Universität Zürich[UZH]

Assignment 3, due 15.12.2022

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

In this paper, we take a closer look at the Multivariate Student T Distribution and its proprieties. In particular, we evaluate four different estimation methods of the parameters, namely, degrees of freedom, mean, and variance-covariance matrix. We distinguish between the accuracy and the computation speed of each method: Multivariate Myriad Filter, ECME algorithm, Brute-Force MLE, and MLE Approximation. Further, we augment the estimation tools with weighted likelihoods that are applied to a data-set that is no longer stationary. We find that an increase in sample size decreases the numbers of outliers and facilitates, on average, more precise estimations. Further, the weighted likelihood method provides better estimation results for the cases when we introduce non-stationary data-sets.

Contents

1 Working with the Multivariate t Distribution

1.1 Alternatives to the EM Algorithm for ML estimation

Hasannasab et al. [2021] proposed in their paper "Alternatives to the EM algorithm for ML estimation of location, scatter matrix, and degree of freedom of the Student t distribution" three alternatives to the classical EM algorithm which reliably estimate the degrees of freedom ν, location parameter μ and the scatter matrix Σ of a multivariate t-Distribution. The pseudocode for one of their algorithms for such maximum likelihood estimations is given by:

MMF Algorithm

Input: $x_1, \ldots, x_n \in \mathbb{R}^d, n \geq d+1, w \in \mathring{\Delta}_n$

Initialization: $v_0 = \varepsilon > 0, \mu_0 = \frac{1}{n}\sum_{i=1}^n x_i, \Sigma_0 = \frac{1}{n}\sum_{i=1}^n (x_i - \mu_0)(x_i - \mu_0)^{\mathrm{T}}$

for $r = 0, \ldots$

E-Step: Compute the weights

$$\delta_{i,r} = (x_i - \mu_r)^{\mathrm{T}} \Sigma_r^{-1} (x_i - \mu_r)$$

$$\gamma_{i,r} = \frac{v_r + d}{v_r + \delta_{i,r}}$$

M-Step: Update the parameters

$$\mu_{r+1} = \frac{\sum_{i=1}^n w_i \gamma_{i,r} x_i}{\sum_{i=1}^n w_i \gamma_{i,r}}$$

$$\Sigma_{r+1} = \sum_{i=1}^n \frac{w_i \gamma_{i,r} (x_i - \mu_{r+1})(x_i - \mu_{r+1})^{\mathrm{T}}}{\sum_{i=1}^n w_i \gamma_{i,r}}$$

$$v_{r+1} = \text{ zero of}$$

$$\phi\left(\frac{v}{2}\right) - \phi\left(\frac{v+d}{2}\right) + \sum_{i=1}^n w_i \left(\frac{v_r + d}{v_r + \delta_{i,r+1}} - \log\left(\frac{v_r + d}{v_r + \delta_{i,r+1}}\right) - 1\right)$$

We implement this pseudocode in Code-Section 1.1 for $d = 3$. This code will be used in the subsequent sections and compared against alternative algorithms to execute maximum likelihood estimations.

```
function [mu_est,sigma_est,nu_est]= MMF(x)
%Input
%x = transpose(x);
[d,n]   = size(x);
w = ones(1,n)/n;

% Initialisation
nu0=2;
mu0 =mean(x,2);
sigma0 =((x-mu0)*(x-mu0)')/n;
sigma_r = sigma0;
```

```
mu_r = mu0;
nu_r = nu0;

for r=0:n


    %E-step
    delta_r=sum(((sigma_r^(-1))*(x-mu_r)).*(x-mu_r),1);
    gamma_r=(nu_r +d)./(nu_r+delta_r);

    %M-Step
    mu_r_plus_one=sum(repmat(w.*gamma_r,d,1).*x,2)/sum(w.*gamma_r,2);
    sigma_r_plus_one=((x-mu_r_plus_one).*repmat(w.*gamma_r,d,1))*(x-mu_r_plus_one)'/sum(w.*gamma_r,2);


    A=@(x)psi(x/2)-log(x/2);
    B=@(x)sum((x+d)./(x+delta_r) - log((x+d)./(x+delta_r)) -1)/n;
    der_A=@(x).5*psi(1,x/2)-1./x;
    b_nu_r=B(nu_r);
    f=@(nu)A(nu)-A(nu+d)+b_nu_r;
    der_f=@(nu)der_A(nu)-der_A(nu+d);

    [nu_r_plus_one,~]=newton(nu_r,f,der_f);

nu_r = nu_r_plus_one;
mu_r = mu_r_plus_one;
sigma_r = sigma_r_plus_one;
end

mu_est = mu_r_plus_one';
nu_est = nu_r_plus_one;
sigma_est = [diag(sigma_r_plus_one);sigma_r_plus_one(1,2);sigma_r_plus_one(1,3);sigma_r_plus_one(2,3)]';
end
```

Code-Section 1.1: Implementation of the MMF algorithm.

1.2 Simulation of a 3-variate IID multivariate Student t

Now that we have an algorithm for the maximum likelihood estimation of a 3-variate IID multivariate Student t, we want to build a method to simulate such samples on which we can then perform our estimations. That way, we can verify the accuracy of the method. We choose the parameters of the desired random distribution a priori, using 4 degrees of freedom ν, a mean vector μ of zeros on all three dimensions and a non-diagonal (positive definite) Σ matrix. As stated in the assignment outline, the Σ matrix has ones on the main diagonal, which allows for the interpretation as a correlation matrix. The choice of a zero location vector is due to the quality of the estimation results mostly likely being invariant to this choice.

To simulate the 3 variate iid Student t we went along as follows:

$df \leftarrow 4$
$T \leftarrow 100;$
$dim \leftarrow 3$
$Cov \leftarrow createCOV(dim)$
$x \leftarrow mvtrnd(Cov, df, T)$
return $(x, lower\_half\_of\_Cov)$

The symmetric positive definite covariance matrix was generated using our own, custom procedure "*createCOV*". The code can be found in Code-Section 1.2. On a high level, the procedure is as follows:

1. Create a n x n random matrix
2. Set the elements of the main diagonal to 0
3. Make the upper and lower diagonals the same (the matrix is now symmetric)
4. Set the main diagonal to 1
5. Check if the eigenvalues are positive. If yes $\rightarrow$ Done; If no, $\rightarrow$ repeat above procedure.

We included a safety measure to avoid the code getting stuck in a nonconverging process, essentially just restarting the procedure with new random values in case it takes longer than 0.09 seconds to find a covariance matrix with all positive eigenvalues. This ensures that we get a covariance matrix every time, and that the code remains fast. Code-Section 1.3 then provides $T \times dim$ random variable samples based on the input parameters $\nu = 4$, $\mu = [0, 0, 0]$ and the random non-diagonal Σ matrix.

```
function C = createCOV(dim)
tic;
EV = -ones(dim,1);
% start with an identity matrix of size dim x dim
C = eye(dim,dim);
while ~isempty(EV(EV<0))
    % safety measure to avoid the code getting stuck in a nonconverging process
    % due to bad initial random values. If it takes longer than 0.09 sec to get a fitting Cov matrix,
    % the process is restarted with fresh random values.
    if toc > 0.09
        C = createCOV(dim);
        break
    end
    r = -1 + 2.*rand(dim,dim);
    r = r - diag(diag(r));
    r = (r + r.')/2;
    C = C + r;
    EV = eig(C);
end
```

Code-Section 1.2: Generation of a symmetric positive definite correlation matrix.

```
function [x,C_ret] = D3_MVT(T,dim,df)
% get random correlation matrix
C = createCOV(dim);
x = mvtrnd(C,df,T);
C_ret = [diag(C);C(1,2);C(1,3);C(2,3)]';
```

Code-Section 1.3: Generation of a 3-variate iid. multivariate Student t distribution with $\nu = 4$, $\mu = [0, 0, 0]$ and a random non-diagonal Σ matrix given from Code-Section 1.2.

1.3 Parameters' estimation via the MMF algorithm, Brute-force MLE, ECME, and Approximation

Having a method to simulate three-dimensional multivariate Students t, as well as the MMF algorithm to run a maximum likelihood estimation of the parameters, we now want to compare the MMF algorithm with two alternative methods, namely a "Brute-Force" MLE method, the so-called "ECME" approach and an approximative method. To have reliable and extensive data for the comparison, the estimations are run on T=200 and T=2000 observations, repeated 500 times each, estimating parameters $\mu = [\mu_1, \mu_2, \mu_3]^T$, $\Sigma = [\Sigma_{1,1}, \Sigma_{2,2}, \Sigma_{3,3}, \Sigma_{1,2}, \Sigma_{1,3}, \Sigma_{2,3}]^T$ and degrees of freedom ν.

Before diving into a comparison of all algorithm performances, the quality of estimation of the MMF algorithm from Code-Section 1.1 is assessed separately. For both T=200 and T=2000, relevant parameters of the 3-variate IID multivariate Student t are estimated 500 times. For each of the 500 estimations, the true value of the respective parameter was subtracted from the estimate to get deviations from the "true" input values. This way, the results can be interpreted as the quality of the estimation of a given algorithm, with a perfect estimation returning deviations close to (or exactly) zero. The results are displayed in Figure 1.3 and Figure 1.4, with the code used to generate these box plots listed in Code-Section A.6 in the appendix.

