University of Zurich[UZH]

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

Assignment 3, due December 15th, 2022

Prof. Dr. Marc Paolella

Ilaria Avallone, 18-435-800

Lorena Tassone, 18-700-237

Naina Srivastava, 21-738-117

Question 1: Working with the Multivariate t Distribution

In this first exercise we consider maximum likelihood estimations of the degree of freedom parameter ν, the location parameter μ and the scatter matrix Σ of the multivariate Student t distribution. We compare four different estimation algorithms in its estimated value distribution using boxplots, and we also compare their estimation time.
The four estimation algorithms are the following:

1. EM algorithm with Multivariate Myriad Filter (MMF)
2. Maximum Likelihood Estimation using Log Likelihood Maximization (MLE)
3. ECME algorithm for Multivariate t Distribution
4. Aeschliman et al. for Multivariate t Distribution parameter approximation

First we implement the MMF algorithm by taking advantage of the pseudocode on page 91 of the paper "Alternatives to the EM algorithm for ML estimation of location, scatter matrix, and degree of freedom of the Student t distribution"[1] by Hasannasab et al., to then create the function *function_MMFAlgorithm* shown in Listing 2.
It takes the multivariate T distribution sample, initial guess for degrees of freedom, the weights given to each data point in the sample and the repetitions as input and provides the estimated values of degrees of freedom ν, the mean μ, and the correlation matrix Σ. All together we have 10 parameter estimations: 1 for ν, three for μ (since we have a 3-variate IID multivariate Student t), and six for Σ with 3 off-diagonal entries and 3 diagonal entries.

The function is then applied on a simulated 3-variate IID multivariate Student t simulation with a true mean of 0 and degrees of freedom of 4. The correlation matrix is invented by us, having diagonal values of 1 and random off-diagonal values. We create a symmetric 3x3 matrix with random numbers. Since it is important that we get a positive definite Σ matrix, we implement this in a loop that ensures all eigenvalues to be positive.
For each sample size, we simulate the distribution and re-sample and estimate 500 times to have 500 parameter estimations for each parameter. We later create and

[1]https://doi.org/10.1007/s11075-020-00959-w

show boxplots for these 500 estimated parameters.

In each rep we estimate the parameters using the four methods described above at the same time, such that we use the same 500 repetitions for all the methods, to have a better comparability.
So, first we estimate via the *function_MMFAlgorithm* and then save all the 10 estimated parameters in pre-defined vectors. Again, this can be found in Listing 2.
We do then the same using the MLE routine for log likelihood maximization. For this purpose, we adjusted the functions in Program Listings 12.1 and 12.2 provided in the book "Linear Models and Time-Series Analysis" by Marc S. Paolella, to be able to input a 3-dimensional data set. The updated function *MVTestimation_3d* can be found in Listing 3.
In a third step, we apply the ECME algorithm from the paper "ML estimation of the t distribution using EM and its extensions, ECM and ECME" written by C. Liu and D. B. Rubin, (1995)[2]. This function can be found in Listing 4 and 5.
And in a final step, we use the approximation method from C. Aeschliman, J. Park and K.A. Cak, form their paper "A Novel Parameter Estimation Algorithm for the Multivariate t-Distribution and its Application to Computer Vision"[3]. The according function can be found in Listing 6.

Once we have obtained all our estimations according to the different methods, to compare them we report wonderful boxplots to assess the quality of the methods implied. Importantly, before this, we subtract the true value of the parameter from our estimates to make sure that the boxplots illustrate the deviation from the truth (the only exception is the ν parameter, for which we do not subtract the true value). We further calculate the average estimation time of each of the four methods.

The complete MATLAB code for Question 1 can be found in Listing 1.

Listing 1: Parameter estimation using MMF, MLE, ECME and the Aeschliman approximation method.

```
% Input parameters
dim = 3;               % matrix dimensionality
diag = 1;              % diagonal values
M = zeros(3, 3);       % initializing correlation matrix
```

[2] 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]

