University of Zurich UZH

Faculty of Business, Economics and Informatics

Statistical Foundations for Finance

Prof. Dr. Marc Paolella

Assignment 3

Authors:	Broennimann, Nicoletta	16-738-148
	Jezler, Linda	15-703-333
	Villiger, Cesare	17-920-208

In Partial Fulfillment of the Requirements for the Course "Statistical Foundations for Finance (Mathematical and Computational Statistics with a View Towards Finance)".

Date of Submission
3 December 2022

Contents

1 Multivariate t Distribution

In the first question, we look at the simulation and estimation of a Multivariate t Distribution (MVT). In section 1.1, we provide an implementation of the Multivariate Myriad Filter (MMF), which is similar to the expectation-maximization algorithm (EM), to estimate the parameters of a MVT distribution. In section 1.2, we provide an implementation of the brut-force maximum likelihood estimation (MLE) for the same parameters. In section 1.3, we simulate MVT random variates and find a general construction for the correlation matrix Σ. In the same script, we iterate over different sample sizes and repetitions for all four estimation techniques: MMF[1], MLE, ECEM[2] and the fast approximation[3] (denoted 'Fast' from here on forth). Finally, we plot the results in boxplots. This corresponds to question 1.b)-d) in the assignment. In the last section 1.4, we show the boxplots and discuss the results.

1.1 MMF Algorithm for Parameter Estimation

In this section, we look at the implementation of MMF algorithm by Hasannasab et al. (2021). They determine three algorithms as alternatives to the classical EM algorithm, namely the accelatered EM-like algorithm (aEM), Multivariate Myrias Filter (MMF) algorithm and Generalized MMF (GMMF). The MMF algorithm is similar to the aEM algorithm except for the (what we will call it from here on forth) df function:

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

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

where v stands for the degrees of freedom (we denote v by df), d for the number of dimensions and δ is a weight computed in the expectation step. They allow for arbitrary weights w_i, given they sum up to 1.
In the following code, we look at the implementation of the MMF algorithm provided in pseudocode. The only tricky part in the implementation of the pseudocode has been the df function. Firstly, we need to pay attention w.r.t. which variable we find the zeros in the update step for the df. We should find the zeros w.r.t. v and not v_r, as the authors describe in their lemma 3! We also include the function '*phi*' to compute the difference between the digamma function and the log. For this, we need to specify that the input x, which is the one for which we find the zeros, is positive, resp. put it in absolute value. The last step is technically irrelevant because of the following insights that were gained when analyzing the df function more closely:

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

[2]Liu, C., Rubin, D.B. ML estimation of the t distribution using EM and its extensions, ECM and ECME. Statistica Sinica. Vol.5. 19-39 (1995).

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

- Firstly, the authors define the df function only for x > 0. So this should be used as a first and very simple constraint in the finding of zeros. This eliminates the need for the absolute value in the digamma function of '*phi*'.
- Secondly, the df function behaves asymptotically around 0, which can either be seen from a plot of the function (see Appendix: 3.1), or from equations (8) and (9) in the original paper. This means we should exclude even positive values close to zero, as functions designed to find zeros struggle with asymptotic behaviour. Some test showed that using the interval [1, 20] as the initial values already yielded excellent results with no error, when using the built-in '*fzero*' function in Matlab. This simply avoids the asymptotic part of the function, with little loss in information, as the true value for the df is surely not below 1. The function, thus, finds zeros in the specified interval.

The general outline of the whole function was copied from the Fundamental Statistical Inference book[4], which accounts for the presence of outliers.

```
% =======================MMF Function===============================
% This calls the relevant MMF function and removes outliers. As ...
    taken from the statistical inference book.
% ==================================================================

function [ solvec , crit, iter] = mvtMMF(y, d, tol, maxit)
if nargin < 4 , maxit=5e4 ; end , if nargin < 3, tol =1e-6; end
outlier = 0;
while 1
    [solvec, crit, iter] = MMF(y, d, tol, maxit);
    if all (¬isnan ( solvec ) ) , break
    else
        y=sort(y) ; left =y(2)-y(1) ; right =y(end)-y ( end-1) ;
        if left > right , y=y ( 2:  end ) ; else y=y ( 1:  end-1) ...
            ; end
        ' removing an outlier '; outlier = outlier +1;
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function computes the MMF of the MVT
function [solvec, crit, iter] = MMF(y, d, tol, maxit)
% initialise the parameters, with df = epsilon > 0
mu_old = zeros(d,d); df_old = zeros(d,1);
mu_old(1,:) = mean(y); sigma_r = cov(y); df_old(1) = 2;
iter = 0; crit = 0;
len_y = size(y); n = len_y(1);
df_fix = df_old(1);

old = [mu_old sigma_r df_old]; new = zeros(d, 2*d+1);

% Initialise random weight vec: simulate len(y) random numbers and
% normalise them via division by sum
randnmbr = rand(1, n);
```