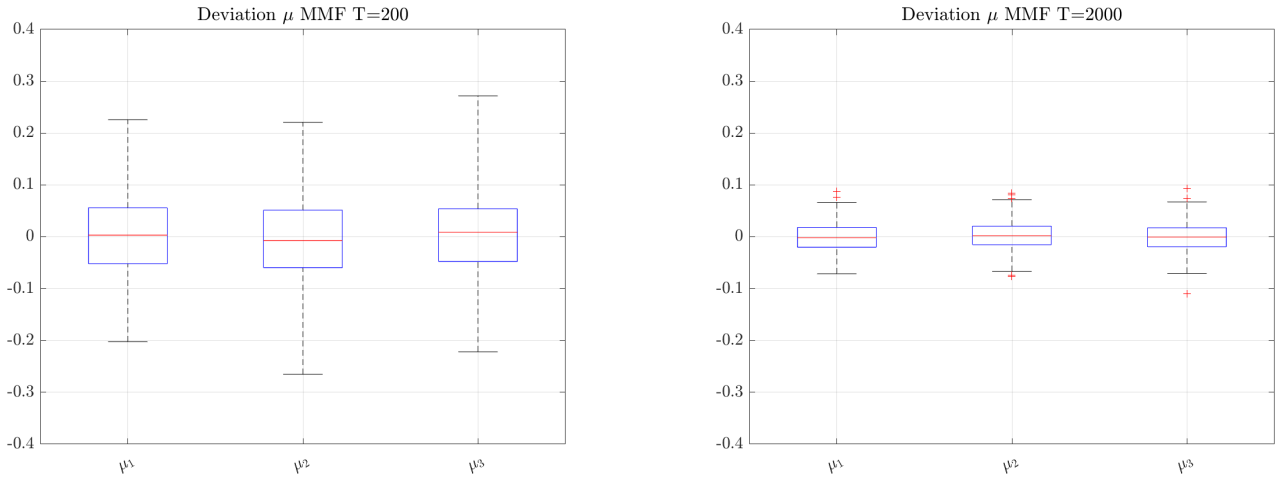

Figure 1.3: MMF Method Performance for μ with $T = 200$ and $T = 2000$.

As expected, the first and third quartiles are wider in the boxplots with T = 200 versus the boxplots with T = 2000. The medians of all plots are close to 0.

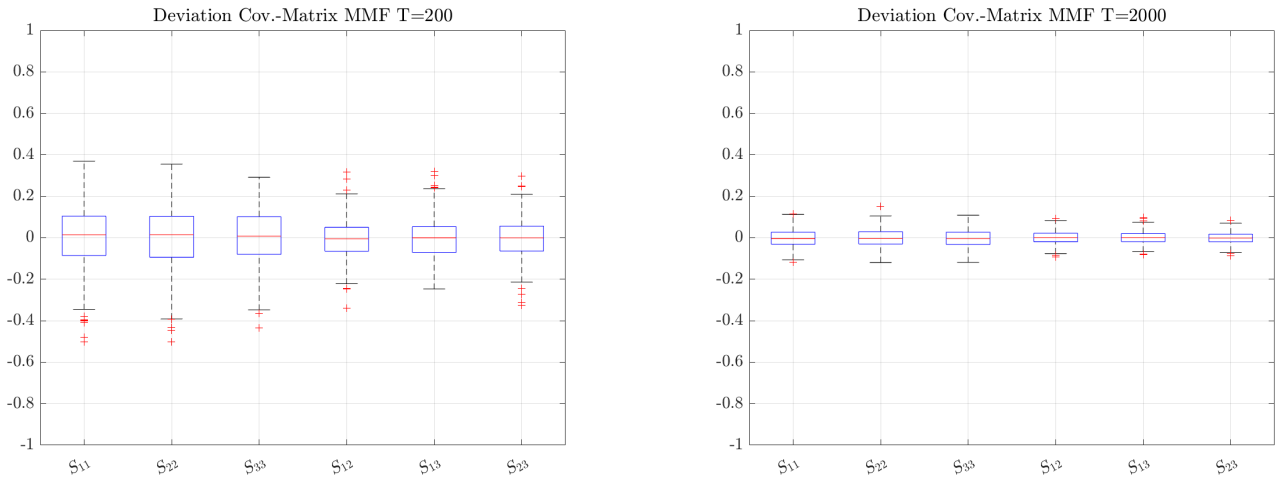

Figure 1.4: MMF Method Performance for Σ with $T = 200$ and $T = 2000$.

The estimates cluster close to 0 with the boxplots for T = 200 being wider than the boxplots for T = 2000, which is sensible since a larger dataset offers higher accuracy per estimation, thus the 500 iterations of the estimation should (and do) return results with less variance in their accuracy. It seems surprising that the boxplots from the main diagonal are generally wider than the elements on the outer diagonals.
The above procedure for evaluation of the goodness of parameter estimation of the "MMF" algorithm is further applied to a "Brute-Force log-likelihood" maximization in Code-Section 1.4 (the function "*MVTestimation_rho*" is computed in Code-Section A.3 in the appendix). Results are displayed in Figure 1.5 and 1.6. When compared to the "MMF" Method, the "Brute-Force" method produces noticeably more outliers. However, the quartiles of the two methods are not significantly different. It can be concluded that, while both estimation procedures produce accurate results, the estimation with the "MMF" algorithm is more robust than the Brute-Force approach.

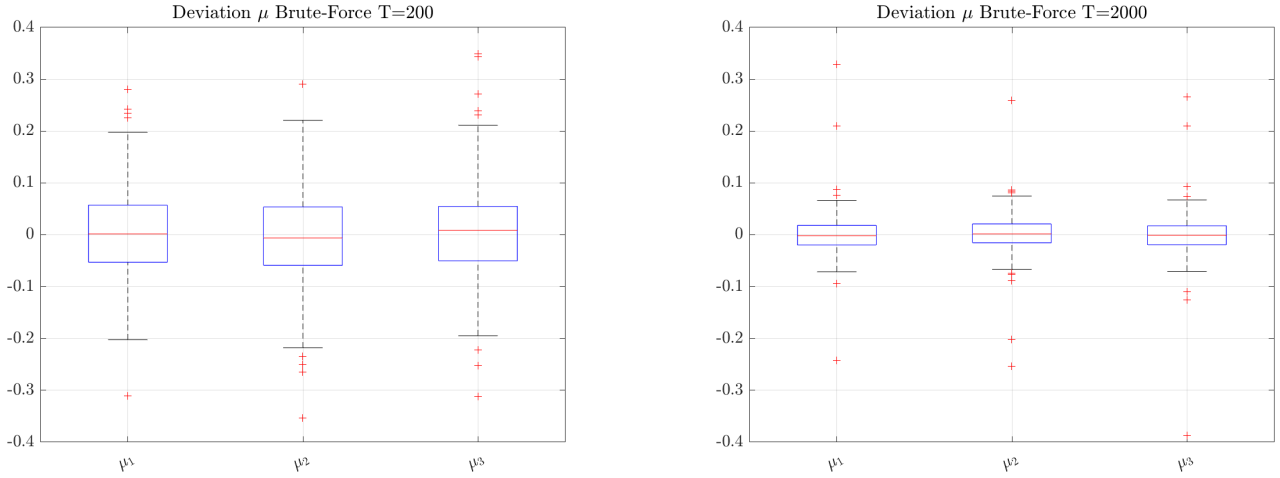

Figure 1.5: "Brute-Force" Log-Likelihood Performance for μ with $T = 200$ and $T = 2000$. Compared to Figure 1.4, there are much more outliers with this method.

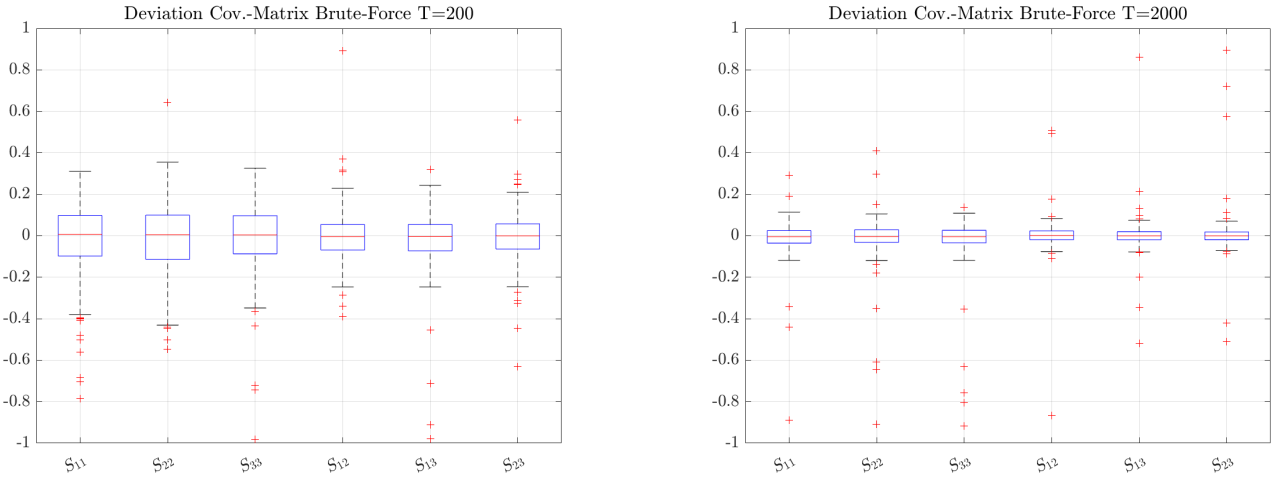

Figure 1.6: "Brute-Force" Log-Likelihood Performance for Σ with $T = 200$ and $T = 2000$.