[3] C. Aeschliman, J. Park and K.A. Cak, "A Novel Parameter Estimation Algorithm for the Multivariate t-Distribution and its Application to Computer Vision" [ECCV 2010]

```

%%% Create a covariance matrix with positive eigenvalues and diagonal values of 1
while (sum(eig(M)<=0) ~= 0)
    t = triu(bsxfun(@min,diag,-diag.').*rand(dim),1); % The upper trianglar random values
    M = diag(diag)+t+t.'; % Putting them together in a symmetric matrix
end
disp('Sigma matrix: '); disp(M);
disp('Eigenvalues of sigma matrix: '); disp(eig(M));


%%% Simulate 3d MVT and estimate parameters
% Input parameters
T_samples = [200,2000];    % [200, 2000]
rep = 500;                 % # repititions: 500
loc = 0;
scale = 1;
df_true = 4;               % true df value
reps = 500;                % # repititions for MMF: 500
initial_df = 0.1;

% Parameters for MMF
step_algorithm = 'MMF';
anz_steps = 500;
stop = 1;
abs_criteria = 1;
regularize = 1;
save_obj = 1;

% Initializing vectors
mus_mmf = zeros(rep, length(T_samples)*dim);
nus_mmf = zeros(rep, length(T_samples));
sigmas_mmf = zeros(rep, length(T_samples)*6);
mus_mmf_adj = zeros(rep, length(T_samples)*dim);
nus_mmf_adj = zeros(rep, length(T_samples));
sigmas_mmf_adj = zeros(rep, length(T_samples)*6);

mus_mle = zeros(rep, length(T_samples)*dim);
nus_mle = zeros(rep, length(T_samples));
sigmas_mle = zeros(rep, length(T_samples)*6);
mus_mle_adj = zeros(rep, length(T_samples)*dim);
nus_mle_adj = zeros(rep, length(T_samples));
sigmas_mle_adj = zeros(rep, length(T_samples)*6);

mus_ecme = zeros(rep, length(T_samples)*dim);
```

```
nus_ecme = zeros(rep, length(T_samples));
sigmas_ecme = zeros(rep, length(T_samples)*6);
mus_ecme_adj = zeros(rep, length(T_samples)*dim);
nus_ecme_adj = zeros(rep, length(T_samples));
sigmas_ecme_adj = zeros(rep, length(T_samples)*6);

mus_approx = zeros(rep, length(T_samples)*dim);
nus_approx = zeros(rep, length(T_samples));
sigmas_approx = zeros(rep, length(T_samples)*6);
mus_approx_adj = zeros(rep, length(T_samples)*dim);
nus_approx_adj = zeros(rep, length(T_samples));
sigmas_approx_adj = zeros(rep, length(T_samples)*6);

time_mmf = zeros(rep,length(T_samples));
time_mle = zeros(rep,length(T_samples));
time_ecme = zeros(rep,length(T_samples));
time_approx = zeros(rep,length(T_samples));

ts = 0; tm = 0;

% True values
mu_true = [0 0 0];
df_true = 4;
sigma_true = M;


for t = 1:length(T_samples)

    % Simulate multi variate t distribution (MVT)
    data=loc+scale*mvtrnd(M,df_true,T_samples(t));

    for r = 1:rep

        % Sampling from MVT
        ind = unidrnd(T_samples(t),[T_samples(t),1]);
        mvt_sample=data(ind,1:3)';

        %%% Estimating parameters through MMF
        % and creating adjusted estimates by true values
        weights = 1/T_samples(t) * ones(T_samples(t), 1);
        tic
        [nu, nu_vec, mu, sigma] = function_MMFAlgorithm(mvt_sample,
            initial_df, weights, reps);
        time_mmf(r,t) = toc;

        mus_mmf(r,1+tm) = mu(1);    mus_mmf_adj(r,1+tm) = mu(1) -
```

```
        mu_true(1);
        mus_mmf(r,2+tm) = mu(2);    mus_mmf_adj(r,2+tm) = mu(2) - mu_true(2);
        mus_mmf(r,3+tm) = mu(3);    mus_mmf_adj(r,3+tm) = mu(3) - mu_true(3);

        nus_mmf(r,t) = nu;    nus_mmf_adj(r,t) = nu - df_true;

        sigmas_mmf(r,1+ts) = sigma(1,1);  sigmas_mmf_adj(r,1+ts) = sigma(1,1) - M(1,1);
        sigmas_mmf(r,2+ts) = sigma(2,1);  sigmas_mmf_adj(r,2+ts) = sigma(2,1) - M(2,1);
        sigmas_mmf(r,3+ts) = sigma(3,1);  sigmas_mmf_adj(r,3+ts) = sigma(3,1) - M(3,1);
        sigmas_mmf(r,4+ts) = sigma(2,2);  sigmas_mmf_adj(r,4+ts) = sigma(2,2) - M(2,2);
        sigmas_mmf(r,5+ts) = sigma(3,2);  sigmas_mmf_adj(r,5+ts) = sigma(3,2) - M(3,2);
        sigmas_mmf(r,6+ts) = sigma(3,3);  sigmas_mmf_adj(r,6+ts) = sigma(3,3) - M(3,3);

        %%% Estimating parameters through MLE
        % and creating adjusted estimates by true values
        tic
        [param,stderr,iters,loglik,Varcov] = MVTestimation_3d(mvt_sample', weights);
        time_mle(r,t) = toc;

        % param output order:
        % k, mu1, mu2, mu3, Sigma_11, Sigma_12, Sigma_13,Sigma_22, Sigma_23, Sigma_33)
        mus_mle(r,1+tm) = param(2);    mus_mle_adj(r,1+tm) = param(2) - mu_true(1);
        mus_mle(r,2+tm) = param(3);    mus_mle_adj(r,2+tm) = param(3) - mu_true(2);
        mus_mle(r,3+tm) = param(4);    mus_mle_adj(r,3+tm) = param(4) - mu_true(3);

        nus_mle(r,t) = param(1);    nus_mle_adj(r,t) = param(1) - df_true;

        sigmas_mle(r,1+ts) = param(5);  sigmas_mle_adj(r,1+ts) = param(5) - M(1,1);
        sigmas_mle(r,2+ts) = param(6);  sigmas_mle_adj(r,2+ts) = param(6) - M(2,1);
        sigmas_mle(r,3+ts) = param(7);  sigmas_mle_adj(r,3+ts) = param
```

```
        (7) - M(3,1);
    sigmas_mle(r,4+ts) = param(8);  sigmas_mle_adj(r,4+ts) = param
        (8) - M(2,2);
    sigmas_mle(r,5+ts) = param(9);  sigmas_mle_adj(r,5+ts) = param
        (9) - M(3,2);
    sigmas_mle(r,6+ts) = param(10);  sigmas_mle_adj(r,6+ts) = param
        (10) - M(3,3);

    %%% Estimating parameters through ECME
    % and creating adjusted estimates by true values
    tic
    [mu_ECME, S_ECME, nu_ECME] = fitt(mvt_sample');
    time_ecme(r,t) = toc;

    mus_ecme(r,1+tm) = mu_ECME(1);    mus_ecme_adj(r,1+tm) = mu_ECME
        (1) - mu_true(1);
    mus_ecme(r,2+tm) = mu_ECME(2);    mus_ecme_adj(r,2+tm) = mu_ECME
        (2) - mu_true(2);
    mus_ecme(r,3+tm) = mu_ECME(3);    mus_ecme_adj(r,3+tm) = mu_ECME
        (3) - mu_true(3);

    nus_ecme(r,t) = nu_ECME;    nus_ecme_adj(r,t) = nu_ECME -
        df_true;

    sigmas_ecme(r,1+ts) = S_ECME(1,1);  sigmas_ecme_adj(r,1+ts) =
        S_ECME(1,1) - M(1,1);
    sigmas_ecme(r,2+ts) = S_ECME(2,1);  sigmas_ecme_adj(r,2+ts) =
        S_ECME(2,1) - M(2,1);
    sigmas_ecme(r,3+ts) = S_ECME(3,1);  sigmas_ecme_adj(r,3+ts) =
        S_ECME(3,1) - M(3,1);
    sigmas_ecme(r,4+ts) = S_ECME(2,2);  sigmas_ecme_adj(r,4+ts) =
        S_ECME(2,2) - M(2,2);
    sigmas_ecme(r,5+ts) = S_ECME(3,2);  sigmas_ecme_adj(r,5+ts) =
        S_ECME(3,2) - M(3,2);
    sigmas_ecme(r,6+ts) = S_ECME(3,3);  sigmas_ecme_adj(r,6+ts) =
        S_ECME(3,3) - M(3,3);

    %%% Estimating parameters through the approximation method
    % and creating adjusted estimates by true values
    tic
    [mu_approx, S_approx, nu_approx] = fitt_approx(mvt_sample');
    time_approx(r,t) = toc;

    mus_approx(r,1+tm) = mu_approx(1);    mus_approx_adj(r,1+tm) =
        mu_approx(1) - mu_true(1);
    mus_approx(r,2+tm) = mu_approx(2);    mus_approx_adj(r,2+tm) =
```

```
                mu_approx(2) - mu_true(2);
            mus_approx(r,3+tm) = mu_approx(3);   mus_approx_adj(r,3+tm) = mu_approx(3) - mu_true(3);

            nus_approx(r,t) = nu_approx;   nus_approx_adj(r,t) = nu_approx - df_true;

            sigmas_approx(r,1+ts) = S_approx(1,1);  sigmas_approx_adj(r,1+ts) = S_approx(1,1) - M(1,1);
            sigmas_approx(r,2+ts) = S_approx(2,1);  sigmas_approx_adj(r,2+ts) = S_approx(2,1) - M(2,1);
            sigmas_approx(r,3+ts) = S_approx(3,1);  sigmas_approx_adj(r,3+ts) = S_approx(3,1) - M(3,1);
            sigmas_approx(r,4+ts) = S_approx(2,2);  sigmas_approx_adj(r,4+ts) = S_approx(2,2) - M(2,2);
            sigmas_approx(r,5+ts) = S_approx(3,2);  sigmas_approx_adj(r,5+ts) = S_approx(3,2) - M(3,2);
            sigmas_approx(r,6+ts) = S_approx(3,3);  sigmas_approx_adj(r,6+ts) = S_approx(3,3) - M(3,3);
    end
    ts = ts+6; tm = tm+dim;
end


%% Printing values
disp('***MMF***')
disp(['Sample size: ', num2str(T_samples(1))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_mmf(:,1))/60), rem(mean(time_mmf(:,1)),60));
disp(['Mean nu: ', num2str(mean(nus_mmf(:,1)))])
disp(['Mean mu: ', num2str(mean(mus_mmf(:,1:3)))])
disp(['Sample size: ', num2str(T_samples(2))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_mmf(:,2))/60), rem(mean(time_mmf(:,2)),60));
disp(['Mean nu: ', num2str(mean(nus_mmf(:,2)))])
disp(['Mean mu: ', num2str(mean(mus_mmf(:,4:6)))])

disp('***MLE***')
disp(['Sample size: ', num2str(T_samples(1))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_mle(:,1))/60), rem(mean(time_mle(:,1)),60));
disp(['Mean nu: ', num2str(mean(nus_mle(:,1)))])
disp(['Mean mu: ', num2str(mean(mus_mle(:,1:3)))])
disp(['Sample size: ', num2str(T_samples(2))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_mle(:,2))/60), rem(mean(time_mle(:,2)),60));
```

```
disp(['Mean nu: ', num2str(mean(nus_mle(:,2)))])
disp(['Mean mu: ', num2str(mean(mus_mle(:,4:6)))])

disp('**ECME***')
disp(['Sample size: ', num2str(T_samples(1))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_ecme(:,1))/60), rem(mean(time_ecme(:,1)),60));
disp(['Mean nu: ', num2str(mean(nus_ecme(:,1)))])
disp(['Mean mu: ', num2str(mean(mus_ecme(:,1:3)))])
disp(['Sample size: ', num2str(T_samples(2))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_ecme(:,2))/60), rem(mean(time_ecme(:,2)),60));
disp(['Mean nu: ', num2str(mean(nus_ecme(:,2)))])
disp(['Mean mu: ', num2str(mean(mus_ecme(:,4:6)))])

disp('**APPROX method***')
disp(['Sample size: ', num2str(T_samples(1))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_approx(:,1))/60), rem(mean(time_approx(:,1)),60));
disp(['Mean nu: ', num2str(mean(nus_approx(:,1)))])
disp(['Mean mu: ', num2str(mean(mus_approx(:,1:3)))])
disp(['Sample size: ', num2str(T_samples(2))])
fprintf('Estimation time: %d min %f sec\n', floor(mean(time_approx(:,2))/60), rem(mean(time_approx(:,2)),60));
disp(['Mean nu: ', num2str(mean(nus_approx(:,2)))])
disp(['Mean mu: ', num2str(mean(mus_approx(:,4:6)))])


%%% Plotting
% Grouped plot - mu
hAxes.TickLabelInterpreter = 'latex';

figure, lab = {'\mu_1', '\mu_2', '\mu_3'};
boxplotGroup({mus_mmf_adj(:,1:3),mus_mle_adj(:,1:3),mus_ecme_adj(:,1:3),mus_approx_adj(:,1:3)}, 'PrimaryLabels',{'MMF','MLE','ECME','Aeschlimann'},'SecondaryLabels',lab,'GroupLabelType','Vertical','Whisker',1.5); set(gca,'fontsize',12)
title(['\mu values (sample size: ',num2str(T_samples(1)),')'])
name_mu_1 = ['Assignment3_ex1_mu_',T_samples(1),'.png'];
saveas(gcf,name_mu_1)

figure, lab = {'\mu_1', '\mu_2', '\mu_3'};
boxplotGroup({mus_mmf_adj(:,4:6),mus_mle_adj(:,4:6),mus_ecme_adj(:,4:6),mus_approx_adj(:,4:6)}, 'PrimaryLabels',{'MMF','MLE','ECME','Aeschlimann'},'SecondaryLabels',lab,'GroupLabelType','Vertical','Whisker',1.5); set(gca,'fontsize',12)
```

```
title(['\mu values (sample size: ',num2str(T_samples(2)),')'])
name_mu_2 = ['Assignment3_ex1_mu_',T_samples(2),'.png'];
saveas(gcf,name_mu_2)


% Grouped plot - sigma
figure, lab = {'\sigma_{11}', '\sigma_{21}','\sigma_{31}', '\sigma_{22}','\sigma_{32}', '\sigma_{33}'};
boxplotGroup({sigmas_mmf_adj(:,1:6),sigmas_mle_adj(:,1:6),sigmas_ecme_adj(:,1:6),sigmas_approx_adj(:,1:6)},'PrimaryLabels',{'MMF','MLE','ECME','Aeschlimann'},'SecondaryLabels',lab,'GroupLabelType','Vertical','Whisker',1.5);
title(['\Sigma values (sample size: ',num2str(T_samples(1)),')'])
name_sigma_1 = ['Assignment3_ex1_sigma_',T_samples(1),'.png'];
saveas(gcf,name_sigma_1)

figure, lab = {'\sigma_{11}', '\sigma_{21}','\sigma_{31}', '\sigma_{22}','\sigma_{32}', '\sigma_{33}'};
boxplotGroup({sigmas_mmf_adj(:,7:12),sigmas_mle_adj(:,7:12),sigmas_ecme_adj(:,7:12),sigmas_approx_adj(:,7:12)},'PrimaryLabels',{'MMF','MLE','ECME','Aeschlimann'},'SecondaryLabels',lab,'GroupLabelType','Vertical','Whisker',1.5);
title(['\Sigma values (sample size: ',num2str(T_samples(2)),')'])
name_sigma_2 = ['Assignment3_ex1_sigma_',T_samples(2),'.png'];
saveas(gcf,name_sigma_2)


% Grouped plot - nu (not adjusted values)
figure
boxplot([nus_mmf(:,1),nus_mle(:,1),nus_ecme(:,1),nus_approx(:,1)],'Labels',{'MMF','MLE','ECME','Aeschlimann'},'Whisker',1.5)
set(gca,'fontsize',12)
title(['\nu values (sample size: ',num2str(T_samples(1)),')'])
name_nu_1 = ['Assignment3_ex1_nu_',T_samples(1),'.png'];
saveas(gcf,name_nu_1)

figure
boxplot([nus_mmf(:,2),nus_mle(:,2),nus_ecme(:,2),nus_approx(:,2)],'Labels',{'MMF','MLE','ECME','Aeschlimann'},'Whisker',1.5)
set(gca,'fontsize',12)
title(['\nu values (sample size: ',num2str(T_samples(2)),')'])
name_nu_2 = ['Assignment3_ex1_nu_',T_samples(2),'.png'];
saveas(gcf,name_nu_2)
```

