6.0	0.0	0.24	35.30%
		-1.0	0.33	56.00%
		-2.0	0.43	67.40%
		-3.0	0.55	44.60%

Table 5: Results of Code-Section 3 for different choices of sample size T, degrees of freedom *df* and non-centrality μ.

From the results above, we indeed confirm that the non-parametric bootstrap does better than the parametric. For T = 100 this is not entirely visible as the sample size is too small. For the rest, we see that as we derail more from the non-scaled (Student-t) distribution, the gap in performance of the two methods increases (i.e. the parametric CI has the true ES less often). Lastly, it is worth noticing that the CI length decreases as the sample size increases.

```
function t = nctrnd_1(mu,df,rows,col)
t = normrnd(mu,1,rows,col)./ sqrt(chi2rnd(df,rows,col)/df);
```

Code-Section 2: Function *nctrnd_1* used in Code-Section 3 and 6.

```
% Question 2
loc=0; scale=1; xi = 0.05; T=[100;500;2000]; rep=1000; Breps = 1000; confidence = 0.9;
df = [3;6]; mu = linspace(0,-3,4)';
rng(0,'twister');

for h = 1:size(T,1)
    for i = 1:size(df,1)
        for j = 1:size(mu,1)
            question2(loc,scale,xi,df(i),T(h),rep,Breps,confidence,mu(j));
        end
    end
end

function question2(loc,scale,xi,df,T,rep,Breps,confidence,mu)
fprintf('T= %.0f, mu= %.1f, df=%.0f:\n\n',T,mu,df)

% True ES
% Simulation
u = nctrnd_1(mu,df,1e6);
use = u(u<quantile(u,xi));
trueES_sim = mean(use);
fprintf('Simulation True ES: %.4f\n',trueES_sim)
% Numeric Integration
[trueES_numint,~] = nctES(xi,df,mu);
fprintf('Num Integ. True ES: %.4f\n\n',trueES_numint)


[CI,CI_nonpar] = CI_calculation(loc,scale,xi,df,T,rep,Breps,confidence,mu,0,0,0,0,0);

% length of the CI's
CI_length = CI(:,2)-CI(:,1);
CI_nonpar_length = CI_nonpar(:,2)-CI_nonpar(:,1);
fprintf('the parametric approach has an average CI length of %.5f\n',mean(CI_length));
fprintf('the non-parametric approach has an average CI length of %.5f\n\n',mean(CI_nonpar_length));

% is the ES in the CI?
CI_hasTrueEs = ((trueES_sim <= CI(:,2)) & (trueES_sim >= CI(:,1)));
CI_nonpar_hasTrueEs = ((trueES_sim <= CI_nonpar(:,2)) & (trueES_sim >= CI_nonpar(:,1)));
fprintf('the parametric CI has the true ES %.2f%% of the time\n',sum(CI_hasTrueEs)/rep*100);
fprintf('the non-parametric CI has the true ES %.2f%% of the
    time\n\n',sum(CI_nonpar_hasTrueEs)/rep*100);
```

Code-Section 3: Calculation of the accuracy of parametric and non-parametric bootstrap approaches for ES estimation on a **non-central** Student‘s t distribution.

3 Question 3

Question three brings back the symmetric stable Paretian distribution from assignment 1, with the goal of calculating the CI lengths and accuracies once again using parametric and non-parametric bootstraps, however this time applied to the symmetric stable Paretian distribution with parameters $a = [1.6, 1.8]$, $b = 0$, $c = 1$, $d = 0$. Once again, this is done for multiple parameter values (in this case of a) in order to better understand the behavior of the respective methods under changes in the underlying distribution. The parametric bootstrap is expected to perform worse than the non-parametric bootstrap, as the parameters used for the parametric bootstrap stem from a MLE which wrongfully assumes the Student t as the underlying distribution (despite it actually being a symmetric stable Paretian distribution). The results of these computations are shown in Table 6 and were computed using the code from Code-Section 4. This code uses functions from Code-Section A.1 in the appendix.

For the DGP of iid symmetric stable Paretian we use the *stablernd* function. The true ES could be computed in two ways: using the command *asymstableES* or taking the specific quantile from the DGP and averaging the components that are lower than the specific q(0.05), also called VaR. The degrees of freedom *df* which is passed to the *CI_calculation* function doesn't serve any real purpose, except for the MLE function *tlikmax0*, where it functions as an initial value for the MLE process, alongside the location and scale parameters. Additionally, passing values of μ to the *CI_calculation* function does not yield any effect, as it is not utilized in any code function applied to this question.