As a next estimator, the ECME algorithm from Liu and Rubin [1995] is compared with the methods described above. In Liu and Rubin's article, they examine the use of maximum likelihood estimation for the multivariate t distribution using the expectation-maximization algorithm and its extensions, ECM and ECME. The expectation-maximization algorithm works by iteratively maximizing the expected value of a function.
The results in Figure 1.7 is not significantly different from the other two estimation methods. However, what catches the eye is that the covariane matrix estimation in Figure 1.8 shows once again much more outliers than the MMF approach in Figure 1.4. Especially on the estimates for the elements of the covariance matrix which are not on the main diagonal, many outliers are visible.

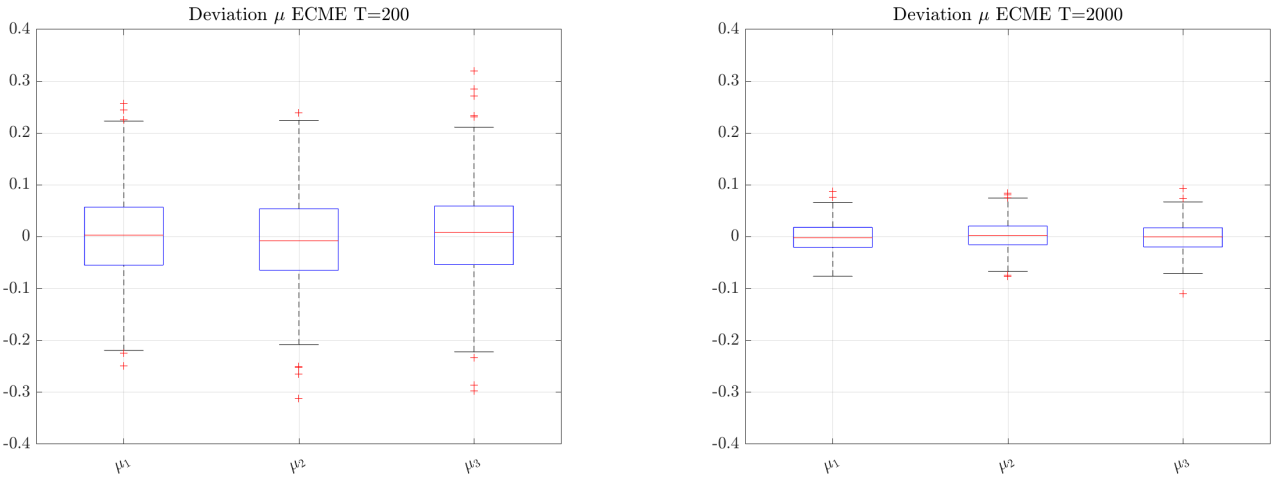

Figure 1.7: ECME Algorithm Performance for μ with $T = 200$ and $T = 2000$.

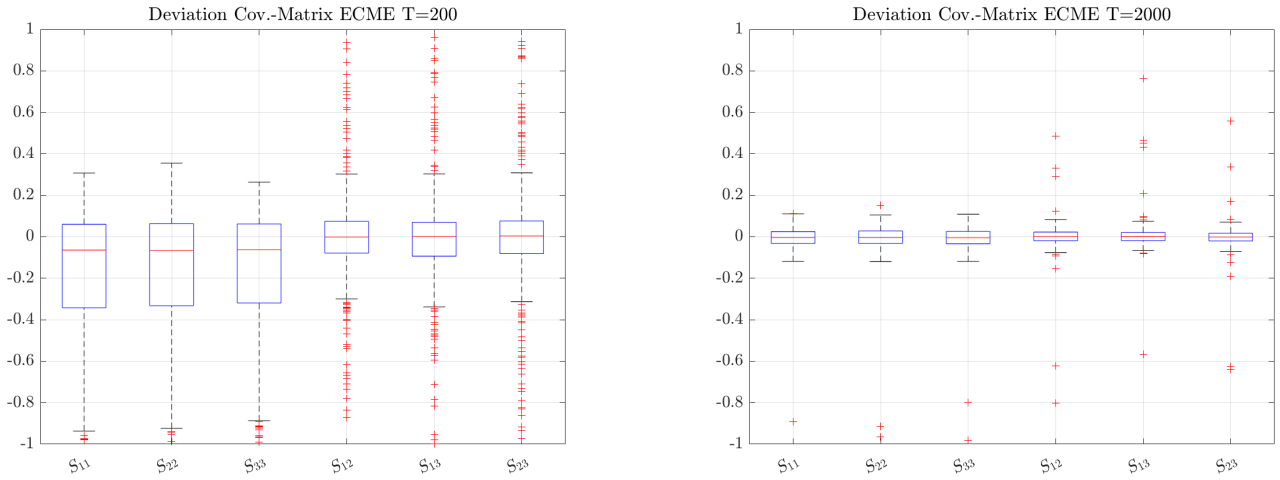

Figure 1.8: ECME Algorithm for Σ with $T = 200$ and $T = 2000$.

Finally, we would like to compare a faster variant of the ECME, the approximation algorithm of Aeschlimna et al. [2010]. Approximation algorithms are a type of algorithm that are used to find approximate solutions to optimization problems. They are often used when the exact solution is too time-consuming or difficult to find. Approximation algorithms are usually faster than exact algorithms, but they are also less accurate. The approximation algorithm of Aeschlimna et al. [2010] is an "Online Algorithm". An Online Algorithm is a type of algorithm that can process data as it arrives, without having to wait for all of the data to be available before starting the computation. These types of algorithms are commonly used in situations where it is not practical to wait for all of the data to be collected before starting the analysis, such as in network routing and monitoring, stock trading, and online advertising. Online Algorithms usually have the trade-off between two competing goals: accuracy and latency (computation time). They are usually designed to operate in real-time and to be able to handle large amounts of data. As such, they are often compared to offline algorithms, which are designed to work with all of the data available before starting the computation. But how does such an algorithm perform compared to an offline algorithm?

Figure 1.9 shows that the online algorithm does a relatively good job in searching for μ. The 1st and 3rd quartiles are only slightly different from those in Figure 1.3, and there are not dramatically many outliers. The situation is different for the estimation of the covariance matrix in Figure 1.10. Here there are many more outliers and the 1st and 3rd quartiles differ considerably from those in Figure 1.4 (MMF Algorithm). Furthermore, in the approximation algorithm we see that the median deviates considerably from 0. Overall these results are not surprising as there is a trade off to be made between computation time and accuracy. The main advantage of the approximation Algorithm becomes evident in Figure 1.12 comparing computation times between the different algorithms. Computation times are very close to 0 seconds, while for example the MMF Algorithm takes more than 100 times longer to compute the estimates.

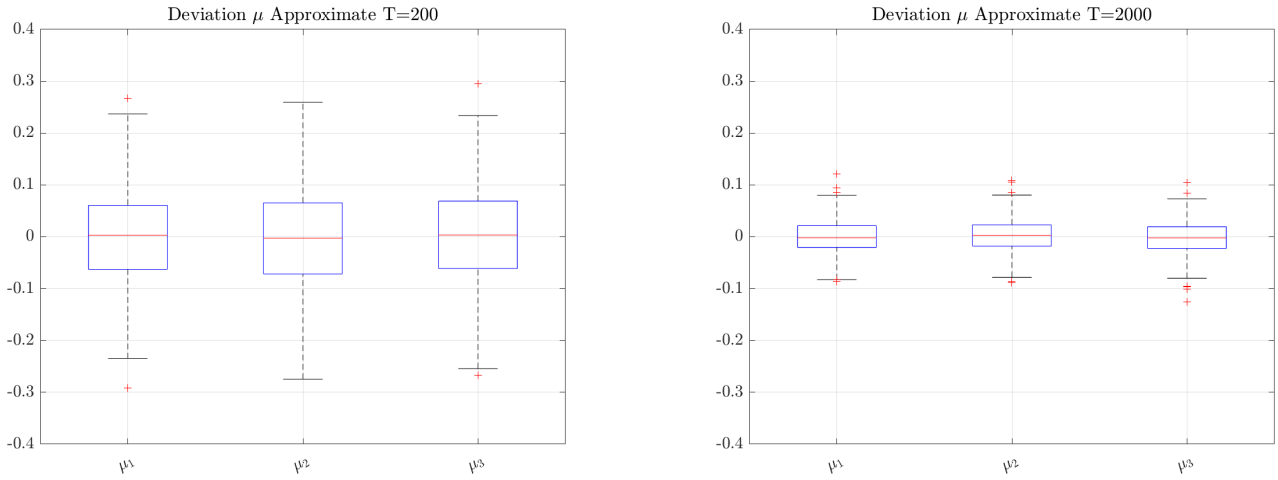

Figure 1.9: Approximation Algorithm Performance for μ with $T = 200$ and $T = 2000$.