Listing 2: Implementation of the MMF Algorithm deriving from the paper "Alternatives to the EM algorithm for ML estimation of location, scatter matrix, and degree of freedom of the Student t distribution" by Hasannasab et al.

```
function [final_nu, nu_vec, mu, sigma] = function_MMFAlgorithm(x, initial_df, weights, reps)
%%% input parameters
% x                    matrix of random samples of a multivarite t dist
% initial_df           starting value for the degrees of freedom
% weights              weights
% reps                 number of repetitions

%%% output
% final_nu             degrees of freedom of the latest iteration
% nu_vec               estimate of the degrees of freedom
% mu                   mean vector of the latest iteration
% sigma                variance-covariance matrix of the latest iteration


% check input
if initial_df <= 0
    error ("df_initial must be strictly larger 0")
end

if size(x, 1) > size(x, 2)
    error("number of samples must be less than dim + 1 (where dim: sample size)")
end

if sum(weights <= 0) > 0
    error("all weights must be strictly positive")
end

if sum(weights) - 1 > 1e-10
    error("the weights must sum to one")
end

if sum(weights) - 1 < -1e-10
    error("the weights must sum to one")
end


% initialize variables
d = size(x , 1);           % dimension
nu = initial_df;           % nu
```

```
nu_vec = zeros(reps, 1);
mu = sum(x, 2)/size(x, 2);  % mu

% initialize sigma matrix
sigma0 = 0;
for i = 1:length(x)
    sigma0 = sigma0 + (x(:, i) - mu)*(x(:, i) - mu)';
end
sigma = sigma0/length(x);


for r = 1:reps
    %%%% E-step: compute weights
    % initialize vectors for delta and gamma
    delta = zeros(size(x, 2), 1);
    gamma = zeros(size(x, 2), 1);
    % fill vectors with values for the current rep loop
    for i = 1:size(x, 2)
       delta(i) = (x(:, i) - mu)' / sigma * (x(:, i) - mu);
       gamma(i) = (nu + d)/(nu + delta(i,1));
    end

    %%%% M-step: update the parameters
    % initialize (set to zero) variable to save denominator for updating mu and sigma
    denom = 0;
    for i = 1:size(x, 2)
        denom = denom + weights(i)*gamma(i);
    end

    %%% mu
    % initialize (set to zero) variable to save the numerator
    mu_nom = 0;
    % calculate the nominator for updating mu
    for i = 1:size(x, 2)
        mu_nom = mu_nom + weights(i) * gamma(i) * x(:, i);
    end
    % update mu value
    mu = mu_nom / denom;

    %%% sigma
    % initialize (set to zero) variable to save the numerator
    sigma_num = 0;
    % calculate the nominator for updating sigma
    for i = 1:length(x)
        sigma_num = sigma_num + ( weights(i) * gamma(i) * (x(:,i) - mu)
```

```
                * (x(:,i) - mu)' );
    end
    % update sigma value
    sigma = sigma_num / denom;

    %%% nu
    % initialize (set to zero) variable to save the sum part of the
        updating step for nu
    nu_sum = 0;
    for i = 1:length(x)
        nu_sum = nu_sum + weights(i) * ( (nu + d) / (nu + delta(i)) -
            log( (nu + d)/(nu + delta(i)) ) - 1 );
    end
    nu = fzero(@(x) phi_func(x/2) - phi_func((x+d) / 2) + nu_sum , [1e
        -100, 1e100]);
    nu_vec(r) = nu;

end
final_nu = nu;

end

function [phi] = phi_func(x)
    phi = psi(x) - log(x);
end
```

Listing 3: MLE Log Likelihood Maximization, adjusted for 3-variate MVT

```
function [param,stderr,iters,loglik,Varcov] = MVTestimation_3d(x,
    weights)
% param: (k, mu1, mu2, mu3, Sigma_11, Sigma_12, Sigma_13, Sigma_22,
    Sigma_23, Sigma_33)
[nobs d]=size(x); if d~=3, error('not done yet, use EM'), end
if d==3
    %%%%%%%%        k       mu1      mu2     mu3     s11      s12   s13
        s22    s23      s33
    bound.lo= [     0.2     -1       -1      -1      0.01     -90   0.01
        -90    0.01     -90];
    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    -0.8    20       2     10
        20      2        10];
end

```

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

function ll=MVTloglik(param,x,bound,weights)
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); % Assume d=2
Sig(1,1)=param(5); Sig(1,2)=param(6); Sig(1,3)=param(7); Sig(2,2)=param(8);Sig(2,3)=param(9); Sig(3,3)=param(10); Sig(2,1)=Sig(1,2); 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=-sum(llvec.*weights)/sum(weights); if isinf(ll), ll=1e5; end

end


function y = mvtpdfmine(x,df,mu,Sigma)
% x is a d X 1 vector. Unlike Matlab's version, cannot pass a matrix.
% Matlab's routine accepts correlation (not dispersion) matrix.
% So, just need to do the usual scale transfrom. For example:
%   x = [0.2 0.3]'; C = [1 .4; .4 1]; df = 2;
%   scalevec = [1 2]'; xx = x./scalevec; mvtpdf(xx,C,df)/prod(scalevec)
% Same as:
%   Sigma = diag(scalevec) * C * diag(scalevec); mvtpdfmine(x,df,[],Sigma)
d=length(x);
if nargin<3, mu = []; end, if isempty(mu), mu = zeros(d,1); end
if nargin<4, Sigma = eye(d); end
```