[4]Paolella, M. S. Fundamental Statistical Inference: A Computational Approach. Vol.216. John Wiley & Sons (2018).

```
w = randnmbr./sum(randnmbr);


% Begin iteration for the EM
while 1
    iter= iter +1;
    % Define parameters for iteration
    mu_r = old(1, 1:d) ; sigma_r = old(:, d+1:2*d); df_r = ...
        old(1, 2*d+1);

    % Append all Δ_ri and gam_ri values
    Δ_r = zeros(1,n);
    gam_r = zeros(1,n);

    % Loop over all n samples
    for i = 1:n

        % E-step: compute the weigths
        % Δ:
        dif1 = y(i,:)-mu_r;
        tim1 = mtimes(dif1, inv(sigma_r));
        Δ_r(i) = mtimes(tim1, dif1');

        % gamma (matrix of over Nxd)
        gam_r(i) = (df_r + d)/(df_r + Δ_r(i));

    end

    % M-step:
    % new mu:
    mu_rplus1 = mtimes((w.*gam_r),y) ./ sum(w.*gam_r);
    new(1, 1:3) = mu_rplus1;

    % new sigma:
    temp1 = (w.*gam_r).*y';
    sigma_rplus1 = mtimes(temp1,y) ./ sum(w.*gam_r);
    new(1:3, 4:6) = sigma_rplus1;

    % Find 0's of the function for df
    temp = (df_r+d)./(df_r+Δ_r);
    df_fun = @(x) phi(x/2) - phi((x+d)/2) + ...
        sum(w.*(temp-log(temp)-1));

    % Find the 0 of the function and append it to array
    df_rplus1 = fzero(df_fun, [1 20]);
    new(1, 2*d+1) = real(df_rplus1);

    % Stopping criteria
    crit = max ( abs ( old(1,end) - new(1,end) ) ) ; solvec=new ...
        ; if any ( isnan ( solvec ) ) , break , end
    if ( crit < tol ) || ( iter ≥ maxit ) , break , end
    old = new;

end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Used to compute phi in the fzero-function
function phi = phi(y)
    y = abs(y);
    phi = psi(y) - log(y);
end
```

1.2 MLE for Parameter Estimation

This code is an adaptation from the code provided in the Time-Series book[5]. We simply extended the framework to accommodate 3-dimensional MVT random variates. This, of course, entails the choice of suitable box constraints, which we accounted for via the '*einschrk*' function. The function to compute the pdf of MVT random variates was provided in the book as well.

```
% =======================MLE Function================================
% This function calculates the MLE as taken from the statistical ...
    inference book.
% ==================================================================

function [param,stderr,iters,loglik,Varcov] = MVTestimation_3d(x)
[nobs, d]=size(x);

%%%%%%%% k mu1 mu2 mu3 s11 s12 s22 s13 s33 s23
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 =[1 -0.8 -0.2 -0.5 20 0 20 0 20 0];

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),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 ∆ 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:d+1);
Sig(1,1)=param(d+2); Sig(2,2)=param(d+4); Sig(1,2)=param(d+3); ...
    Sig(2,1)=Sig(1,2);
```

[5]Paolella, M. S. Linear models and time-series analysis: regression, ANOVA, ARMA and GARCH. John Wiley & Sons (2018).

```
Sig(3,3)=param(d+6); Sig(1,3)=param(d+5); Sig(3,1)=Sig(1,3); ...
    Sig(2,3)=param(d+7);
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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 a correlation (not dispersion) matrix.
% So, just need to do the usual scale transform. 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);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 e r r o r s
    trans=2*pin .* (lo-hi ) ./ (1+pin .^2) .^2;
    d=(1-welche) + welche .* trans ; % either unity or △ method .
    J=diag (d) ; Vout = J* Vin * J ;
end
```

1.3 Code for Output

The generation of random samples takes $\mathbf{y} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ and $\gamma \sim \chi^2(\nu)$ to obtain:

$$\mathbf{x} = \sqrt{\frac{\nu}{\gamma}} T'\mathbf{y} + c, \tag{2}$$

where $\mathbf{x} \sim MVT(\mathbf{c}, T'T, \nu)$. The implementation in Matlab looks like this:

```
function [x] = MVTrandom(Sigma,df,T,mu)
    [¬, d] = size(Sigma);
    mu_normal = zeros(1,d);

    Y = mvnrnd(mu_normal,Sigma,T); % The normal needs to be ...
        location 0
    U = chi2rnd(df);
    x = mu + Y*sqrt(df./U);

end
```

Initially, we used the '*mvtrnd*' function which is built-into Matlab. This led to very weird parameter estimates of the diagonal values in Σ. With great effort, we figured out that this implementation transforms any covariance matrix, given as input, into a correlation matrix. So, only values of 1 on the diagonal are estimated correctly. As this is not what we want, we use our own implementation.

We then estimate the parameters via the MMF, MLE, ECME and Fast for different sample sizes and plot the results in boxplots for visual comparison. In order to simulate random variates, we construct a symmetric, positive definite (PD) covariance matrix. Instead of guessing values, we have generalized this procedure in the following way: We start with a matrix of random values between -1 and 1, which we transform into a symmetric matrix. Since the resulting matrix is only positive semi-definite (PSD), we add perturbations along the diagonal to make it PD. Adding these perturbations along the diagonal elevates the values above 1 and the matrix thus becomes a covariance matrix.
The functions for the computation of the MMF and MLE were discussed in the two previous sections. The functions for the ECME and Fast can be found under the link in this footnote[6].
Finally, we get to the boxplots of the estimated parameters. We estimate 10 parameters, one for the df, three for the location parameter μ and six for the correlation matrix Σ (technically, there are nine parameters in Σ, but since it is symmetric, we can leave away the equivalent values on the off-diagonals). Since we loop both over the number of samples, T, and the repetitions, the output arrays for the estimates become very large and have to be indexed correctly (which was a very tedious task), so that the whole output can be displayed in one run of the script.