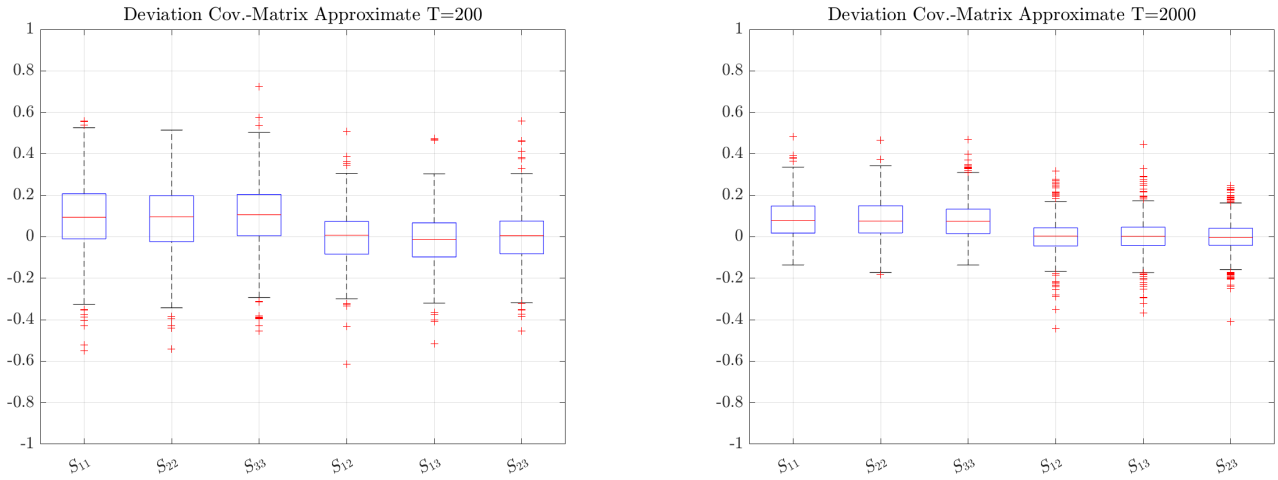

Figure 1.10: Approximation Algorithm for Σ with $T = 200$ and $T = 2000$.

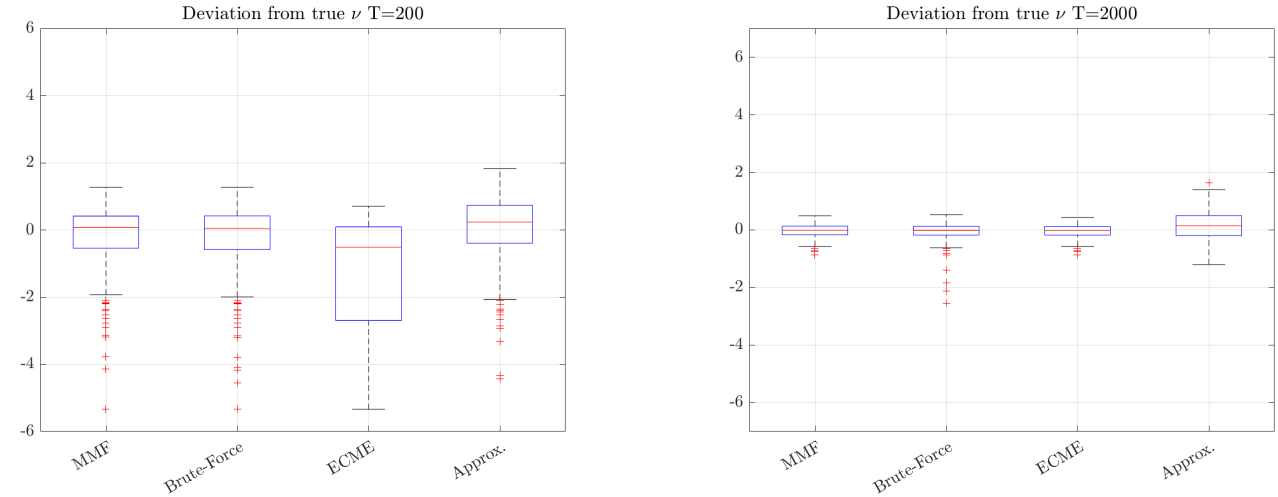

Figure 1.11: Comparison of ν with $T = 200$ and $T = 2000$ for all four algorithms.

Now it remains to discuss the estimates of the four algorithms with respect to the distributions degrees of freedom ν. The comparison is visualized in Figure 1.11. We find it interesting that the algorithms with T = 200 have shown strongly different performances. The MMF and the "Brute-Force" approach seem to be more robust with respect to small sample sizes. At T = 2000, i.e. with a sufficiently large sample size, the algorithms converge more and more. Here, there are hardly any differences between the algorithms. Only the approximation algorithm has its 1st and 3rd quartile somewhat wider than the other estimators. This is not surprising given the online nature of the algorithm.

If we compare the computation times of the four algorithms (Figure 1.12), the enormous duration of the brute-force algorithm is particularly striking. The brute-force algorithm is several thousand times slower than the fastest method, the approximation algorithm. The MMF algorithm seems to be a good compromise between computation time and robustness. Even though it is several 100 times slower than the approximation algorithm, it shows good robustness even with small samples. The ECME algorithm, however, disappointed in our tests. It had a lot of outliers, does not handle small samples well and is slower than the approximation algorithm.

	MMF	*BruteForce*	*ECME*	*Approx.*
Mean	0.7916	16.3753	0.0402	0.0007
Median	0.7806	15.5867	0.0372	0.0004

Table 1: Execution Times (sec.) of the four methods based on 500 repetitions.

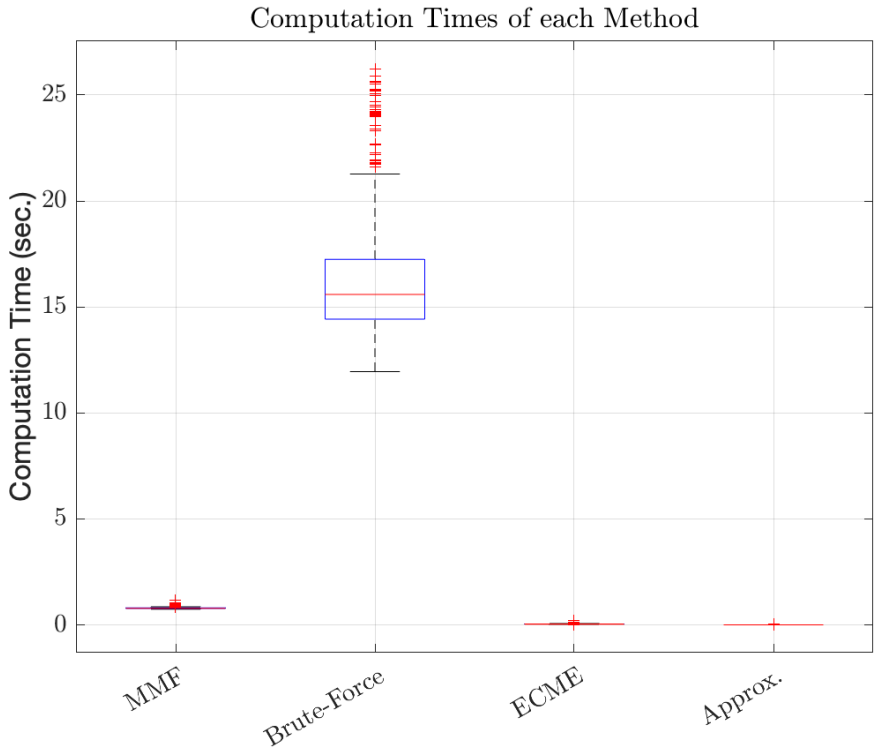

Figure 1.12: Boxplots of the execution Times of the four methods based on 500 repetitions.

```
%% 1.c.d.e
rng('shuffle')
T=[200,2000];%200,2000
Reps = 500;
dim=3;nmbr_par=3;
% Parameters for student t: mu=[0,0,0], Cov-Matrix, df=4
df = 4;
true_param200 = [zeros(Reps,3),zeros(Reps,6), df.*ones(Reps,1)];
true_param2000 = [zeros(Reps,3),zeros(Reps,6), df.*ones(Reps,1)];

mu200 = zeros(Reps,dim);
mu2000 = zeros(Reps,dim);
mu200_bF = zeros(Reps,dim);
mu2000_bF = zeros(Reps,dim);
mu200_ecme = zeros(Reps,dim);
mu2000_ecme = zeros(Reps,dim);
mu200_approx = zeros(Reps,dim);
mu2000_approx = zeros(Reps,dim);

Sm200 = zeros(Reps,6);
Sm2000 = zeros(Reps,6);
Sm200_bF = zeros(Reps,6);
Sm2000_bF = zeros(Reps,6);
Sm200_ecme = zeros(Reps,6);
Sm2000_ecme = zeros(Reps,6);
Sm200_approx = zeros(Reps,6);
Sm2000_approx = zeros(Reps,6);

df200 = zeros(Reps,1);
df2000 = zeros(Reps,1);
df200_bF = zeros(Reps,1);
df2000_bF = zeros(Reps,1);
df200_ecme = zeros(Reps,1);
df2000_ecme = zeros(Reps,1);
df200_approx = zeros(Reps,1);
df2000_approx = zeros(Reps,1);

Times = zeros(Reps*5,1);

%Estimation of the parameters & computation of the deviations from the true parameters - repeated 500
    times
tic;
for i=1:Reps
    fprintf('Round %d\n',i);
    [x200,S200] = D3_MVT(T(1),dim,df);
    [x2000,S2000] = D3_MVT(T(2),dim,df);
    true_param200(i,4:9)=S200;
    true_param2000(i,4:9)=S2000;
    Times(5*(i-1)+1)=toc;
    fprintf('Numbers done\n');

    [mu200(i,:),Sm200(i,:),df200(i)] = MMF(x200');
    [mu2000(i,:),Sm2000(i,:),df2000(i)] = MMF(x2000');
    Times(5*(i-1)+2)=toc;
    fprintf('MMF done\n');