```
x = reshape(x,d,1); mu = reshape(mu,d,1); term = (x-mu)' * inv(Sigma) *
    (x-mu);
logN=-((df+d)/2)*log(1+term/df); logD=0.5*log(det(Sigma))+(d/2)*log(df*
    pi);
y = exp(gammaln((df+d)/2) - gammaln(df/2) + logN -logD);
end
end


function[pout, Vout] = einschrk(pin, bound, Vin)

lo = bound.lo; hi = bound.hi; welche = bound.which;

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; % eitherunity or delta method .
    J = diag(d); Vout = J*Vin*J;
end
```

Listing 4: Fits a t distribution by using the ECME algorithm (Lui & Rubin, 1995)

```
function [mu, S, nu] = fitt(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);
nu = 0.1;

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 = nu;
    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 + nu) ./ (delta + nu);

    % 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 + nu) ./ (delta + nu);

        % CM-2 Step
        optfun = @(v) fitt_optnu(v, delta, p);
        [nu, ~, flag] = fsolve(optfun, nu, 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 = nu/2;
    nup2 = (nu+p)/2;
    H = sum(log(diag(chS))) ...
               + log( ((nu*pi).^p2) * beta(p2, nu2) ) - gammaln(p2) ...
               + nup2*(psi(nup2)-psi(nu2));
    converged = abs(H - H_old) < tol;
end
% theta = theta(:,1:t);
```

```

if ~converged
    error('fitt:ECME algorithm did not converge (maxiter exceeded)')
end
```

Listing 5: Solves for ML nu estimates

```
function [f, df] = fitt_optnu(nu, delta, p)
% function to solve for ML nu estimate
nu2 = nu/2;
pnu2 = (p+nu)/2;
w = (p + nu) ./ (delta + nu);

f = -psi(nu2) + log(nu2) + (sum(log(w)-w)/length(delta)) + 1 ...
     + psi(pnu2) - log(pnu2);

% jacobian

if nargout > 1
    dp = delta - p;
    dnu = delta + nu;
    sumterm = (1/(length(delta)*(p+nu))) * sum( ((delta-p)./(delta+nu))
        .^2 );
    df = -0.5*psi(1,nu2) + (1/nu) + sumterm + 0.5*psi(1,pnu2) - 1/(p+nu
        );
end
```

Listing 6: Function allows to fit a multivariate t distribution to data using the approximation method (Aeschlimna, Park and Cak, 2010)

```
function [c, S, nu] = fitt_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);

c = median(x);
```

```
% centered data
cx = bsxfun(@minus, x, median(c));

zi = log(sum(cx.^2,2));
z = sum( zi ) ./ Ntrl;

b = (sum( (zi-z).^2 ) ./ Ntrl) - psi(1, Nvar/2);
nu = (1 + sqrt(1+4*b)) / b;

alpha = exp(z - log(nu) + psi(0, nu/2) - psi(0, Nvar/2));
beta = (2*log2(Nvar))/(nu^2 + log2(Nvar));