[6]https://github.com/robince/tdistfituser-content-fn-1-4d141606ec47b515bc78789792e0c34e

```
% ========================Exercise 1==================================
% We simulate 200 and 2000 random variates of a multivariate t
% distribution and estimte the parameters via MMF and MLE.
% The outputs are plotted as boxplots.
% ===================================================================

% Initialise parameters
df = 4; d = 3; mu = zeros(1, d); %T = 200;

% Initialise all arrays to append all 10 parameters of the outputs
dfvecMMF = []; dfvecMLE = []; dfvecEC = []; dfvecFast = [];
muvecMMF_x = []; muvecMLE_x = []; muvecEC_x = []; muvecFast_x = [];
muvecMMF_y = []; muvecMLE_y = []; muvecEC_y = []; muvecFast_y = [];
muvecMMF_z = []; muvecMLE_z = []; muvecEC_z = []; muvecFast_z = [];
sigvecMMF_11 = []; sigvecMLE_11 = []; sigvecEC_11 = []; ...
    sigvecFast_11 = [];
sigvecMMF_22 = []; sigvecMLE_22 = []; sigvecEC_12 = []; ...
    sigvecFast_12 = [];
sigvecMMF_33 = []; sigvecMLE_33 = []; sigvecEC_22 = []; ...
    sigvecFast_22 = [];
sigvecMMF_12 = []; sigvecMLE_12 = []; sigvecEC_13 = []; ...
    sigvecFast_13 = [];
sigvecMMF_13 = []; sigvecMLE_13 = []; sigvecEC_33 = []; ...
    sigvecFast_33 = [];
sigvecMMF_23 = []; sigvecMLE_23 = []; sigvecEC_23 = []; ...
    sigvecFast_23 = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generate a random n x n matrix
sigma = -1 + 2.*rand(d,d);
% % construct a symmetric matrix
sigma = triu(sigma.',1) + tril(sigma);
% % Adding perturbations along the diagonal of a PSD matrix will ...
    make it PD
sigma = sigma + d*eye(d)
% Check if sigma is PD
%{
symm = issymmetric(sigma)
d = eig(sigma)
isposdef = all(d>0)
%}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For loop over sample sizes and repetitions for all methods.
for T = [200 2000]

    for rep = 1:1 rep

        % Simulate random variates from MVT with sample size T
        y = MVTrandom(sigma, df, T, [0 0 0]);

        % Estimate parameters with MMF
        z = mvtMMF(y, d, 1e-3, 100);

        % Estiamte parameters with MLE
        [param,stderr,iters,loglik,Varcov] = MVTestimation_3d(y);

        % Estimate parameters with ECME
        [mu_EC, Sig_EC, df_EC] = fitt_approx(y);

        % Estimate parameters with the fast approximation ...
            algorithm for the
        % MLE.
        [mu_Fast, Sig_Fast, df_Fast] = fitt(y);

        %============================================================
        % Append outputs to arrays
        % Append parameters to arrays for output (df)
        dfvecMMF(end+1) = z(1,end); dfvecMLE(end+1) = param(1);
        dfvecEC(end+1) = df_EC; dfvecFast(end+1) = df_Fast;

        % Append parameters to arrays for output (mu)
        muvecMMF_x(end+1) = z(1, 1); muvecMLE_x(end+1) = param(2);
        muvecEC_x(end+1) = mu_EC(1,1); muvecFast_x(end+1) = ...
            mu_Fast(1,1);

        muvecMMF_y(end+1) = z(1, 2); muvecMLE_y(end+1) = param(3);
        muvecEC_y(end+1) = mu_EC(1,2); muvecFast_y(end+1) = ...
            mu_Fast(1,2);

        muvecMMF_z(end+1) = z(1, 3); muvecMLE_z(end+1) = param(4);
        muvecEC_z(end+1) = mu_EC(1,3); muvecFast_z(end+1) = ...
            mu_Fast(1,3);

        % Append parameters to arrays for output (Sigma)
        sigvecMMF_11(end+1) = z(1,d+1); sigvecMLE_11(end+1) = ...
            param(d+2);
        sigvecEC_11(end+1) = Sig_EC(1,1); sigvecFast_11(end+1) = ...
            Sig_Fast(1,1);

        sigvecMMF_12(end+1) = z(1,d+2); sigvecMLE_12(end+1) = ...
            param(d+3);
        sigvecEC_12(end+1) = Sig_EC(1,2); sigvecFast_12(end+1) = ...
            Sig_Fast(1,2);

        sigvecMMF_22(end+1) = z(2,d+2); sigvecMLE_22(end+1) = ...
            param(d+4);
        sigvecEC_22(end+1) = Sig_EC(2,2); sigvecFast_22(end+1) = ...
            Sig_Fast(2,2);

        sigvecMMF_13(end+1) = z(1,d+3); sigvecMLE_13(end+1) = ...
            param(d+5);
        sigvecEC_13(end+1) = Sig_EC(1,3); sigvecFast_13(end+1) = ...
            Sig_Fast(1,3);

        sigvecMMF_33(end+1) = z(3,d+3); sigvecMLE_33(end+1) = ...
            param(d+6);
        sigvecEC_33(end+1) = Sig_EC(3,3); sigvecFast_33(end+1) = ...
            Sig_Fast(3,3);

        sigvecMMF_23(end+1) = z(2,d+3); sigvecMLE_23(end+1) = ...
            param(d+7);
        sigvecEC_23(end+1) = Sig_EC(2,3); sigvecFast_23(end+1) = ...
```

```
                Sig_Fast(2,3);
            %==========================================================

        end

end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BOXPLOT FOR DF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Assign the outputs to variables for the boxplot and subtract ...