```

```
56      [param200] = MVTestimation(x200);
57       mu200_bF(i,:) = param200([2,3,4]);
58       Sm200_bF(i,:) = param200([5,8,10,6,7,9]);
59       df200_bF(i) = param200(1);
60      [param2000] = MVTestimation(x2000);
61       mu2000_bF(i,:) = param2000([2,3,4]);
62       Sm2000_bF(i,:) = param2000([5,8,10,6,7,9]);
63       df2000_bF(i) = param2000(1);
64
65      Times(5*(i-1)+3)=toc;
66       fprintf('Brute-Force done\n');
67
68       [mu200_ecme(i,:),Sm200_ecme(i,:),df200_ecme(i)] = ECME(x200);
69       [mu2000_ecme(i,:),Sm2000_ecme(i,:),df2000_ecme(i)] = ECME(x2000);
70       Times(5*(i-1)+4)=toc;
71       fprintf('ECME done\n');
72
73       [mu200_approx(i,:),Sm200_approx(i,:),df200_approx(i)] = MLE_approx(x200);
74       [mu2000_approx(i,:),Sm2000_approx(i,:),df2000_approx(i)] = MLE_approx(x2000);
75       Times(5*(i-1)+5)=toc;
76      fprintf('Approx done\n');
77  end
78  k = find(~Times);
79  Times(k)=Times(k-1);
80  Times = [Times(1);Times(2:end)-Times(1:end-1)];
81
82  Times=[Times(1:5:size(Times,1)-4),Times(2:5:size(Times,1)-3),Times(3:5:size(Times,1)-2),...
83      Times(4:5:size(Times,1)-1),Times(5:5:size(Times,1))];
84
85  deviation200 = true_param200 - [mu200,Sm200,df200];
86  deviation2000= true_param2000 - [mu2000,Sm2000,df2000];
87  deviation200_bF = true_param200 - [mu200_bF,Sm200_bF,df200_bF];
88  deviation2000_bF= true_param2000 - [mu2000_bF,Sm2000_bF,df2000_bF];
89  deviation200_ecme = true_param200 - [mu200_ecme,Sm200_ecme,df200_ecme];
90  deviation2000_ecme= true_param2000 - [mu2000_ecme,Sm2000_ecme,df2000_ecme];
91  deviation200_approx = true_param200 - [mu200_approx,Sm200_approx,df200_approx];
92  deviation2000_approx= true_param2000 - [mu2000_approx,Sm2000_approx,df2000_approx];
```

Code-Section 1.4: Computation of the deviations from the true parameters using the four methods.

2 Using Weighted Likelihoods

2.1 Weighted Likelihood MMF

Now that we have established four different algorithms to (more or less accurately) estimate the parameters of a multivariate iid Student t distribution, we want to extend the algorithms capabilities to include hyperbolic weighted likelihood. A hyperbolic weighting scheme given by

$$\varpi = (\varpi_1, \ldots, \varpi_v), \qquad \varpi_t \propto (v - t + 1)^{\rho - 1}, \quad \sum_{t=1}^{v} \varpi_t = 1$$

is implemented by Paolella in Matlab and is found in Code-Section A.3, where it is integrated into the MMF method. Code-Section 2.1 applies this method as a first trial on a dataset with dimensions $T \times 3$.

```
%% 2.b
rng('shuffle')
T=100;df=4;
dim = 3; [x,S] = D3_MVT(T,dim,df); rho = 0.8;
[mu_hat,Sm_hat,df_hat] = MMF_rho(x',rho);
```

Code-Section 2.1: Application of the weighted-likelihood based MMF estimation from Code-Section A.7 on a small dataset using $\rho = 0.8$, $\nu = 4$ and $T = 100$.

Using the adjusted MMF algorithm with the weighted likelihood, we perform a test employing $\nu = 4$ and $\rho = 0.2, 0.3, 0.4, ..., 0.9, 1.0$, with the result represented below in the Figure 2.1. The facilitation of the projection is represented in Code-Section 2.2. We observe that the estimation is the closest at $\rho = 0.6$. As noted earlier, the equally-weighted MLE used to overestimate the degrees of freedom parameter. Therefore, for $\rho < 1$ we expect to have an estimate closer to the true parameter.

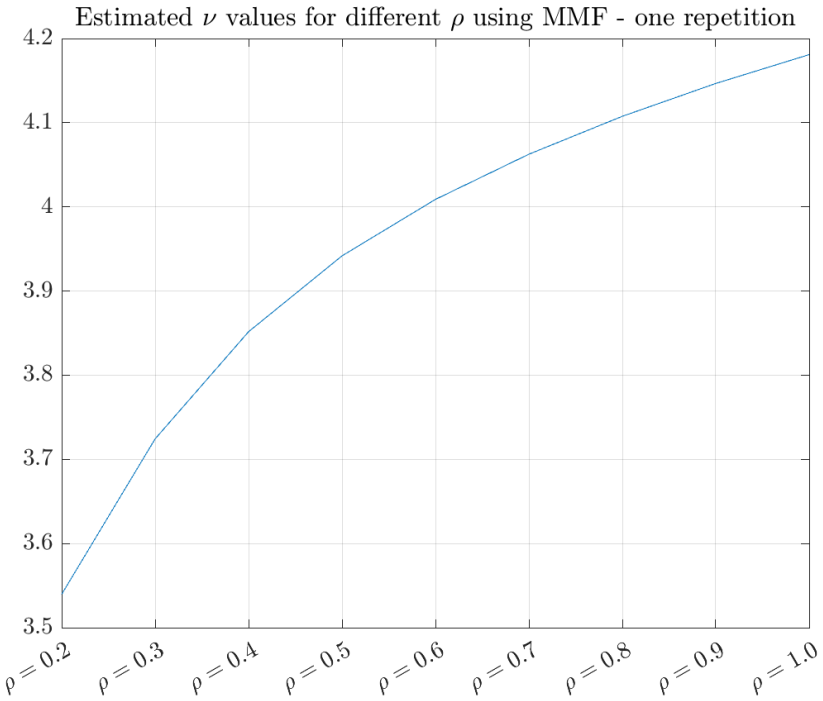

Figure 2.1: ν estimation using the weighted-likelihood estimation *MMF_rho* (Code-Section A.2.2) for different values of ρ - one repetition.

Intensifying the performance analysis of the weighted likelihood estimation method, it is subsequently applied to a more complex dataset of random samples from the three-dimensional Student t distribution, however, with two parameters being manipulated. For one, the dataset builds on varying degrees of freedom ν, namely 200 equal-spaced values between 6 and 3, thus $\nu = 6, 5.985, 5.97, ..., 3.0$. Additionally, the weighting parameter ρ which comes into effect in the weighted-likelihood estimation algorithms is also varying: $\rho = 0.2, 0.3, 0.4, ..., 0.9, 1.0$. The code is implemented in Code-Section A.7, with the resulting estimations of the degrees of freedom for different ρ choices shown in Figure 2.2.

It should be noted that the graph displays the estimate and not (as previously done in Section 1) the deviation from the true parameter values. What can be seen is how with an increase of ρ, the estimation of ν tends to become less fluctuant, with less deviations from its average estimate. The average estimation also slightly moves upward from about $\nu = 3.8$ to around $\nu = 4.3$. Given that the average degrees of freedom of the dataset is given by $mean(\nu_0, ..., \nu_{199}) = \frac{6-3}{2} = 4.5$, the estimations (in this case) become more accurate for higher ρ.

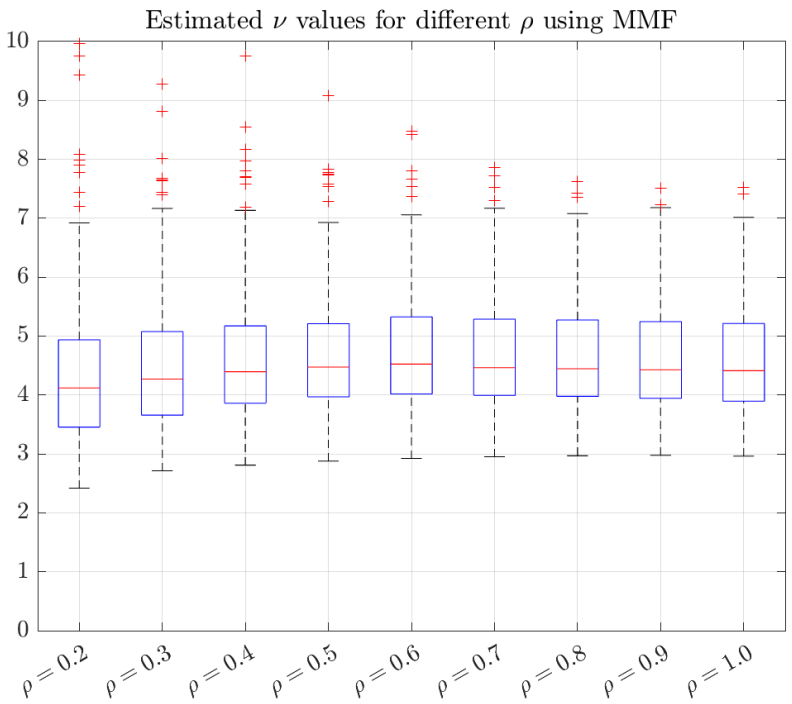

Figure 2.2: ν estimation results using the weighted-likelihood estimation *MMF_rho* (Code-Section A.7) for different values of ρ.