S = (cx'*bsxfun(@rdivide,cx, sum(cx.^2,2).^(beta/2)))./Ntrl;
S = (alpha*Nvar/trace(S)) * S;
```

As we will see in the results in the following sub-chapters, the MLE, ECME and MMF methods result in very similar and close estimates, while the approximation method does not seem to be that accurate.

Estimated μ values

Table 1: Average estimated μ for the 500 repetitions of the four methods.

	MMF	MLE	ECME	Aeschliman et al.
Sample size			200	
μ_1	0.1170	0.1169	0.1170	0.1308
μ_2	0.0858	0.0858	0.0858	0.1493
μ_3	0.1011	0.1011	0.1011	0.1386
Sample size			2000	
μ_1	0.0302	0.0302	0.0302	0.0263
μ_2	0.0383	0.0383	0.0383	0.0151
μ_3	0.0233	0.0233	0.0233	0.0477

We can see from Table 1 and Figures 1 and 2 that MMF, MLE, and ECME give similar, if not the same, accurate values for μ, whereas the approximation method proposed by Aeschliman et al. gives slightly different values and also wider distribution of values and is hence less accurate. However, the approximation method is definitely better in terms of parameter estimation time as we will see in Table 4. Furthermore, the estimation seems to be better with a larger sample size of T=2000 than T=200. Intuitively this makes sense, since as the sample size increases, the estimated μ gets closer to the true value which is 0, for each of the three dimensions.

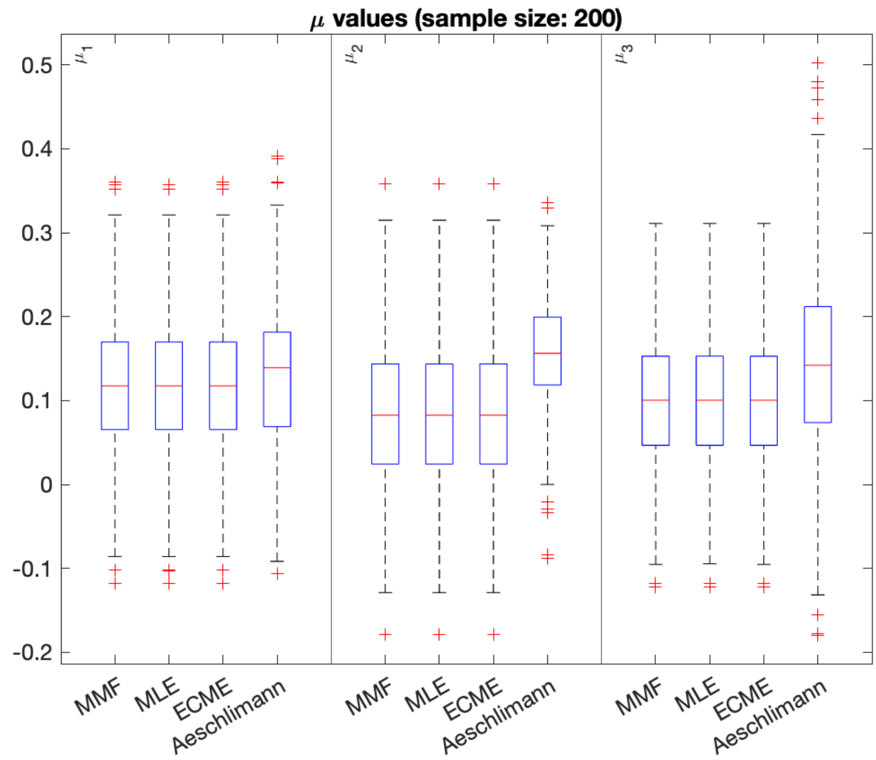

Figure 1: Estimated μ values for T=200 using the different methods

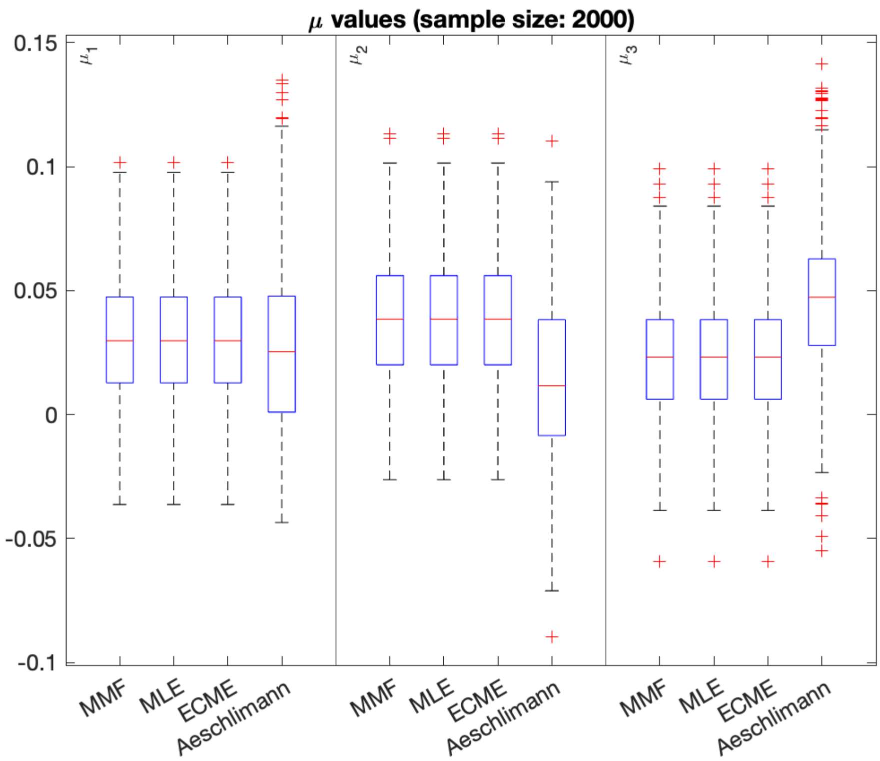

Figure 2: Estimated μ values for T=2000 using the different methods

Estimated ν values

Table 2: Average estimated ν for the 500 repetitions of the four methods.

	MMF	MLE	ECME	Aeschliman et al.
Sample size			200	
ν	4.6715	4.6728	4.6715	3.663
Sample size			2000	
ν	3.9389	3.939	3.9389	3.3989

From Table 2 above and the Figures 3 and 4 shown below for the estimated ν, we can again see that the MMF, MLE, and the ECME methods give similar ν estimates for both T=200 and T=2000, whereas, the approximation method gives lower estimates. Here also we notice that the estimation is better for T=2000 than for T=200. Again, this is due to the fact that a bigger sample size gives estimations closer to the true value which in this case is 4.

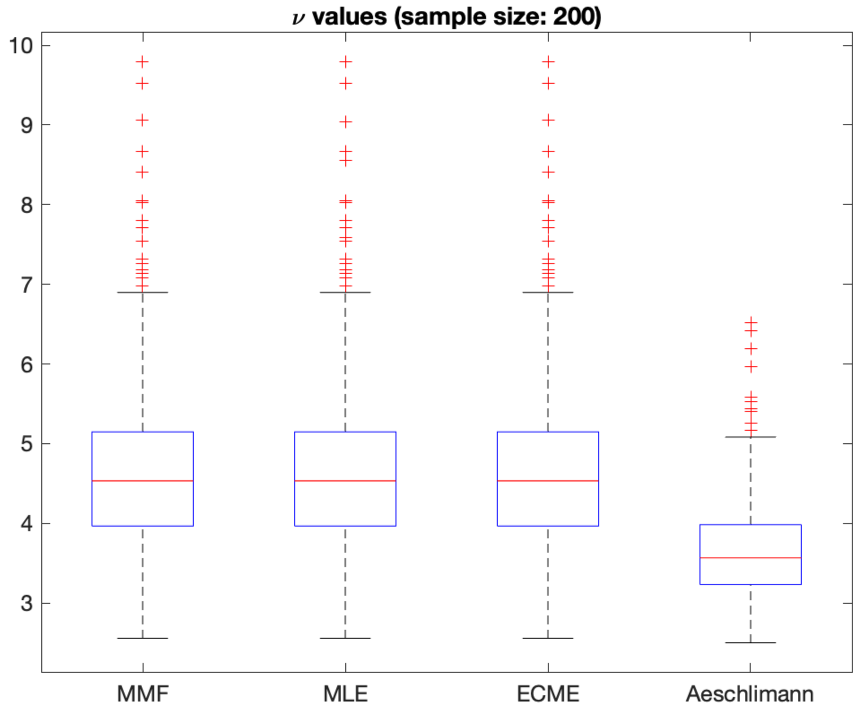

Figure 3: Estimated ν values for T=200 using the different methods

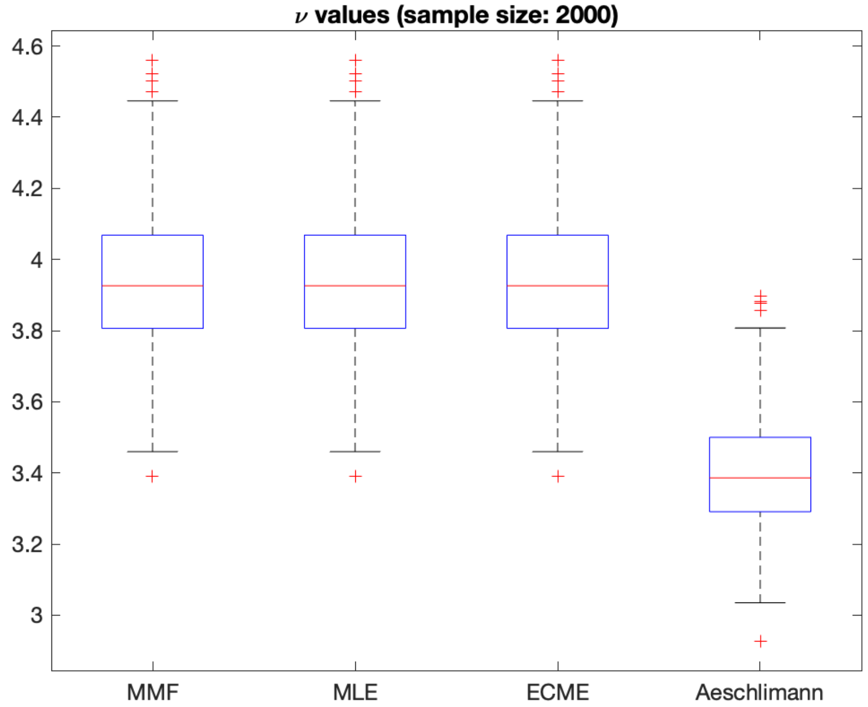

Figure 4: Estimated ν values for T=2000 using the different methods

Estimated Σ values

Table 3: Average estimated Σ for the 500 repetitions of the four methods.

	MMF	MLE	ECME	Aeschliman et al.
Sample size			200	
σ_{11}	0.073	0.073	0.073	-0.148
σ_{22}	0.091	0.091	0.090	-0.114
σ_{33}	-0.034	-0.035	-0.034	-0.177
σ_{12}=σ_{21}	0.033	0.033	0.033	-0.139
σ_{13}=σ_{31}	-0.145	-0.145	-0.145	-0.206
σ_{23}=σ_{32}	-0.122	-0.125	-0.122	-0.169
Sample size			2000	
σ_{11}	0.002	0.002	0.002	-0.171
σ_{22}	0.065	0.065	0.065	-0.118
σ_{33}	-0.003	-0.002	-0.002	-0.167
σ_{12}=σ_{21}	0.030	0.030	0.030	-0.133
σ_{13}=σ_{31}	-0.005	-0.005	-0.005	-0.102
σ_{23}=σ_{32}	0.015	0.015	0.015	-0.073

From Table 3 and Figures 5 and 6, we can clearly see the difference in terms of precision of the four methods once again. More precisely, while MMF, MLE and ECME report almost identical estimations, we can easily notice the lack of accuracy of the Aeschliman et al. methodology. In addition, since the estimates show the deviation from the true values, the approximation methodology always presents higher values in absolute terms, compared to the other three options, representing its lack of accuracy.

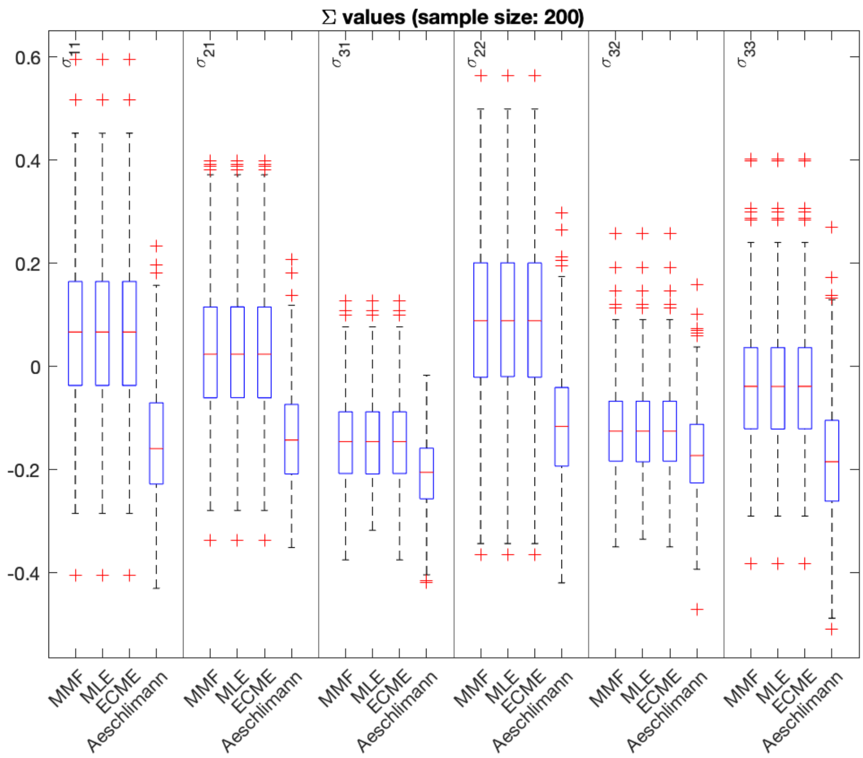

Figure 5: Estimated σ values for T=200 using different methods

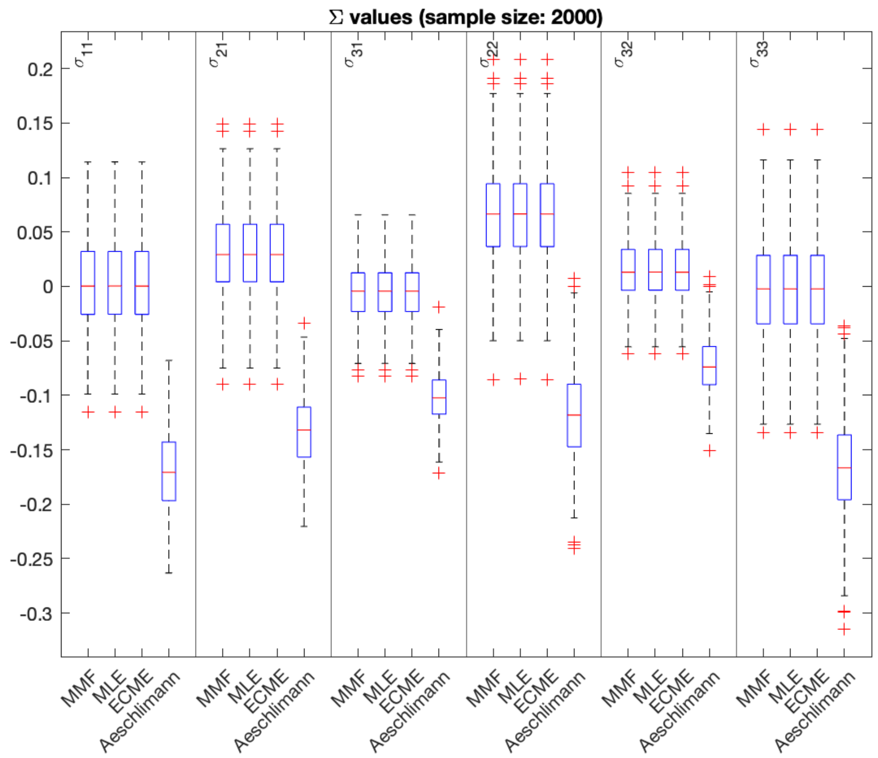

Figure 6: Estimated Σ values for T=2000 using different methods

Average Estimation Time

Table 4: Average estimation time (in seconds) for one run of the four methods.

	MMF	MLE	ECME	Aeschliman et al.
Sample size			200	
Estimation Time	0.5154	1.6222	0.0186	0.0003
Sample size			2000	
Estimation Time	3.4529	13.7758	0.0312	0.0006

As we can see from Table 4, the approximation method proposed by Aeschliman et al. is as expected the fastest, followed by ECME and MMF. Also as expected, the self-made Log likelihood method for MLE is the slowest.

Question 2: Working with Weighted Likelihood

In this second exercise we make use of the weighted likelihood.
The main general code structure in Listing 7 is the same as in Question 1, but the applied estimation functions deviate from Question 1 and also the simulated MVT sequence differs. In this case, we simulate a sequence of MVT random variables such that the observations are independent, but not identical distributed and have some time variation in the prameter ν. ν decreases from 6 to 3 in step-size.

We replace our *function_MMFAlgorithm* function with the function *MMFAlgorithm_weighted* from Listing 8 which calls our implementation of the normal MMF routine, but beforehand calculates the according weights. For this it uses the input variable ρ, which is the parameter dictating the hyperbolic weight decay in equation 13.1 in the Program Listing 13.1 provided in the book "Linear Models and Time-Series Analysis" by Marc S. Paolella. From this we get the weight vector for the T samples.
We compute 200 replications, and within each we estimate the parameters for different values of ρ. ρ increases from 0.2 to 1 in a step-size of 0.1.
In the same replication we also estimate the parameters with the brute force MLE routine for the 3-d MVT. As can be noticed in Listing 9, we adjusted the MLE routine in order to support the likelihood weight vector based on the value of ρ. Same as for the MMF, we created the function *MVTestimation_3d_weighted* that calculates the weights and then calls our normal MLE routine.

The code in Listing 7 contains all the computations for Question 2.

Listing 7: MMF and MLE Parameter Estimation using Weighted Likelihood