The results of Table 6 show the non-parametric bootstrap mostly out-performing the parametric bootstrap, however not very strongly. Especially for lower sample sizes the effect tends to disappear, which could be attributed to the fact that the underlying DGP is also of a size relative to the sample size, thus for $T = 100$ the generated stable Paretian dataset represents its underlying distribution less "accurately", such that the difference to the Student t distribution reduces, leading to equally good results for the parametric bootstrap method. Noticably, with an increase in stable parameter a, the overall confidence interval length decreases for both methods, and overall CI accuracy increases. This can be attributed to the fact that for larger stable parameter values, the stable Paretian distribution becomes more similar to a Student t distribution (since they are both symmetric in this case). Additionally, as for all results, an increase in sample size T generally leads to higher accuracy and smaller CI lengths. The low overall accuracy level can partly be attributed to the fact that we work with a relatively small tail parameter $\xi = 0.05$.

Parametric Bootstrap

a	T	CI length	CI Accuracy
1.6	100	4.99	28.20%
	500	3.11	37.50%
	2000	1.23	42.10%
1.8	100	3.21	32.27%
	500	1.29	49.58%
	2000	0.66	42.67%

Non-Parametric Bootstrap

a	T	CI length	CI Accuracy
1.6	100	4.20	27.82%
	500	2.67	39.19%
	2000	1.01	43.21%
1.8	100	2.43	33.40%
	500	0.98	47.40%
	2000	0.58	45.64%

Table 6: Results of Code-Section 3 for different choices of $a = [1.6, 1.8]$ and sample sizes T. In this case, the code is applied on a symmetric stable Paretian dataset generated using *stablernd* with parameters $a = [1.6, 1.8]$, $b = 0$, $c = 1$, $d = 0$.

```matlab
% Question 3
loc=0; scale=1; T=[100;500;2000]; rep=1000; Breps = 1000; confidence = 0.9;
df = [3;6]; mu = 0; limit = 6; xi = 0.05;
a = [1.6;1.8]; b=0;c=1;d=0;
rng(0,'twister');

for h = 1:size(T,1)
    for k = 1:size(a,1)
        question3(loc,scale,xi,df(1),T(h),rep,Breps,confidence,mu,a(k),b,c,d);
    end
end


function question3(loc,scale,xi,df,T,rep,Breps,confidence,mu,a,b,c,d)
fprintf('T= %.0f, mu= %.1f, df=%.0f, a = %.1f:\n\n',T,mu,df,a)

% True ES using Stoyanov et al
x = stablernd(a,b,c,d,1e6,1);
q = quantile(x,xi);
trueES_stoy = mean(x(x< q));
%trueES_stoy = asymstableES(xi,alpha(1),b,d,c,1);
fprintf('Simulation True ES: %.4f\n',trueES_stoy)

[CI,CI_nonpar] = CI_calculation(loc,scale,xi,df,T,rep,Breps,confidence,mu,a,b,c,d,0);

% length of the CI's
CI_length = CI(:,2)-CI(:,1);
CI_nonpar_length = CI_nonpar(:,2)-CI_nonpar(:,1);
fprintf('the parametric approach has an average CI length of %.5f\n',mean(CI_length));
fprintf('the non-parametric approach has an average CI length of %.5f\n\n',mean(CI_nonpar_length));

% is the ES in the CI?
CI_hasTrueEs = ((trueES_stoy <= CI(:,2)) & (trueES_stoy >= CI(:,1)));
CI_nonpar_hasTrueEs = ((trueES_stoy <= CI_nonpar(:,2)) & (trueES_stoy >= CI_nonpar(:,1)));
fprintf('the parametric CI has the true ES %.2f%% of the time\n',sum(CI_hasTrueEs)/rep*100);
fprintf('the non-parametric CI has the true ES %.2f%% of the
    time\n\n',sum(CI_nonpar_hasTrueEs)/rep*100);
```

Code-Section 4: Computing the ES of the symmetric stable distribution and implementing both parametric and non-parametric bootstraps assuming a Student t distribution