2.2 Weighted Likelihood Brute-Force Method

In a second step, the same analysis and weighted-likelihood approach is applied to the Brute-Force method. The resulting weighted-likelihood method "*MVTestimation_rho*" is shown in Code-Section A.8 and applied in Code-Section 2.3, resulting in Figure 2.3. In addition to the comparison of accuracy between the two methods, Table 2 also shows their computational speed.

	MMF	*BruteForce*
Mean	0.0674	13.5471
Median	0.0637	12.8685

Table 2: Execution Times (sec.) of MMF and Brute Force based on 200 repetitions.

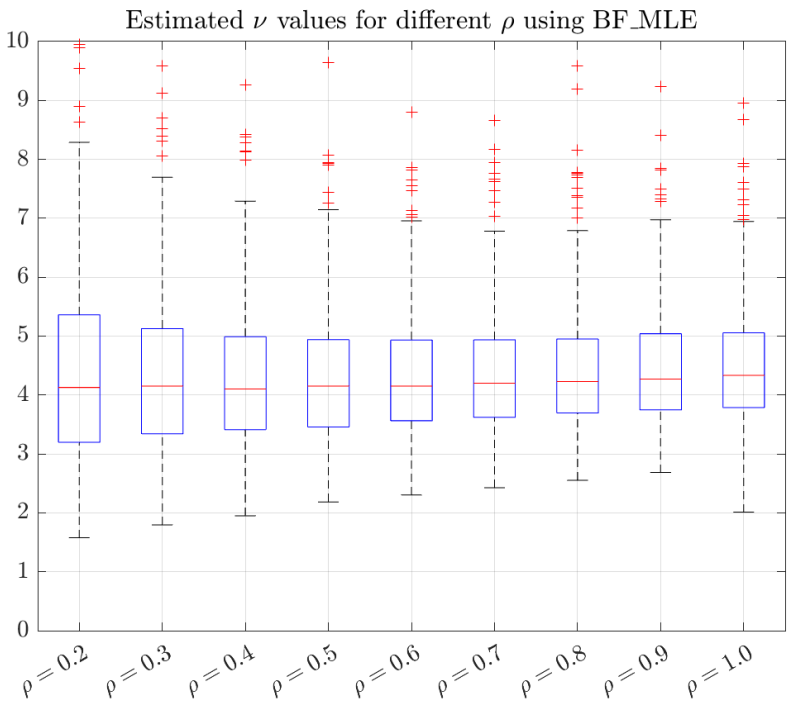

Figure 2.3: ν estimation results using the weighted-likelihood estimation *MVTestimation_rho* (Code-Section A.8) for different values of ρ.

```
%% 2.c & d

T=200;
dim=3;iter=9;
df_test  = 4;
rho = linspace(0.2,1, iter);
S_test=createCOV(dim);
x_test  = mvtrnd(S_test,df_test,T);
df_hat_test  = zeros(1,9);

for r=1:9
        [~,~, df_hat_test(r)] = MMF_rho(x_test',rho(r));
end

figure
plot(rho, df_hat_test)
title('Estimated $\nu$ values for different $\rho$ using MMF - one
    repetition','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\rho=0.2$','$\rho=0.3$','$\rho=0.4$','$\rho=0.5$', ...
    '$\rho=0.6$','$\rho=0.7$','$\rho=0.8$','$\rho=0.9$','$\rho=1.0$'})
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'MMF_rho_test.png')

%% make grid of df values
T=200;
Reps = 200;

dim=3;iter=9;
```

```
dfvec=linspace(6,3,T);
rho = linspace(0.2,1, iter);
df_hat=zeros(Reps,iter);
time_MFF = zeros(Reps,1);

for k=1:Reps
    tic
    fprintf('Round %d\n',k);
    % Create Cov Matrix to use for data
    S=createCOV(dim);
    x=zeros(T,dim);
    % Per value of dfvec, we create 1 instance of x
    for i=1:T
        x(i,:) = mvtrnd(S,dfvec(i),1);
    end
    %fprintf('Numbers done\n');

    for r=1:9
        [~,~, df_hat(k,r)] = MMF_rho(x',rho(r));
    end
    fprintf('MMF_rho done\n');
    time_MFF(k) = toc;
end
%df plot

figure
plot(rho,df_hat(1,1:9))
grid on
title('Estimated $\nu$ values for different $\rho$ using MMF','Interpreter','latex','FontSize',14)
xticklabels({'$\rho=0.2$','$\rho=0.3$','$\rho=0.4$','$\rho=0.5$','$\rho=0.6$', ...
    '$\rho=0.7$','$\rho=0.8$','$\rho=0.9$','$\rho=1.0$'})
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'MMF_rho_d(1).png')

figure
boxplot(df_hat)
title('Estimated $\nu$ values for different $\rho$ using MMF','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\rho=0.2$','$\rho=0.3$','$\rho=0.4$','$\rho=0.5$','$\rho=0.6$',...
    '$\rho=0.7$','$\rho=0.8$','$\rho=0.9$','$\rho=1.0$'})
ylim([0,10]);
xtickangle(30)
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'MMF_rho.png')
```

Code-Section 2.2: Performance of the weighted-likelihood based MMF estimation for different ρ values, applied on a dataset generated using varying degrees of freedom ν (200 different values between 6 and 3).

```
%% 2.e
% make grid of df values
T=200;
Reps = 200;

dim=3;iter=9;
dfvec=linspace(6,3,T);
rho = linspace(0.2,1, iter);
df_hat=zeros(Reps,iter);
time_BF = zeros(Reps,1);
for k=1:Reps
    tic
    fprintf('Round %d\n',k);
    % Create Cov Matrix to use for data
    S=createCOV(dim);
    x=zeros(T,dim);
    % Per value of dfvec, we create 1 instance of x
    for i=1:T
        x(i,:) = mvtrnd(S,dfvec(i),1);
    end
    fprintf('Numbers done\n');

    for r=1:9
        [param] = MVTestimation_rho(x,rho(r));
        df_hat(k,r) = param(1);
    end
    fprintf('MVTestimation_rho done\n');
    time_BF(k) = toc;
end

%df plot
figure
boxplot(df_hat)
title('Estimated $\nu$ values for different $\rho$ using BF\_MLE','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\rho=0.2$','$\rho=0.3$','$\rho=0.4$','$\rho=0.5$','$\rho=0.6$', ...
    '$\rho=0.7$','$\rho=0.8$','$\rho=0.9$','$\rho=1.0$'})
ylim([0,10]);
xtickangle(30)
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'BF_rho.png')

%% 2.f Boxplot time

figure
boxplot([time_MFF,time_BF])
title('Computation time of weighted MMF and Brute-Force MLE','Interpreter','latex','FontSize',14)
grid on
xticklabels({'MFF','Brute-Force'})
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Time_MMF_BF_rho.png')
```

Code-Section 2.3: Performance of the weighted-likelihood "Brute-Force" method for different ρ values, applied on a dataset generated using varying degrees of freedom ν between 6 and 3. The results are compared to the weighted-likelihood MMF method.

3 Appendix

```
% This code belongs to the paper
%
% M. Hasannasab, J. Hertrich, F. Laus, and G. Steidl.
% Alternatives to the EM algorithm for ML-estimation of location, scatter
% matrix and degree of freedom of the student-t distribution.
% Numerical Algorithms, 2020.
% DOI: https://doi.org/10.1007/s11075-020-00959-w
%
% If you use this code, please cite the paper.
%
% This function implements newtons method for determining the zeros in the nu-steps.
%
% INPUTS:
%   start       - initial value
%   f           - function handle of the function
%   der_f       - function handle of the derivative
%   max_steps   - maximum number of steps
%   tol         - stopping criteria. Stop if abs(f(val))<tol.
%
% OUTPUTS:
%   zero        - root of f
%   evals       - number of evaluations of f and its derivative.
%
function [zero,evals]=newton(start,f,der_f,max_steps,tol)
if nargin < 5
    tol=1e-5;
end
if nargin < 4
    max_steps=1000;
end
rep=0;
zero=start;steps=0;
f_val=f(zero);
eps=abs(f_val);
while steps<max_steps && eps>=tol
    newzero=zero-f_val/der_f(zero);
    zero=newzero;
    if zero<0
        zero=10^(-2);
        rep=rep+1;
        if rep>10
            disp('One nu was reprojected on the interval [1e-2,inf[');
            break;
        end
    end
    f_val=f(zero);
    eps=abs(f_val);
    steps=steps+1;
    if mod(steps,100)==0
        disp(['Reached step ' num2str(steps) ' in Newton. Eps: ' num2str(eps) ' with argument '
            num2str(zero) ' and derivative ' num2str(der_f(zero))])
    end
end
if eps>=tol
```

```
    disp([ 'Newton did not converge in ' num2str(max_steps) ' steps. Eps: ' num2str(eps) ' with argument
        ' num2str(zero) ' and derivative ' num2str(der_f(zero))])
end
evals=2*steps+1;
end
```