    true value
dfvecMMF1 = dfvecMMF(1:rep)'-df;
dfvecMMF2 = dfvecMMF(rep+1:end)'-df;
dfvecMLE1 = dfvecMLE(1:rep)'-df;
dfvecMLE2 = dfvecMLE(rep+1:end)'-df;
dfvecEC1 = dfvecEC(1:rep)'-df;
dfvecEC2 = dfvecEC(rep+1:end)'-df;
dfvecFast1 = dfvecFast(1:rep)'-df;
dfvecFast2 = dfvecFast(rep+1:end)'-df;


% Create boxplot
group1 = [ones(size(dfvecMMF1)); 2 * ones(size(dfvecMMF2)); 3 * ...
    ones(size(dfvecMLE1)); 4 * ones(size(dfvecMLE2));
     5 * ones(size(dfvecEC1)); 6 * ones(size(dfvecEC2)); 7 * ...
        ones(size(dfvecFast1)); 8 * ones(size(dfvecFast2));];

figure(1)
boxplot([dfvecMMF1; dfvecMMF2; dfvecMLE1; dfvecMLE2; dfvecEC1; ...
    dfvecEC2; dfvecFast1; dfvecFast2], group1);
set(gca,'XTickLabel',{'MMF: 200','MMF: 2000', 'MLE: 200','MLE: ...
    2000', 'EC: 200','EC: 2000','Fast: 200','Fast: 2000'})




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BOXPLOT FOR mu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y_bar = mean(y);

% Assign the outputs to variables for the boxplot MMF. Do not ...
    need to
% subtract the true value.
muvecMMF1_x = muvecMMF_x(1:rep)'-y_bar(1);
muvecMMF1_y = muvecMMF_y(1:rep)'-y_bar(2);
muvecMMF1_z = muvecMMF_z(1:rep)'-y_bar(3);
muvecMMF2_x = muvecMMF_x(rep+1:end)'-y_bar(1);
muvecMMF2_y = muvecMMF_y(rep+1:end)'-y_bar(2);
muvecMMF2_z = muvecMMF_z(rep+1:end)'-y_bar(3);

% Assign the outputs to variables for the boxplot MLE.
muvecMLE1_x = muvecMLE_x(1:rep)'-y_bar(1);
muvecMLE1_y = muvecMLE_y(1:rep)'-y_bar(2);
muvecMLE1_z = muvecMLE_z(1:rep)'-y_bar(3);
muvecMLE2_x = muvecMLE_x(rep+1:end)'-y_bar(1);
muvecMLE2_y = muvecMLE_y(rep+1:end)'-y_bar(2);
```

```
muvecMLE2_z = muvecMLE_z(rep+1:end)'-y_bar(3);

% Assign the outputs to variables for the boxplot ECME.
muvecEC1_x = muvecEC_x(1:rep)'-y_bar(1);
muvecEC1_y = muvecEC_y(1:rep)'-y_bar(2);
muvecEC1_z = muvecEC_z(1:rep)'-y_bar(3);
muvecEC2_x = muvecEC_x(rep+1:end)'-y_bar(1);
muvecEC2_y = muvecEC_y(rep+1:end)'-y_bar(2);
muvecEC2_z = muvecEC_z(rep+1:end)'-y_bar(3);

% Assign the outputs to variables for the boxplot fast ...
    approximation.
muvecFast1_x = muvecFast_x(1:rep)'-y_bar(1);
muvecFast1_y = muvecFast_y(1:rep)'-y_bar(2);
muvecFast1_z = muvecFast_z(1:rep)'-y_bar(3);
muvecFast2_x = muvecFast_x(rep+1:end)'-y_bar(1);
muvecFast2_y = muvecFast_y(rep+1:end)'-y_bar(2);
muvecFast2_z = muvecFast_z(rep+1:end)'-y_bar(3);


% Create the two boxplots as subplots
group2 = [ones(size(muvecMMF1_x)); 2 * ones(size(muvecMMF2_x)); ...
    3 * ones(size(muvecMMF1_y));
     4 * ones(size(muvecMMF2_y)); 5 * ones(size(muvecMMF1_z)); 6 ...
        * ones(size(muvecMMF2_z));];
group3 = [ones(size(muvecMLE1_x)); 2 * ones(size(muvecMLE2_x)); ...
    3 * ones(size(muvecMLE1_y));
     4 * ones(size(muvecMLE2_y)); 5 * ones(size(muvecMLE1_z)); 6 ...
        * ones(size(muvecMLE2_z));];
group4 = [ones(size(muvecEC1_x)); 2 * ones(size(muvecEC2_x)); 3 ...
    * ones(size(muvecEC1_y));
     4 * ones(size(muvecEC2_y)); 5 * ones(size(muvecEC1_z)); 6 * ...
        ones(size(muvecEC2_z));];
group5 = [ones(size(muvecFast1_x)); 2 * ...
    ones(size(muvecFast2_x)); 3 * ones(size(muvecFast1_y));
     4 * ones(size(muvecFast2_y)); 5 * ones(size(muvecFast1_z)); ...
        6 * ones(size(muvecFast2_z));];

figure(2)
boxplot([muvecMMF1_x; muvecMMF2_x; muvecMMF1_y; muvecMMF2_y; ...
    muvecMMF1_z; muvecMMF2_z,], group2);
set(gca,'XTickLabel',{'MMF_x: 200','MMF_x: 2000', 'MMF_y: 200', ...
    'MMF_y 2000','MMF_z: 200', 'MMF_z: 2000'})

figure(3)
boxplot([muvecMLE1_x; muvecMLE2_x; muvecMLE1_y; muvecMLE2_y; ...
    muvecMLE1_z; muvecMLE2_z], group3);
set(gca,'XTickLabel',{'MLE_x: 200','MLE_x: 2000', 'MLE_y: 200', ...
    'MLE_y 2000','MLE_z: 200', 'MLE_z: 2000'})

figure(4)
boxplot([muvecEC1_x; muvecEC2_x; muvecEC1_y; muvecEC2_y; ...
    muvecEC1_z; muvecEC2_z,], group4);
set(gca,'XTickLabel',{'EC_x: 200','EC_x: 2000', 'EC_y: 200', ...
    'EC_y 2000','EC_z: 200', 'EC_z: 2000'})

figure(5)
boxplot([muvecFast1_x; muvecFast2_x; muvecFast1_y; muvecFast2_y; ...
    muvecFast1_z; muvecFast2_z], group5);
set(gca,'XTickLabel',{'Fast_x: 200','Fast_x: 2000', 'Fast_y: ...
    200', 'Fast_y 2000','Fast_z: 200', 'Fast_z: 2000'})


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BOXPLOT FOR sigma
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Assign the outputs to variables for the boxplot MMF. Do not ...
    need to
% subtract the true value.
sigvecMMF1_11 = sigvecMMF_11(1:rep)'-sigma(1,1);
sigvecMMF1_12 = sigvecMMF_12(1:rep)'-sigma(1,2);
sigvecMMF1_22 = sigvecMMF_22(1:rep)'-sigma(2,2);
sigvecMMF1_13 = sigvecMMF_13(1:rep)'-sigma(1,3);
sigvecMMF1_33 = sigvecMMF_33(1:rep)'-sigma(3,3);
sigvecMMF1_23 = sigvecMMF_23(1:rep)'-sigma(2,3);