4 Question 4

In a variation of question one, we now compare the accuracy of confidence intervals based on the parametric and non-parametric bootstrap with the twist, that the parametric bootstrap utilizes parameter estimates generated from a MLE method actually fitted to estimate a location-scale NCT. With parameters remaining the same as in question 2, namely:

$$location = 0.0, \quad scale = 1.0, \quad \xi = 0.05, \quad df = [3,6], \quad \mu = [0,-1,-2,-3]$$
$$confidence = 90\%, \quad T = [100, 500, 2000], \quad rep = 1000, \quad Breps = 1000$$

we implement this task in Code-Section 6, generating the results shown in Table 7. The most important part of this exercise is the formulation of a MLE method that can reliably estimate parameters of a location-scale NCT. The code for this is presented in Code-Section 5. The optimization condition $\frac{\partial L}{\partial a_j} = 0$, at the point $\hat{a}$ where L is our likelihood function and a_j, $j \in \{1, 2, ...n\}$ different parameters of $f(\overrightarrow{x}, \overrightarrow{a})$, helps us in finding the best estimate of the parameter to maximize the likelihood function. Taking the logarithm of the likelihood function allows as to sum the log p.d.f instead of taking the product of all the functions, which would be more cumbersome. Because Matlab has a built-in minimization function *fminunc*, we minimize w.r.t the negative sum of log densities, which is the same as the maximization of the positive sum.

After finding the true ES of the location-scale NCT, for the DGP, we use the created function from Q2: *nctrnd_1* that takes advantage of the definition of the NCT random variable. The bootstrap procedure is identical to the previous questions', except that the adapted MLE for the location-scale NCT is used. This time, we expect the parametric bootstrap to do better than the non-parametric one, as both the data and the MLE agree w.r.t the probability distribution. This is confirmed in the tables below.

The first thing to notice in the results is that, different to the results in question 3, the parametric bootstrap now outperforms the non-parametric bootstrap method basically every time. With an increase in the non-centrality parameter μ, the size of the generated confidence intervals increases, as well as being larger for iterations with larger degrees of freedom df. As to be expected, with an increase in iterations T the accuracy of the generated confidence intervals also increases, from around 50% for $T = 100$ up to 85% for $T = 2000$. The computation time of the two applied MLE methods (Matlab's built-in function *nctpdfWrapper* and Paolella's *stdnctpdfln_j* function) varies obviously, as one is an approximation of the other.

Parametric Bootstrap (d.d.a. Approx.)

T	df	μ	CI length	CI Accuracy
100	3.0	0.0	1.93	66.50%
		-1.0	3.24	63.50%
		-2.0	4.86	64.00%
		-3.0	6.80	64.50%
	6.0	0.0	0.97	72.90%
		-1.0	1.33	74.20%
		-2.0	1.71	72.70%
		-3.0	2.18	71.50%
500	3.0	0.0	0.96	69.90%
		-1.0	1.61	68.50%
		-2.0	2.38	69.10%
		-3.0	3.30	72.90%
	6.0	0.0	0.48	83.40%
		-1.0	0.66	84.00%
		-2.0	0.87	83.30%
		-3.0	1.10	83.60%
2000	3.0	0.0	0.48	69.90%
		-1.0	0.82	68.00%
		-2.0	1.19	67.00%
		-3.0	1.62	69.80%
	6.0	0.0	0.24	86.10%
		-1.0	0.33	86.00%
		-2.0	0.39	71.30%
		-3.0	0.59	70.80%

Non-Parametric Bootstrap (d.d.a. Approx.)

T	df	μ	CI length	CI Accuracy
100	3.0	0.0	1.78	52.20%
		-1.0	3.06	53.50%
		-2.0	4.46	51.60%
		-3.0	6.07	52.10%
	6.0	0.0	0.92	61.10%
		-1.0	1.22	61.80%
		-2.0	1.61	60.40%
		-3.0	2.03	58.80%
500	3.0	0.0	0.94	59.70%
		-1.0	1.59	59.50%
		-2.0	2.30	61.90%
		-3.0	3.22	61.20%
	6.0	0.0	0.48	70.40%
		-1.0	0.63	67.60%
		-2.0	0.86	69.60%
		-3.0	1.07	70.10%
2000	3.0	0.0	0.48	59.90%
		-1.0	0.81	59.60%
		-2.0	1.18	59.20%
		-3.0	1.63	62.10%
	6.0	0.0	0.24	74.20%
		-1.0	0.32	73.30%
		-2.0	0.36	67.70%
		-3.0	0.47	65.30%

Table 7: Results of Code-Section 5 and 6 for different choices of sample size T, degrees of freedom df and non-centrality μ.