```
%%% Create covariance matrix with positive eigenvalues and diagonal values of 1
% Parameters
dim = 3;              % dimensionality
diag = 1;             % diagonal values

% Initializing vectors
M = zeros(3, 3);

while (sum(eig(M)<=0) ~= 0)
    t = triu(bsxfun(@min,diag,-diag.').*rand(dim),1); % The upper triangler random values
    M = diag(diag)+t+t.'; % Put them together in a symmetric matrix
end
```

```
disp('sigma matrix: '); disp(M);
disp('eigenvalues of sigma matrix: '); disp(eig(M));


%%% Simulate 3d MVT and estimate parameters
% Parameters
T = 200;              % sample sizes: [200, 2000] run both separately
rep = 200;            % # repititions: 200
loc = 0;
scale = 1;
reps = 200;           % # repititions for MMF: 200
initial_df = 0.1;

% Initializing data matrix
data_w = zeros([T,3]);

% Grid of T df parameter values, going from 6 to 3 in step size
dfvec = linspace(6,3,T);

% Vector with rho values from 0.2 to 1
rho_vec = linspace(0.2,1,9);

% true values
mu_true = [0 0 0];
df_true = 4;
sigma_true = M;

% Initializing vectors
mus_mmf = zeros([rep, length(rho_vec)*dim]);
nus_mmf = zeros([rep, length(rho_vec)]);
sigmas_mmf = zeros(rep, length(rho_vec)*6);
mus_mmf_adj = zeros(rep, length(rho_vec)*dim);
nus_mmf_adj = zeros(rep, length(rho_vec));
sigmas_mmf_adj = zeros(rep, length(rho_vec)*6);

mus_mle = zeros([rep, length(rho_vec)*dim]);
nus_mle = zeros([rep, length(rho_vec)]);
sigmas_mle = zeros(rep, length(rho_vec)*6);
mus_mle_adj = zeros(rep, length(rho_vec)*dim);
nus_mle_adj = zeros(rep, length(rho_vec));
sigmas_mle_adj = zeros(rep, length(rho_vec)*6);

time_mmf = zeros(rep,length(rho_vec));
time_mle = zeros(rep,length(rho_vec));

rs = 1; ts = 0; tm = 0;
```

```


for r = 1:rep

    % Simulate a sequence of MVT random variables such that the
    % observations are independent, but not identically distributed
    % and have some time variation in the parameters
    for t = 1:T
        data_w(t,:) = mvtrnd(sigma_true,dfvec(t));
    end

    for rho = 1:length(rho_vec)

        %%% estimating parameters through weighted MMF
        % and creating adjusted estimates by true values
        tic
        [nu, nu_vec, mu, sigma] = MMFAlgorithm_weighted(rho_vec(rho), data_w', initial_df, reps);
        time_mmf(r,rho) = toc;

        mus_mmf(r,1+tm) = mu(1);    mus_mmf_adj(r,1+tm) = mu(1) - mu_true(1);
        mus_mmf(r,2+tm) = mu(2);    mus_mmf_adj(r,2+tm) = mu(2) - mu_true(2);
        mus_mmf(r,3+tm) = mu(3);    mus_mmf_adj(r,3+tm) = mu(3) - mu_true(3);

        nus_mmf(r,rho) = nu;

        sigmas_mmf(r,1+ts) = sigma(1,1);  sigmas_mmf_adj(r,1+ts) = sigma(1,1) - M(1,1);
        sigmas_mmf(r,2+ts) = sigma(2,1);  sigmas_mmf_adj(r,2+ts) = sigma(2,1) - M(2,1);
        sigmas_mmf(r,3+ts) = sigma(3,1);  sigmas_mmf_adj(r,3+ts) = sigma(3,1) - M(3,1);
        sigmas_mmf(r,4+ts) = sigma(2,1);  sigmas_mmf_adj(r,4+ts) = sigma(2,2) - M(2,2);
        sigmas_mmf(r,5+ts) = sigma(3,2);  sigmas_mmf_adj(r,5+ts) = sigma(3,2) - M(3,2);
        sigmas_mmf(r,6+ts) = sigma(3,3);  sigmas_mmf_adj(r,6+ts) = sigma(3,3) - M(3,3);