Code-Section A.1: newton.m function that facilitates the computation of the zeros. The code stems from Hasannasab et al. [2021].

```
function [pout,Vout]=einschrk(pin,bound,Vin)
% [pout,Vout]=einschrk(pin,bound,Vin)
% if Vin specified, then pout is untransformed, otherwise pout is transformed
% M. Paolella, 1997

welche=bound.which;
if all(welche==0) % no bounds!
  pout=pin;
  if nargin==3, Vout=Vin; end
  return
end

lo=bound.lo; hi=bound.hi;
if nargin < 3
  trans=sqrt((hi-pin) ./ (pin-lo));
  pout=(1-welche).* pin + welche .* trans;
  Vout=[];
else
  trans=(hi+lo.*pin.^2) ./ (1+pin.^2);
  pout=(1-welche).* pin + welche .* trans;
  % now adjust the standard errors
  trans=2*pin.*(lo-hi) ./ (1+pin.^2).^2;
  d=(1-welche) + welche .* trans; % either unity or delta method.
  J=diag(d);
  Vout = J * Vin * J;
end
```

Code-Section A.2: Function einschrk used to perform conversion between θ and ϕ. Source: *"Fundamental Statistical Inference A Computational Approach"* Marc. S. Paollela p. 142.

```
function [param,stderr,iters,loglik,Varcov] = MVTestimation(x)

[nobs d] = size(x);

if d~=3, error('not done yet,use EM'), end
if d==3
    %%%%%%% k  mu1 m2 m3 s11 s12 s13 s22 s23 s33
    bound.lo = [0.2 -1 -1 -1 0.01 -90 -90 0.01 -90 0.01];
    bound.hi = [20 1 1 1 90 90 90 90 90  90];
    bound.which = [1 0 0 0 1 1 1 1 1  1];
    initvec  = [2 -0.8 -0.2 -0.3 20 4 2 10 3 20];

end
maxiter=300; tol=1e-7; MaxFunEvals=length(initvec)*maxiter;
opts=optimset('Display','off','Maxiter',maxiter,'TolFun',tol,'TolX',tol,'MaxFunEvals', ...
    MaxFunEvals,'LargeScale','Off');
[pout,fval,~,theoutput,~,hess]= fminunc(@(param)
    MVTloglik(param,x,bound),einschrk(initvec,bound),opts);

V=inv(hess)/nobs; % Don't negate because we work with the negative of the loglik
[param,V]=einschrk(pout,bound,V); % transform and apply delta method to get V
param=param'; Varcov=V; stderr=sqrt(diag(V)); % Approximate standard errors
loglik=-fval*nobs; iters=theoutput.iterations;

function ll=MVTloglik(param,x,bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param),bound,999); end
[nobs d]=size(x); Sig=zeros(d,d); k=param(1); mu=param(2:4);
Sig(1,1)=param(5); Sig(1,2)=param(6); Sig(1,3)=param(7); Sig(2,1)=Sig(1,2); Sig(2,2)=param(8);
    Sig(2,3)=param(9); Sig(3,3)=param(10); Sig(3,1) = Sig(1,3); Sig(3,2) = Sig(2,3);
if min(eig(Sig))<1e-10, ll=1e5;
else
pdf=zeros(nobs,1);
for i=1:nobs, pdf(i) = mvtpdfmine(x(i,:),k,mu,Sig); end
llvec=log(pdf); ll=-mean(llvec); if isinf(ll), ll=1e5; end
end
```

Code-Section A.3: MVT estimation adapted for 3 dimensions

```
function [mu, S, df] = ECME(x)
% FITT(x)
%
% Fit a t-distribution using the ECME algorithm (Lui & Rubin, 1995)
%
% C Liu and D B Rubin, (1995) "ML estimation of the t distribution using EM and
% its extensions, ECM and ECME", Statistica Sinica, 5, pp19-39
% http://www3.stat.sinica.edu.tw/statistica/oldpdf/A5n12.pdf
%
if isvector(x)
    x = x(:);
end
Ntrl = size(x,1);
Nvar = size(x,2);
p = Nvar;

% tolerance for entropy
```

```
tol = 1e-8;
% Seperate tolerances for S and nu convergence
% S_tol = 1e-6;
% nu_tol = 1e-5;
maxiter = 400;

% Initial conditions
mu = mean(x);
S = cov(x);
df = 10;

t = 1;
converged = false;
H = 0;

% history of parameters
% theta = zeros(p+numel(S)+2, maxiter);
% theta(:,1) = [mu S(:)' nu H];

% fsolve options
arg = {
'TolFun', 1e-10
'Jacobian', 'on'
% 'DerivativeCheck', 'on'
'Display', 'off'
% 'Algorithm','levenberg-marquardt'
};
arg = arg';
opt = optimset(arg{:});

p2 = p/2;
while ~converged && (t < maxiter)
    S_old = S;
    nu_old = df;
    H_old = H;
    t = t+1;

    % E step
    % mahalonobis distance with current params
    chS = chol(S)';
    cx = bsxfun(@minus, x, mu)';
    M = chS\cx;
    % M is the normalised innovation and M(:,i)'*M(:,i) gives the Mahalanobis
    % distance for each x(:,i).
    delta = sum(M.*M,1)';
    w = (p + df) ./ (delta + df);

    % CM-1 Step
    % ML estimates of mu, S
    mu = sum(bsxfun(@times, x, w)) ./ sum(w);
    % centered with updated mean
    cx = bsxfun(@minus, x, mu);
    cxw = bsxfun(@times,cx,sqrt(w));
    S = (cxw'*cxw) ./ Ntrl;
    chS = chol(S)';

```

```
    % line search is slow so only do it every other iteration
    if mod(t+1,2)==0
        % E step again
        M = chS\(cx');
        delta = sum(M.*M,1)';
        w = (p + df) ./ (delta + df);

        % CM-2 Step
        optfun = @(v) fitt_optnu(v, delta, p);
        [df, ~, flag] = fsolve(optfun, df, opt);
        if flag < 1
            error('fitt:fsolve did not converge')
        end
    end

    % convergence detection

    % overall difference in parameters
%     theta(:,t) = [mu S(:)' nu H];
%     converged = sum(abs(theta(:,t)-theta(:,t-1))) < tol;

    % don't care about mean for entropy calculation so just check
    % S and nu have converged into a reasonable range
%     converged = (mean(abs(S(:)-S_old(:))) < S_tol) & (abs(nu - nu_old) < nu_tol);

    % use entropy as convergence criteria
    nu2 = df/2;
    nup2 = (df+p)/2;
    H = sum(log(diag(chS))) ...
                + log( ((df*pi).^p2) * beta(p2, nu2) ) - gammaln(p2) ...
                + nup2*(psi(nup2)-psi(nu2));
    converged = abs(H - H_old) < tol;
end
% theta = theta(:,1:t);
S = [diag(S);S(1,2);S(1,3);S(2,3)]';
if ~converged
    error('fitt:ECME algorithm did not converge (maxiter exceeded)')
end
```

Code-Section A.4: "Expectation-conditional maximization either" method based on Liu and Rubin [1995]. The code is provided on Github by Robin Ince.

```
function [mu, S, df]=MLE_approx(x)
% FITT_APPROX(x)
%
% Fit a multivariate t-distribution to data using the approximation method
% from:
% C Aeschlimna, J Park and KA Cak, "A Novel Parameter Estimation Algorithm
% for the Multivariate t-Distribution and its Application to Computer
% Vision" ECCV 2010
% http://link.springer.com/chapter/10.1007%2F978-3-642-15552-9_43

Ntrl = size(x,1);
Nvar = size(x,2);
```

```
13
14 mu = median(x,1);
15 % centered data
16 cx = bsxfun(@minus, x, median(mu));
17
18 zi = log(sum(cx.^2,2));
19 z = sum( zi ) ./ Ntrl;
20
21 b = (sum( (zi-z).^2 ) ./ Ntrl) - psi(1, Nvar/2);
22 df = (1 + sqrt(1+4*b)) / b;
23
24 alpha = exp(z - log(df) + psi(0, df/2) - psi(0, Nvar/2));
25 beta = (2*log2(Nvar))/(df^2 + log2(Nvar));
26
27 S = (cx'*bsxfun(@rdivide,cx, sum(cx.^2,2).^(beta/2)))./Ntrl;
28 S = (alpha*Nvar/trace(S)) * S;
29 S=[diag(S);S(1,2);S(1,3);S(2,3)]';
```

Code-Section A.5: Approximative Method to execute mean likelihood estimations based on Aeschlimna et al. [2010]. The code is provided on Github by Robin Ince.

```
%% Plot data
% mu plot
figure
boxplot(deviation200(:,1:3),'whisker',inf)
title('Deviation $\mu$ MMF T=200','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_MMF_200.png')

figure
boxplot(deviation2000(:,1:3))
title('Deviation $\mu$ MMF T=2000','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_MMF_2000.png')

figure
boxplot(deviation200_bF(:,1:3))
title('Deviation $\mu$ Brute-Force T=200','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_BF_200.png')

figure
boxplot(deviation2000_bF(:,1:3))
title('Deviation $\mu$ Brute-Force T=2000','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_BF_2000.png')

figure
boxplot(deviation200_ecme(:,1:3))
title('Deviation $\mu$ ECME T=200','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_ECME_200.png')

figure
boxplot(deviation2000_ecme(:,1:3))
title('Deviation $\mu$ ECME T=2000','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_ECME_2000.png')