The results of the d.d.a. Approximation method, in this instance, when both the DGP and the estimation model agree w.r.t. the distribution, show that the parametric bootstrap outperforms the non-parametric one in all the cases. The same is true for most of the instances using the built-in Matlab *nctpdf* command, with the results shown below. Further, consistent with the previous cases, as the sample size increases, the length of the CI decreases. Concerning the speed of the two methods, it (surprisingly so) seems that the "d.d.a." approximation method by Paolella runs about eight times faster than Matlab's built-in function *nctpdfWrapper*, with about 9.34sec. for 1000 iterations, compared to 72.39sec. The speed advantage could be rooted in the *stdnctpdfln_j* function using approximate methods, thus leading to increased computation times with a penalty in accuracy, or it is simply a superior implementation. Either way, the resulting accuracy is sufficient for our analysis.

Parametric Bootstrap (Matlab pdf)

T	df	μ	CI length	CI Accuracy
100	3.0	0.0	2.05	55.20%
		-1.0	3.43	55.00%
		-2.0	5.96	54.00%
		-3.0	7.26	55.20%
	6.0	0.0	1.02	62.80%
		-1.0	1.45	60.60%
		-2.0	1.88	60.10%
		-3.0	2.29	61.90%
500	3.0	0.0	0.99	63.40%
		-1.0	1.64	56.30%
		-2.0	2.40	57.70%
		-3.0	3.24	59.30%
	6.0	0.0	0.49	72.60%
		-1.0	0.65	63.80%
		-2.0	0.88	66.00%
		-3.0	1.12	68.40%
2000	3.0	0.0	0.49	62.80%
		-1.0	0.81	55.30%
		-2.0	1.22	58.90%
		-3.0	1.62	60.80%
	6.0	0.0	0.24	72.60%
		-1.0	0.32	69.70%
		-2.0	0.36	71.30%
		-3.0	0.46	74.80%

Non-Parametric Bootstrap (Matlab pdf)

T	df	μ	CI length	CI Accuracy
100	3.0	0.0	1.74	50.00%
		-1.0	2.92	52.30%
		-2.0	4.84	53.30%
		-3.0	6.18	51.40%
	6.0	0.0	0.91	61.00%
		-1.0	1.24	59.70%
		-2.0	1.61	57.40%
		-3.0	2.01	57.10%
500	3.0	0.0	0.97	60.90%
		-1.0	1.59	60.20%
		-2.0	2.33	59.20%
		-3.0	3.18	61.90%
	6.0	0.0	0.48	71.70%
		-1.0	0.64	65.60%
		-2.0	0.85	69.80%
		-3.0	1.08	69.90%
2000	3.0	0.0	0.48	63.50%
		-1.0	0.80	58.10%
		-2.0	1.21	62.70%
		-3.0	1.61	65.10%
	6.0	0.0	0.24	72.50%
		-1.0	0.31	67.50%
		-2.0	0.42	69.90%
		-3.0	0.53	70.10%

Table 8: Results of Code-Section 5 and 6 for different choices of sample size T, degrees of freedom *df* and non-centrality μ, using Paolella's "d.d.a." method for NCT approximation.

```matlab
function MLE = nctlikmax0(x, initvec, method)

 if method == 1
     %method of prof. Paolella
      tol=1e-5;
     opts=optimset('Disp','none','LargeScale','Off','TolFun',tol,'TolX',tol,'Maxiter',200);
     fun = @(param)(-sum(stdnctpdfln_j(x, param)));
     MLE=fminunc(fun,initvec,opts);
end
 if method == 2
     % pdf method from matlab
      tol=1e-5;
     opts=optimset('Disp','none','LargeScale','Off','TolFun',tol,'TolX',tol,'Maxiter',200);
     fun = @(param)(-sum(nctpdfWrapper(x, param)));
     MLE=fminunc(fun,initvec,opts);
end
end

function y = nctpdfWrapper(x, param)
nu = param(1);
delta = param(2);
loc = param(3)
scale = param(4)
x = x*scale+loc
y = log(nctpdf(x, nu, delta));
end
```