        %%% estimating parameters through weighted MLE
        % and creating adjusted estimates by true values
```

```
        tic
        [param,stderr,iters,loglik,Varcov] = MVTestimation_3d_weighted(data_w, rho_vec(rho));
        time_mle(r,rho) = toc;

        % (k, mu1, mu2, mu3, Sigma_11, Sigma_12, Sigma_13, Sigma_22, Sigma_23, Sigma_33)
        mus_mle(r,1+tm) = param(2);    mus_mle_adj(r,1+tm) = param(2) - mu_true(1);
        mus_mle(r,2+tm) = param(3);    mus_mle_adj(r,2+tm) = param(3) - mu_true(2);
        mus_mle(r,3+tm) = param(4);    mus_mle_adj(r,3+tm) = param(4) - mu_true(3);

        nus_mle(r,rho) = param(1);    nus_mle_adj(r,rho) = param(1) - df_true;

        sigmas_mle(r,1+ts) = param(5);  sigmas_mle_adj(r,1+ts) = param(5) - M(1,1);
        sigmas_mle(r,2+ts) = param(6);  sigmas_mle_adj(r,2+ts) = param(6) - M(2,1);
        sigmas_mle(r,3+ts) = param(7);  sigmas_mle_adj(r,3+ts) = param(7) - M(3,1);
        sigmas_mle(r,4+ts) = param(8);  sigmas_mle_adj(r,4+ts) = param(8) - M(2,2);
        sigmas_mle(r,5+ts) = param(9);  sigmas_mle_adj(r,5+ts) = param(9) - M(3,2);
        sigmas_mle(r,6+ts) = param(10);  sigmas_mle_adj(r,6+ts) = param(10) - M(3,3);

    end
    rs = rs+dim; ts = ts+6; tm = tm+dim;
end



%%% Printing values
disp('=================================')
disp(['Sample size: ', num2str(T)])
disp('***MMF***')
ns = 1; ss = 1;
for n = 1:length(rho_vec)
    disp(['Rho: ', num2str(rho_vec(n))])
    fprintf('Estimation time: %d min %f sec\n', floor(mean(time_mmf(:,n))/60), rem(mean(time_mmf(:,n)),60));
    disp('Mean nu: '); disp(mean(nus_mmf(:,n)))
```

```
    disp('Mean mu: '); disp(mean(mus_mmf(:,ns:ns+2)))
    disp('Mean sigma values (Sigma_11, Sigma_12, Sigma_13, Sigma_22, Sigma_23, Sigma_33): ');
    disp(mean(sigmas_mmf(:,ss:ss+5)))
    ns = ns+dim; ss = ss+6;
end
disp('***MLE***')
ns = 1; ss = 1;
for n = 1:length(rho_vec)
    disp(['Rho: ', num2str(rho_vec(n))])
    fprintf('Estimation time: %d min %f sec\n', floor(mean(time_mle(:,n))/60), rem(mean(time_mle(:,n)),60));
    disp('Mean nu: '); disp(mean(nus_mle(:,n)))
    disp('Mean mu: '); disp(mean(mus_mle(:,ns:ns+2)))
    disp('Mean sigma values (Sigma_11, Sigma_12, Sigma_13, Sigma_22, Sigma_23, Sigma_33): ');
    disp(mean(sigmas_mle(:,ss:ss+5)))
    ns = ns+dim; ss = ss+6;
end
disp('==========================================')


%%% Plotting

%% Grouped plot - nu
hAxes.TickLabelInterpreter = 'latex';
% MMF
figure, lab={};
for i=1:length(rho_vec)
    str=['\rho=', num2str(rho_vec(i))];
    lab=cat(2,lab,str);
end
boxplot([nus_mmf(:,1),nus_mmf(:,2),nus_mmf(:,3),nus_mmf(:,4),nus_mmf(:,5),nus_mmf(:,6),nus_mmf(:,7),nus_mmf(:,8),nus_mmf(:,9)], 'labels', lab, 'Whisker',1.5)
set(gca, 'fontsize' ,12)
title(['MMF \nu values (sample size: ',num2str(T),')'])
name_nu_mmf = ['Assignment3_ex2_mmf_',num2str(T),'.png'];
saveas(gcf,name_nu_mmf)

% MLE
hAxes.TickLabelInterpreter = 'latex';
figure, lab={};
for i=1:length(rho_vec)
    str=['\rho= ', num2str(rho_vec(i))];
    lab=cat(2,lab,str);
```

```
end
boxplot([nus_mle(:,1),nus_mle(:,2),nus_mle(:,3),nus_mle(:,4),nus_mle(:,5),nus_mle(:,6),nus_mle(:,7),nus_mle(:,8),nus_mle(:,9)], 'labels', lab, 'Whisker',1.5)
set(gca, 'fontsize' ,12)
title(['MLE \nu values (sample size: ',num2str(T),')'])
name_nu_mle = ['Assignment3_ex2_mle_',num2str(T),'.png'];
saveas(gcf,name_nu_mle)
```

Listing 8: Function computes the weight vector for the MMF Algorithm and calls MMF routine

```
function [final_nu, nu_vec, mu, sigma] = MMFAlgorithm_weighted(rho, x, initial_df, reps)

% Program Listing 13.1 - Linear Models and Time series
% compute the weight vector
T = length(x); tvec = (1:T); omega=(T-tvec + 1).^(rho-1); weights = omega'/sum(omega);
disp(['Sum of weights: ', num2str(sum(weights))]);

% we recall the original MMF algorithm
[final_nu, nu_vec, mu, sigma] = function_MMFAlgorithm(x, initial_df, weights, reps);
end
```

Listing 9: Function computes the weight vector for the Log Likelihood MLE and calls MLE routine

```
function [param,stderr,iters,loglik,Varcov] = MVTestimation_3d_weighted(x, rho)

% Program Listing 13.1 - Linear Models and Time series
% compute the weight vector
T = length(x); tvec = (1:T); omega=(T-tvec + 1).^(rho-1); weights = omega'/sum(omega);
disp(['Sum of weights: ', num2str(sum(weights))]);

% we recall the original MVT algorithm
[param,stderr,iters,loglik,Varcov] = MVTestimation_3d(x, weights)