sigvecMMF2_11 = sigvecMMF_11(rep+1:end)'-sigma(1,1);
sigvecMMF2_12 = sigvecMMF_12(rep+1:end)'-sigma(1,2);
sigvecMMF2_22 = sigvecMMF_22(rep+1:end)'-sigma(2,2);
sigvecMMF2_13 = sigvecMMF_13(rep+1:end)'-sigma(1,3);
sigvecMMF2_33 = sigvecMMF_33(rep+1:end)'-sigma(3,3);
sigvecMMF2_23 = sigvecMMF_23(rep+1:end)'-sigma(2,3);


% Assign the outputs to variables for the boxplot MLE.
sigvecMLE1_11 = sigvecMLE_11(1:rep)'-sigma(1,1);
sigvecMLE1_12 = sigvecMLE_12(1:rep)'-sigma(1,2);
sigvecMLE1_22 = sigvecMLE_22(1:rep)'-sigma(2,2);
sigvecMLE1_13 = sigvecMLE_13(1:rep)'-sigma(1,3);
sigvecMLE1_33 = sigvecMLE_33(1:rep)'-sigma(3,3);
sigvecMLE1_23 = sigvecMLE_23(1:rep)'-sigma(2,3);
sigvecMLE2_11 = sigvecMLE_11(rep+1:end)'-sigma(1,1);
sigvecMLE2_12 = sigvecMLE_12(rep+1:end)'-sigma(1,2);
sigvecMLE2_22 = sigvecMLE_22(rep+1:end)'-sigma(2,2);
sigvecMLE2_13 = sigvecMLE_13(rep+1:end)'-sigma(1,3);
sigvecMLE2_33 = sigvecMLE_33(rep+1:end)'-sigma(3,3);
sigvecMLE2_23 = sigvecMLE_23(rep+1:end)'-sigma(2,3);

% Assign the outputs to variables for the boxplot ECME.
sigvecEC1_11 = sigvecEC_11(1:rep)'-sigma(1,1);
sigvecEC1_12 = sigvecEC_12(1:rep)'-sigma(1,2);
sigvecEC1_22 = sigvecEC_22(1:rep)'-sigma(2,2);
sigvecEC1_13 = sigvecEC_13(1:rep)'-sigma(1,3);
sigvecEC1_33 = sigvecEC_33(1:rep)'-sigma(3,3);
sigvecEC1_23 = sigvecEC_23(1:rep)'-sigma(2,3);
sigvecEC2_11 = sigvecEC_11(rep+1:end)'-sigma(1,1);
sigvecEC2_12 = sigvecEC_12(rep+1:end)'-sigma(1,2);
sigvecEC2_22 = sigvecEC_22(rep+1:end)'-sigma(2,2);
sigvecEC2_13 = sigvecEC_13(rep+1:end)'-sigma(1,3);
sigvecEC2_33 = sigvecEC_33(rep+1:end)'-sigma(3,3);
sigvecEC2_23 = sigvecEC_23(rep+1:end)'-sigma(2,3);

% Assign the outputs to variables for the boxplot fast ...
    approximation.
sigvecFast1_11 = sigvecFast_11(1:rep)'-sigma(1,1);
sigvecFast1_12 = sigvecFast_12(1:rep)'-sigma(1,2);
sigvecFast1_22 = sigvecFast_22(1:rep)'-sigma(2,2);
```

```
sigvecFast1_13 = sigvecFast_13(1:rep)'-sigma(1,3);
sigvecFast1_33 = sigvecFast_33(1:rep)'-sigma(3,3);
sigvecFast1_23 = sigvecFast_23(1:rep)'-sigma(2,3);
sigvecFast2_11 = sigvecFast_11(rep+1:end)'-sigma(1,1);
sigvecFast2_12 = sigvecFast_12(rep+1:end)'-sigma(1,2);
sigvecFast2_22 = sigvecFast_22(rep+1:end)'-sigma(2,2);
sigvecFast2_13 = sigvecFast_13(rep+1:end)'-sigma(1,3);
sigvecFast2_33 = sigvecFast_33(rep+1:end)'-sigma(3,3);
sigvecFast2_23 = sigvecFast_23(rep+1:end)'-sigma(2,3);


% Create the two boxplots as subplots
group6 = [ones(size(sigvecMMF1_11)); 2 * ...
    ones(size(sigvecMMF2_11)); 3 * ones(size(sigvecMMF1_12));
     4 * ones(size(sigvecMMF2_12)); 5 * ...
         ones(size(sigvecMMF1_22)); 6 * ones(size(sigvecMMF2_22));
     7 * ones(size(sigvecMMF1_13)); 8 * ...
         ones(size(sigvecMMF2_13)); 9 * ones(size(sigvecMMF1_33));
     10 * ones(size(sigvecMMF2_33)); 11 * ...
         ones(size(sigvecMMF1_23)); 12 * ones(size(sigvecMMF2_23));];

group7 = [ones(size(sigvecMLE1_11)); 2 * ...
    ones(size(sigvecMLE2_11)); 3 * ones(size(sigvecMLE1_12));
     4 * ones(size(sigvecMLE2_12)); 5 * ...
         ones(size(sigvecMLE1_22)); 6 * ones(size(sigvecMLE2_22));
     7 * ones(size(sigvecMLE1_13)); 8 * ...
         ones(size(sigvecMLE2_13)); 9 * ones(size(sigvecMLE1_33));
     10 * ones(size(sigvecMLE2_33)); 11 * ...
         ones(size(sigvecMLE1_23)); 12 * ones(size(sigvecMLE2_23));];

group8 = [ones(size(sigvecMMF1_11)); 2 * ...
    ones(size(sigvecMMF2_11)); 3 * ones(size(sigvecMMF1_12));
     4 * ones(size(sigvecMMF2_12)); 5 * ...
         ones(size(sigvecMMF1_22)); 6 * ones(size(sigvecMMF2_22));
     7 * ones(size(sigvecMMF1_13)); 8 * ...
         ones(size(sigvecMMF2_13)); 9 * ones(size(sigvecMMF1_33));
     10 * ones(size(sigvecMMF2_33)); 11 * ...
         ones(size(sigvecMMF1_23)); 12 * ones(size(sigvecMMF2_23));];

group9 = [ones(size(sigvecMLE1_11)); 2 * ...
    ones(size(sigvecMLE2_11)); 3 * ones(size(sigvecMLE1_12));
     4 * ones(size(sigvecMLE2_12)); 5 * ...
         ones(size(sigvecMLE1_22)); 6 * ones(size(sigvecMLE2_22));
     7 * ones(size(sigvecMLE1_13)); 8 * ...
         ones(size(sigvecMLE2_13)); 9 * ones(size(sigvecMLE1_33));
     10 * ones(size(sigvecMLE2_33)); 11 * ...
         ones(size(sigvecMLE1_23)); 12 * ones(size(sigvecMLE2_23));];

figure(6)
boxplot([sigvecMMF1_11; sigvecMMF2_11; sigvecMMF1_12; ...
    sigvecMMF2_12; sigvecMMF1_22; sigvecMMF2_22; ...
     sigvecMMF1_13; sigvecMMF2_13; sigvecMMF1_33; sigvecMMF2_33; ...
         sigvecMMF1_23; sigvecMMF2_23], group6);
set(gca,'XTickLabel',{'MMF_11: 200','MMF_11: 2000', 'MMF_12: ...
    200','MMF_12: 2000','MMF_22: 200','MMF_22: 2000', ...
     'MMF_13: 200','MMF_13: 2000', 'MMF_33: 200','MMF_33: 2000', ...
         'MMF_23: 200','MMF_23: 2000'})