Code-Section 5: MLE function for location-scale NCT

```matlab
% Question 4
loc=0; scale=1; xi = 0.05; T=[100;500;2000]; rep=1000; Breps = 1000; confidence = 0.9;
df = [3;6]; mu = linspace(0,-3,4)'; method =[1;2];
rng(0,'twister');

for h = 1:size(T,1)
    for i = 1:size(df,1)
        for j = 1:size(mu,1)
            for k=1:size(method,1)
                question4(loc,scale,xi,df(i),T(h),rep,Breps,confidence,mu(j),method(k));
            end
        end
    end
end

function question4(loc,scale,xi,df,T,rep,Breps,confidence,mu,method)
fprintf('T= %.0f, mu= %.1f, df=%.0f\n',T,mu,df)
fprintf('method= %.0f:\n\n',method)

% True ES
x = nctrnd_1(mu,df,1e6,1);
q = quantile(x,xi);
ES01 = mean(x(x< q));
trueES = loc+scale*ES01;
fprintf('True ES: %.4f\n',trueES)

[CI,CI_nonpar] = CI_calculation(loc,scale,xi,df,T,rep,Breps,confidence,mu,0,0,0,0,method);

% length of the CI's
CI_length = CI(:,2)-CI(:,1);
CI_nonpar_length = CI_nonpar(:,2)-CI_nonpar(:,1);
fprintf('\nthe parametric approach has an average CI length of %.5f\n',mean(CI_length));
fprintf('the non-parametric approach has an average CI length of %.5f\n\n',mean(CI_nonpar_length));


% is the ES in the CI?
CI_hasTrueEs = ((trueES <= CI(:,2)) & (trueES >= CI(:,1)));
CI_nonpar_hasTrueEs = ((trueES <= CI_nonpar(:,2)) & (trueES >= CI_nonpar(:,1)));
fprintf('the parametric CI has the true ES %.2f%% of the time\n',sum(CI_hasTrueEs)/rep*100);
fprintf('the non-parametric CI has the true ES %.2f%% of the time\n\n',sum(CI_nonpar_hasTrueEs)/rep*100);
end
```

Code-Section 6: Computing the ES for the non-central Student's t distribution, only that we now actually compute the MLE of the location-scale NCT. This is done for a variety of parameter choices and using two methods of MLE.

5 Appendix

```
function   [CI,CI_nonpar] = CI_calculation(loc, scale, xi, df, T,rep,Breps,confidence,mu,a,b,c,d,method)
ESvec=zeros(Breps,1); ESvec_nonpar=zeros(Breps,1);
CI = zeros(rep,2); CI_nonpar = zeros(rep,2);

for i=1:rep
    % ****************** DATA *******************
    % data=loc+scale*trnd(df,T,1);                    % question 1
    % data = nctrnd_1(mu,df,T,1);                     % question 2
    % data = stablernd(a,b,c,d,T,1);                  % question 3
    % data = loc+scale*nctrnd_1(mu,df,T,1);           % question 4
    % ****************** DATA *******************

    % ****************** MLE method *************
    % MLE = tlikmax0(data,[df,loc,scale]);            % question 1,2,3
    % MLE = nctlikmax0(data,[df,mu,loc,scale],method); % question 4
    % ********** MLE param Rest of the Q*************
    % df_hat = MLE(1); loc_hat = MLE(2); scale_hat = MLE(3);
    % B_par_samp = loc_hat+scale_hat .* trnd(df_hat,T,Breps);
    % ****************** MLE param Q4 *************
    % df_hat = MLE(1); loc_hat = MLE(3); scale_hat = MLE(4); mu_hat = MLE(2);
    % B_par_samp = loc_hat+scale_hat .* nctrnd_1(mu_hat,df_hat,T,Breps);

    % non-parametric bootstrap
    % sample with replacement from actual data set
    ind=unidrnd(T,[T Breps]);
    B_nonpar_samp=data(ind);

    % get VaR for each bootstrap
    VaR=quantile(B_par_samp, xi,1);
    VaR_nonpar=quantile(B_nonpar_samp, xi,1);

    % get ES for each bootstrap
    for l=1:Breps
        ESvec(l)=mean(data(data<=VaR(l)));
        ESvec_nonpar(l)=mean(data(data<=VaR_nonpar(l)));
    end

    % get 90% Confidence Interval for ES
    CI(i,:)  = quantile(ESvec,[(1-confidence)/2 1-(1-confidence)/2]);
    CI_nonpar(i,:)  = quantile(ESvec_nonpar,[(1-confidence)/2 1-(1-confidence)/2]);
end
```