end
```

As already mentioned, we run the procedure using 200 replications to obtain for each ρ 200 values of estimated ν.

In the following figures we show the boxplots for the 9 ρ values. We do this for a sample size of both 200 and 2000 observation, to see how the precision of the method increases when we raise the number of observations. We of course plot the results for both methods employed, namely MMF and MLE, to compare their performance.

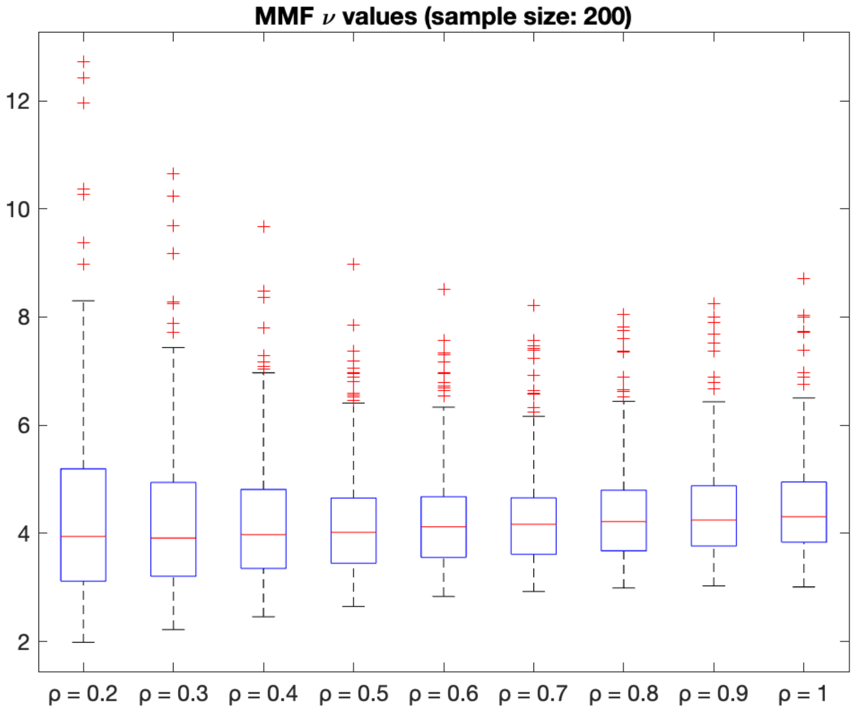

Figure 7: Estimated ν values against ρ for T=200 using MMF

We can see from the Tables 5 and 6, and graphs 7,8,9, and 10, that firstly, the estimation given by MMF and brute force MLE are extremely similar. Secondly, taking the example of T=200 and T=2000 for MMF and MLE, the variance for the estimated ν's increases as we move from ρ =1.0 to ρ = 0.2, as we see that the maximum and the minimum estimated ν's are farther apart when $\rho = 0.2$ as opposed to when ρ =1.0 . We also see that the bias to the true value increases as we move from a lower ρ to a higher one. This can be used to analyse a bias-variance trade-off. We want an estimation with the lowest possible variance and the lowest possible bias. By looking at the plots, we conclude that for T=200, the lowest variance and bias is obtained at ρ= 0.7 for MMF and ρ= 0.6 for brute force MLE. For T=2000, the lowest variance and bias are obtained between ρ= 0.7 and 0.8 for both MMF and brute force MLE. Furthermore, for T=2000, the mean values of estimated ν seem to increase with an increase in ρ for both MMF and MLE. Whereas, for T=200, the mean estimated ν first decreases and then increases or it doesn't even seem to

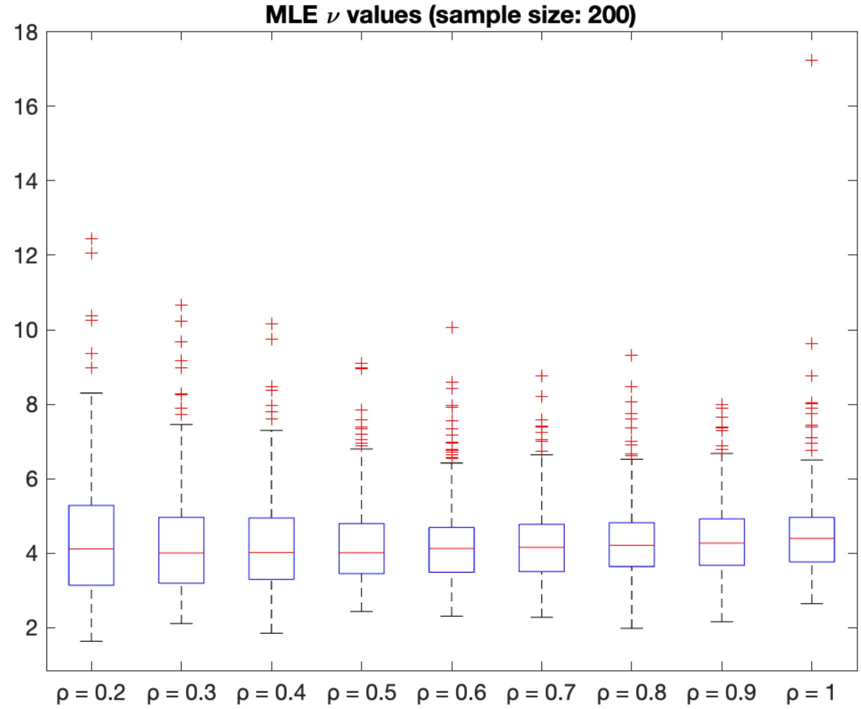

Figure 8: Estimated ν values against ρ for T=200 using Log Likelihood MLE

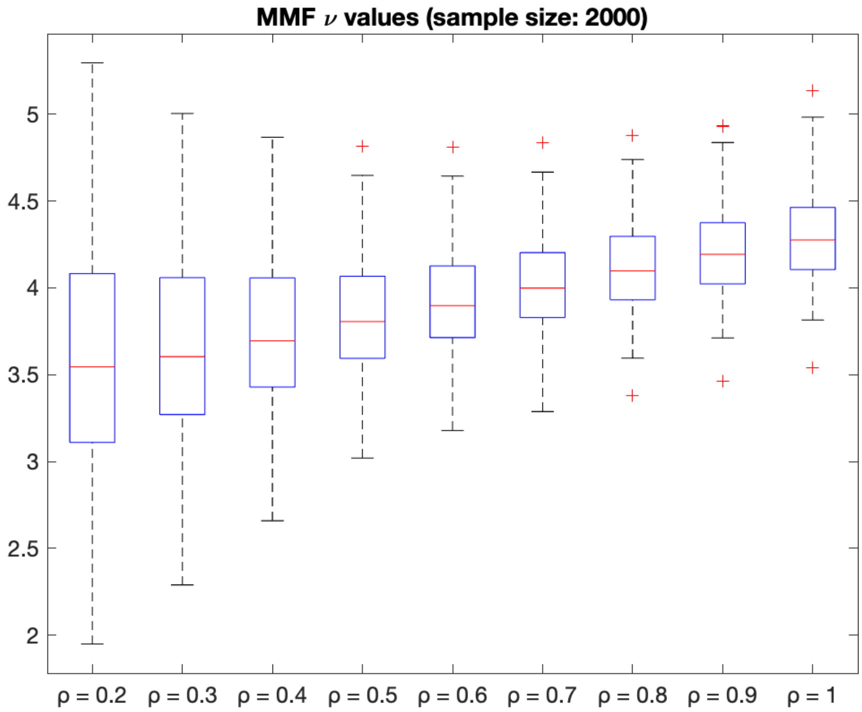

Figure 9: Estimated ν values against ρ for T=2000 using MMF

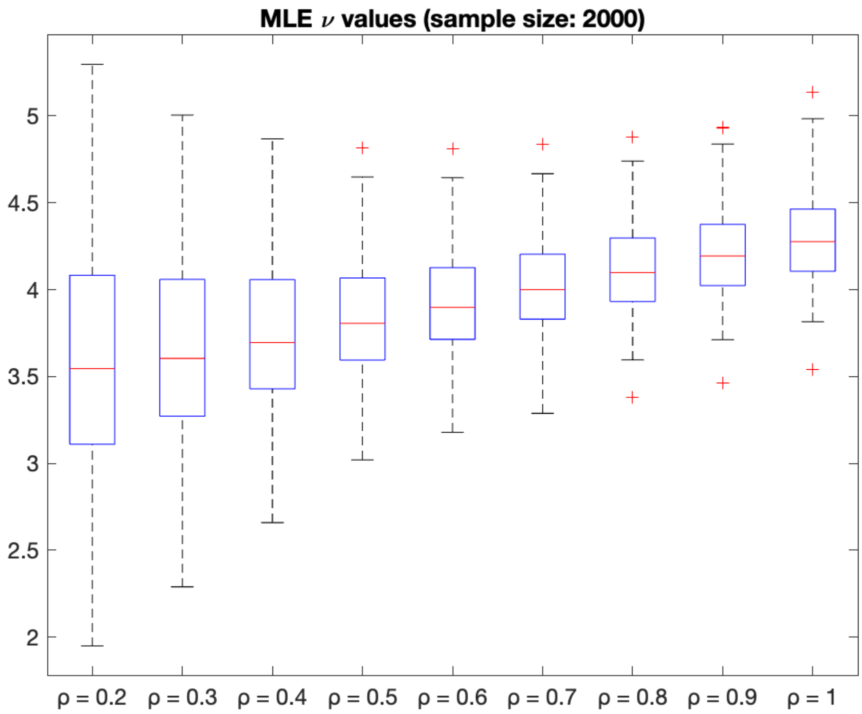

Figure 10: Estimated ν values against ρ for T=2000 using Log Likelihood MLE

follow this pattern in the case of MMF. We also notice that the variance is lower for T=2000 as a larger sample size gives better and more precise estimates.

Table 5: Estimated ν vs. ρ using MMF

ρ	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Sample size					200				
ν	4.4277	4.3222	4.2693	4.2581	4.2793	4.3237	4.3836	4.4532	4.5284
Sample size					2000				
ν	3.6040	3.6568	3.7334	3.8256	3.9248	4.0248	4.1219	4.2141	4.3008

Table 6: Estimated ν vs. ρ using Log Likelihood for MLE

ρ	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Sample size					200				
ν	4.5095	4.3212	4.3129	4.3248	4.3354	4.3176	4.3789	4.4238	4.6074
Sample size					2000				
ν	3.6042	3.6570	3.7336	3.8258	3.9250	4.0251	4.1221	4.2144	4.3011

Average Estimation Time

As we can infer from Table 7, the MMF method again guarantees an efficient estimation time, while MLE is again quite slow.

Table 7: Average estimation time (in seconds) for one run

	MMF	MLE
Sample size	200	
Estimation Time	0.5154	1.6222
Sample size	2000	
Estimation Time	3.4529	13.7758

Hence, for a higher sample size, MMF shows a significant estimation time advantage and also a slight enhanced estimation precision performance.