```

```
figure(7)
boxplot([sigvecMLE1_11; sigvecMLE2_11; sigvecMLE1_12; ...
    sigvecMLE2_12; sigvecMLE1_22; sigvecMLE2_22; ...
     sigvecMLE1_13; sigvecMLE2_13; sigvecMLE1_33; sigvecMLE2_33; ...
         sigvecMLE1_23; sigvecMLE2_23], group7);
set(gca,'XTickLabel',{'MLE_11: 200','MLE_11: 2000', 'MLE_12: ...
    200', 'MLE_12: 2000','MLE_22: 200','MLE_22: 2000', ...
     'MLE_13: 200','MLE_13: 2000', 'MLE_33: 200','MLE_33: 2000', ...
         'MLE_23: 200','MLE_23: 2000',})

figure(8)
boxplot([sigvecEC1_11; sigvecEC2_11; sigvecEC1_12; sigvecEC2_12; ...
    sigvecEC1_22; sigvecEC2_22; ...
     sigvecEC1_13; sigvecEC2_13; sigvecEC1_33; sigvecEC2_33; ...
         sigvecEC1_23; sigvecEC2_23], group8);
set(gca,'XTickLabel',{'EC_11: 200','EC_11: 2000', 'EC_12: ...
    200','EC_12: 2000','EC_22: 200','EC_22: 2000', ...
     'EC_13: 200','EC_13: 2000', 'EC_33: 200','EC_33: 2000', ...
         'EC_23: 200','EC_23: 2000'})

figure(9)
boxplot([sigvecFast1_11; sigvecFast2_11; sigvecFast1_12; ...
    sigvecFast2_12; sigvecFast1_22; sigvecFast2_22; ...
     sigvecFast1_13; sigvecFast2_13; sigvecFast1_33; ...
         sigvecFast2_33; sigvecFast1_23; sigvecFast2_23], group9);
set(gca,'XTickLabel',{'Fast_11: 200','Fast_11: 2000', 'Fast_12: ...
    200', 'Fast_12: 2000','Fast_22: 200','Fast_22: 2000', ...
     'Fast_13: 200','Fast_13: 2000', 'Fast_33: 200','Fast_33: ...
         2000', 'Fast_23: 200','Fast_23: 2000',})
```