```

```
figure
boxplot(deviation200_approx(:,1:3))
 title ('Deviation $\mu$ Approximate T=200','Interpreter','latex','FontSize',14)
grid on
 xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
 fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_Approx_200.png')

 figure
boxplot(deviation2000_approx(:,1:3))
 title ('Deviation $\mu$ Approximate T=2000','Interpreter','latex','FontSize',14)
grid on
 xticklabels({'$\mu_1$','$\mu_2$','$\mu_3$'})
ylim([-0.4,0.4]); xtickangle(30);
 fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_mu_Approx_2000.png')

% S plot
 figure
boxplot(deviation200(:,4:9))
 title ('Deviation Cov.-Matrix MMF T=200','Interpreter','latex','FontSize',14)
grid on
 xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([-1,1]);
xtickangle(30);
 fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_MMF_200.png')

 figure
boxplot(deviation2000(:,4:9))
 title ('Deviation Cov.-Matrix MMF T=2000','Interpreter','latex','FontSize',14)
grid on
 xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([-1,1]);
xtickangle(30);
 fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_MMF_2000.png')

 figure
boxplot(deviation200_bF(:,4:9))
 title ('Deviation Cov.-Matrix Brute-Force T=200','Interpreter','latex','FontSize',14)
grid on
 xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([-1,1]);
xtickangle(30);
 fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_BF_200.png')

 figure
boxplot(deviation2000_bF(:,4:9))
 title ('Deviation Cov.-Matrix Brute-Force T=2000','Interpreter','latex','FontSize',14)
grid on
 xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([-1,1]);
xtickangle(30);
```

```
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_BF_2000.png')

figure
boxplot(deviation200_ecme(:,4:9))
title('Deviation Cov.−Matrix ECME T=200','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([−1,1]);
xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_ECME_200.png')

figure
boxplot(deviation2000_ecme(:,4:9))
title('Deviation Cov.−Matrix ECME T=2000','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([−1,1]);
xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_ECME_2000.png')

figure
boxplot(deviation200_approx(:,4:9))
title('Deviation Cov.−Matrix Approximate T=200','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([−1,1]);
xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_Approx_200.png')

figure
boxplot(deviation2000_approx(:,4:9))
title('Deviation Cov.−Matrix Approximate T=2000','Interpreter','latex','FontSize',14)
grid on
xticklabels({'$S_{11}$','$S_{22}$','$S_{33}$','$S_{12}$','$S_{13}$','$S_{23}$'})
ylim([−1,1]);
xtickangle(30);
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_Sigma_Approx_2000.png')

%df plot
figure
boxplot([deviation200(:,10),deviation200_bF(:,10),deviation200_ecme(:,10),deviation200_approx(:,10)])
title('Deviation from true $\nu$ T=200','Interpreter','latex','FontSize',14)
grid on
xticklabels({'MMF','Brute−Force','ECME','Approx.'})
ylim([−6,6]);
 xtickangle(30)
fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_df_200.png')

figure
boxplot([deviation2000(:,10),deviation2000_bF(:,10),deviation2000_ecme(:,10),deviation2000_approx(:,10)])
```

```
 title ('Deviation from true $\nu$ T=2000','Interpreter','latex','FontSize',14)
grid on
 xticklabels({'MMF','Brute-Force','ECME','Approx.'})
ylim([-7,7]);
  xtickangle(30)
 fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Dev_df_2000.png')


% Times
 figure
boxplot(Times(:,2:end))
 title ('Computation Times of each Method','Interpreter','latex','FontSize',14)
grid on
ylabel('Computation Time (sec.)')
 xticklabels({'MMF','Brute-Force','ECME','Approx.'})
xtickangle(30)
 fig = gcf; ax = gca(fig); ax.FontSize = 14; ax.TickLabelInterpreter = 'latex';
saveas(gcf,'Times.png')
```

Code-Section A.6: Generation of the plots for Section 1.3.

```
function [mu_est,sigma_est,nu_est]= MMF_rho(x,rho)
%Input

[d,n]  = size(x);

% According to Listing 13.1 in Time-Series Book
tvec=(1:n); omega=(n-tvec+1).^(rho-1); w=n*omega'/sum(omega);
w=w';
% Initialisation

nu0=2;
mu0 =mean(x,2);
sigma0 =((x-mu0)*(x-mu0)')/n;
sigma_r = sigma0;
mu_r = mu0;
nu_r = nu0;

for r=0:n


    %E-step
    delta_r=sum(((sigma_r^(-1))*(x-mu_r)).*(x-mu_r),1);
    gamma_r=(nu_r +d)./(nu_r+delta_r);

    %M-Step
    mu_r_plus_one=sum(repmat(w.*gamma_r,d,1).*x,2)/sum(w.*gamma_r,2);
    sigma_r_plus_one=((x-mu_r_plus_one).*repmat(w.*gamma_r,d,1))*(x-mu_r_plus_one)'/sum(w.*gamma_r,2);


    A=@(x)psi(x/2)-log(x/2);
    B=@(x)sum((x+d)./(x+delta_r) - log((x+d)./(x+delta_r)) -1)/n;
    der_A=@(x).5*psi(1,x/2)-1./x;
    b_nu_r=B(nu_r);
    f=@(nu)A(nu)-A(nu+d)+b_nu_r;
    der_f=@(nu)der_A(nu)-der_A(nu+d);

    [nu_r_plus_one,~]=newton(nu_r,f,der_f);

nu_r = nu_r_plus_one;
mu_r = mu_r_plus_one;
sigma_r = sigma_r_plus_one;
end

mu_est = mu_r_plus_one;
nu_est = nu_r_plus_one;
sigma_est = [diag(sigma_r_plus_one);sigma_r_plus_one(1,2);sigma_r_plus_one(1,3);sigma_r_plus_one(2,3)];
end
```

Code-Section A.7: MMF Maximum Likelihood estimation using a hyperbolically weighted likelihood, based on code from Paolella [2019].

```
function [param,stderr,iters,loglik,Varcov] = MVTestimation_rho(x,rho)

[nobs d] = size(x);

if d~=3, error('not done yet,use EM'), end
if d==3
    %%%%%%% k  mu1 m2 m3 s11 s12 s13 s22 s23 s33
    bound.lo = [0.2 -1 -1 -1 0.01 -90 -90 0.01 -90 0.01];
    bound.hi = [20 1 1 1 90 90 90 90 90  90];
    bound.which = [1 0 0 0 1 1 1 1 1  1];
    initvec  = [2 -0.8 -0.2 -0.3 20 4 2 10 3 20];

end
maxiter=300; tol=1e-7; MaxFunEvals=length(initvec)*maxiter;
opts=optimset('Display','off','Maxiter',maxiter,'TolFun',tol,'TolX',tol,'MaxFunEvals'...
    ,MaxFunEvals,'LargeScale','Off');
[pout,fval,~,theoutput,~,hess]= fminunc(@(param)
    MVTloglik(param,x,bound,rho),einschrk(initvec,bound),opts);

V=inv(hess)/nobs; % Don't negate because we work with the negative of the loglik
[param,V]=einschrk(pout,bound,V); % transform and apply delta method to get V
param=param'; Varcov=V; stderr=sqrt(diag(V)); % Approximate standard errors
loglik=-fval*nobs; iters=theoutput.iterations;

function ll=MVTloglik(param,x,bound,rho)
[nobs d] = size(x);
tvec=(1:nobs); omega=(nobs-tvec+1).^(rho-1); w=nobs*omega'/sum(omega);

if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param),bound,999); end
[nobs d]=size(x); Sig=zeros(d,d); k=param(1); mu=param(2:4);
Sig(1,1)=param(5); Sig(1,2)=param(6); Sig(1,3)=param(7); Sig(2,1)=Sig(1,2); Sig(2,2)=param(8);
    Sig(2,3)=param(9); Sig(3,3)=param(10); Sig(3,1) = Sig(1,3); Sig(3,2) = Sig(2,3);
if min(eig(Sig))<1e-10, ll=1e5;
else
pdf=zeros(nobs,1);
for i=1:nobs, pdf(i) = mvtpdfmine(x(i,:),k,mu,Sig); end
llvec=log(pdf); ll=-mean(w.*llvec); if isinf(ll), ll=1e5; end
end
```

Code-Section A.8: "Brute-Force" maximum likelihood estimation using weighted-likelihood ρ.

References

C Aeschlimna, J Park, and KA Cak. *A Novel Parameter Estimation Algorithm for the Multivariate t-Distribution and its Application to Computer Vision*, 2010. http://link.springer.com/chapter/10.1007%2F978-3-642-15552-9$_4$3, *ECCV*.

M. Hasannasab, J. Hertrich, F. Laus, and et al. *Alternatives to the EM algorithm for ML estimation of location, scatter matrix, and degree of freedom of the Student t distribution.*, 2021. https://doi.org/10.1007/s11075-020-00959-w, Numer Algor 87, 77–118.

C Liu and D B Rubin. *ML estimation of the t distribution using EM and its extensions, ECM and ECME*, 1995. http://www3.stat.sinica.edu.tw/statistica/oldpdf/A5n12.pdf, Statistica Sinica, 5, pp19-39.

Marc S. Paolella. *Intermediate Probability: A Computational Approach*, 2019. John Wiley & Sons Ltd.