Code-Section A.1: Function used throughout each exercerise to calculate bootstraps and confidence intervals based on given parameters. To apply to different questions, one un-comments the appropriate data generating process as well as the appropriate MLE method.

```
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);

function ll=tloglik(param,x)
v=param(1); mu=param(2); c=param(3);
if v<0.05, v=rand; end          %An adhoc way of preventing negative values
if c<0.05, c=rand; end          % which works, but is 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);
```

Code-Section A.2: Functions *tlikmax0* and *tloglik* used for Maximum Likelihood estimation.

```
function r = stablernd(alpha,beta,sigma,mu,m,n,par);
if nargin<2,
    error('Requires at least two input arguments.');
end
if nargin<3, sigma = 1; end
if nargin<4, mu = 0; end
if nargin<5, m = 1; end
if nargin<6, n = 1; end
if nargin<7, par = 0; end

% Initialize r to zero.
r = zeros(m,n);
% Run the Chambers-Mallows-Stuck algorithm
U = pi.*(rand(size(r)) - 0.5);
W = -log(rand(size(r)));
piby2 = pi/2;
if alpha~=1,
    zeta = -beta.*tan(piby2.*alpha);
    xi = atan(-zeta)./alpha;
    S_ab = (1+zeta.^2).^(0.5./alpha);
    r = S_ab.*sin(alpha.*(U+xi))./(cos(U).^(1./alpha)).*(cos(U-alpha.*(U+xi))./W).^((1-alpha)./alpha);
else % alpha==0
    r = ((piby2 + beta.*U).*tan(U) - beta.*log(piby2*W.*cos(U)./(piby2 + beta.*U)))./piby2;
end

% Add scale and location
r = sigma.*r + mu;
if par==0, % Correct for alpha<>1 in the S0 parametrization
    if alpha~=1,
        r = r - sigma*beta*tan(alpha*piby2);
    end
else % Correct for alpha==1 in the S (or S1) parametrization
    if alpha==1,
        r = r + sigma*beta*log(sigma)/piby2;
    end
end
```

Code-Section A.3: stablernd.m Copyright (c) 2010 by Rafal Weron. The function generates random variables according to the input parameters of a stable distribution.

```
mu = -3; df = 1000000000;
test_a = zeros(1000,1);
test_b = zeros(1000,1);
test_c = zeros(1000,1);
for n = 1:1000
a = nctrnd_1(df,mu,T);
b = nctrnd(df,mu,T,1);
c = trnd(df,T,1);
test_a(n) = mean(a); test_b(n) = mean(b); test_c(n) = mean(c);
end
mean(test_a)
mean(test_b)
mean(test_c)
```

Code-Section A.4: function to perform a verification of the *nctrnd_1* from Code-Section 2.

```
function pdfln = stdnctpdfln_j(x, param)
nu = param(1);
gam = param(2);
loc = param(3);
scale = param(4);


if nu<0.01, nu=rand; end
if scale<0.01, scale=rand; end


vn2 = (nu + 1) / 2;
z = x*scale + loc;
rho = z.^2;
pdfln = gammaln( vn2 ) - 1/2*log( pi*nu ) - gammaln( nu / 2 ) - vn2*log1p( rho / nu );
if ( all (gam == 0) ) , return , end
idx = ( pdfln >= -37); % 36.841 = log(1e16)
if( any( idx ) )
    gcg = gam.^2; pdfln = pdfln - 0.5*gcg ; xcg = z .* gam;
    term = 0.5*log(2) + log( xcg ) - 0.5*log(max( realmin , nu+rho ) );
    term( term == -inf ) = log( realmin ) ; term(term == +inf ) = log( realmax );
    maxiter = 1e4 ; k = 0;
    logterms = gammaln( ( nu+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( ( nu+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
end
```

Code-Section A.5: function that approximates the location-scale NCT pdf adapted for the MLE application and used in Code-Section 5.

References

[1] Marc S. Paolella (2018). Fundamental Statistical Inference: A Computational Approach, *New York: John Wiley Sons.*

[2] Marc S. Paolella (2007). Intermediate Probability: A Computational Approach, *New York: John Wiley Sons.*