1.4 Output

Before we discuss the output, we compare the computation times in table 1 for one run of the same 200 MVT random variates for all methods used: MMF, MLE, ECME and Fast. As expected, the brute force is the slowest by a huge margin. According the Monte Carlo results[7], the MMF method is much faster than the vanilla EM algorithm. It was also shown that the ECME can have a dramatically faster rate of convergence than the EM algorithm[8]. That being said, we can now compare the MMF and the ECME in terms of speed and find that the ECME is roughly three times faster. It was also shown that Fast is significantly faster than EM, "at the expense of slightly decreased accuracy in the estimates"[9]. We are now able to compare all three algorithm that stem from the EM algorithm. We find that Fast takes only $\frac{6}{1000}$ the amount of time compared to the MMF for the same calculation and is thus also significantly faster than than all the others. Now, there is also a caveat, at least for our implementation of the MMF. We apply stopping criteria that could be altered to improve computation time. We use a maximum number

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

[8]Liu, C., and Rubin, D.B. ML estimation of the t distribution using EM and its extensions, ECM and ECME. Statistica Sinica. Vol. 5. 19-390. (1995).

[9]Aeschliman,C., Park, J., and Cak, K.A. A Novel Parameter Estimation Algorithm for the Multivariate t-Distribution and its Application to Computer Vision. Purdue University. (2010).

of iterations and the absolute distance between the current and the updated values in the MMF. If the former was small and the latter large, we would see very short convergence times, however, the results would definitely suffer from this. We used an adequately small tolerance value of 0.001 and 100 iterations. This should be enough not to be misleading with regard to the relative computation time, which is confirmed by the results looking very similar to the ones yielded by the other methods. A second, smaller caveat: we used one single repetition. We could have computed several and took the average value of the computation times to get an even more accurate result. However, we are only really interested in the relative speed, and they roughly persisted over several tries (that we did not report for simplicity). In conclusion, the fast approximation is incredibly fast in comparison to the others and the MLE is aggravatingly slow, using up nearly all computation time for this assignment.

Table 1: Comparison of the Computation Time for One Run of the MMF, MLE, ECME and Fast Approximation Method.

	Time (in sec.)	Normalized Time (in sec.)
MMF	0.1119	1
MLE	20.5520	183.7438
ECME	0.0379	0.3386
Fast	0.0007	0.0061

Note: The column 'Normalized Time' computes the relative times w.r.t. the time for the weighted MMF.

The outputs are displayed in figures 1-9. We used 500 repetitions and sample sizes, T, of 200 and 2000, as suggested. The true value for degrees of freedom (df) was taken to be 4. We compare the estimated values via MMF, MLE, ECME and Fast method, hence the labelling on the x-axis (MMF: 200 means estimation via MMF for 200 samples). We decided to show each parameter for every method of estimation separately, just to show that everything worked fine. Before we dive deeper into the single boxplots, we remind ourselves that we expect the boxplots to be centred around 0. This is because we subtract the true value from all estimated values to get the deviation from the truth.

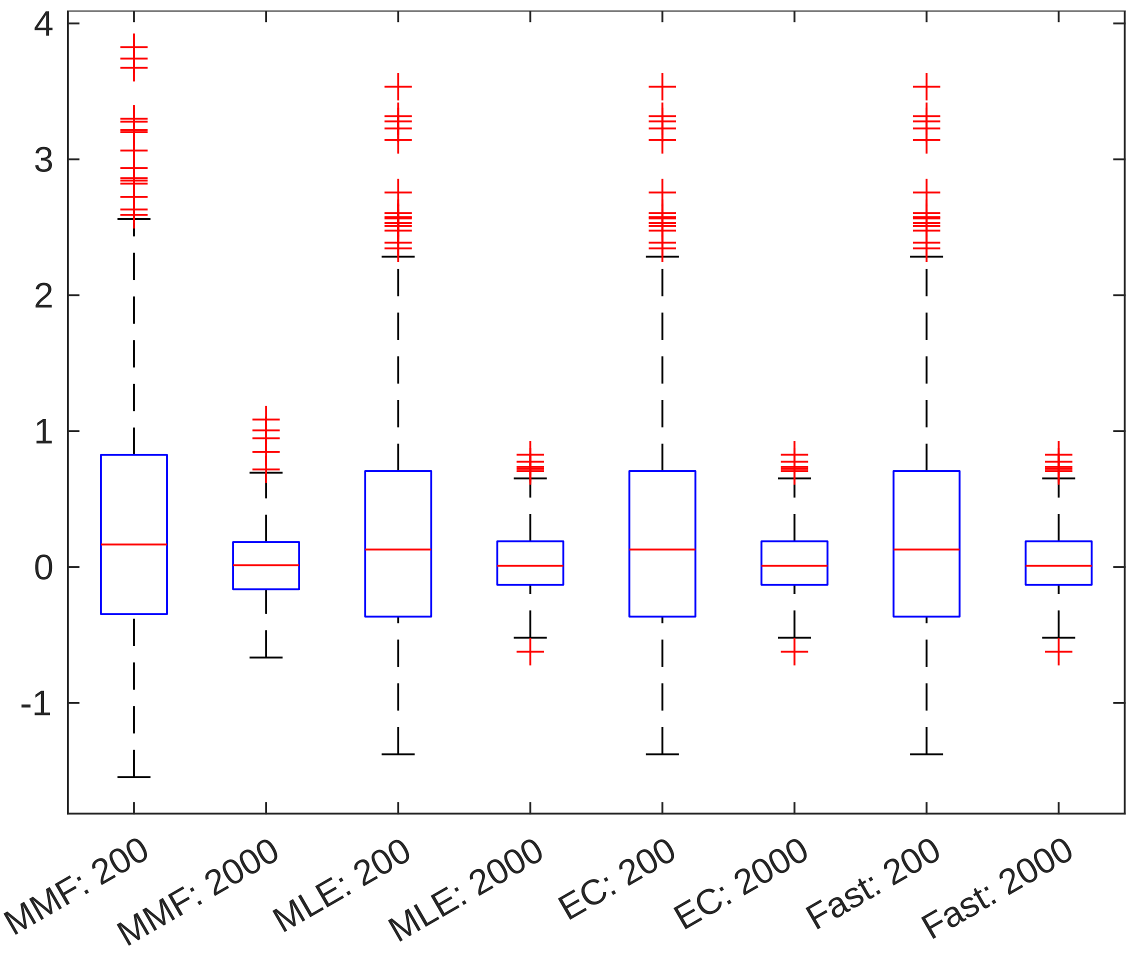
Figure 1 shows the boxplots for the df parameter. We display the results for all methods nicely in one table, which shows that all boxes are centred around 0. Further, it shows that including more samples leads to a much shorter box in all cases. This was to be expected, as more samples increase the accuracy, resp. decrease the variance. The visual comparison between the methods is not so easy in terms of box-size. It seems that the MLE's boxes are slightly shorter than the rest, indicating smaller variance. Paolella (2018) suggests using the direct method for MLE, which could seem reasonable given the outputs presented here. However, they are not very conclusive with regard to the comparison of the two methods, resp. their relative performances. The ECME and Fast also give very similar results, which provides an argument for using the Fast algorithm, considering the much faster computation times. As mentioned before, the Fast algorithm trades off speed for accuracy, still, we only really observe the positive sides of this trade-off. Another thing all methods have in common is that the centres of the boxplots are all slightly above 0, possibly indicating a very small, positive bias. This is corrected when using more samples, as the higher sample boxplots are perfectly centred on 0.

Figures 2-5 show the results for the estimation of the location parameter μ. Since we are looking at a MVT in 3 dimensions, we plot each dimension separately, calling the dimensions x, y and z. For two different sample sizes per array of estimates, we arrive at six different boxplots per method. To plot all methods in one diagram would cause more confusion than it would be informative, so there is a separate diagram for each method. We see, again, that most of the boxes are centred around 0, as they should be. As in the case for the estimation of the df parameter, the methods all give fairly similar results. This advocates for the use of the fastest method.

Finally, we look at the estimates of the correlation matrix Σ in figures 6-9. As discussed before, we only need six estimates, as the mirrored values in the off-diagonal are equivalent in a symmetric matrix. MLE_11:200 thus refers to the MLE of the estimate at index (1,1) for 200 samples. Again, the estimates are centred around 0 and more samples decreases the box height, as was to be expected. The methods give very similar results here as well. Only the ECME in figure 8 shows outliers that are much farther from the centres of the boxplots than in the other estimation techniques. This is indicative of a larger variance, so we suggest using any of the methods other than ECME. Still, everything seems to work as expected. We might add one caveat: we must pay great attention to the Σ we use in the creation of random variates. If the eigenvalues thereof are such that they are close to being PSD, then the estimates of the diagonal values can be biased.

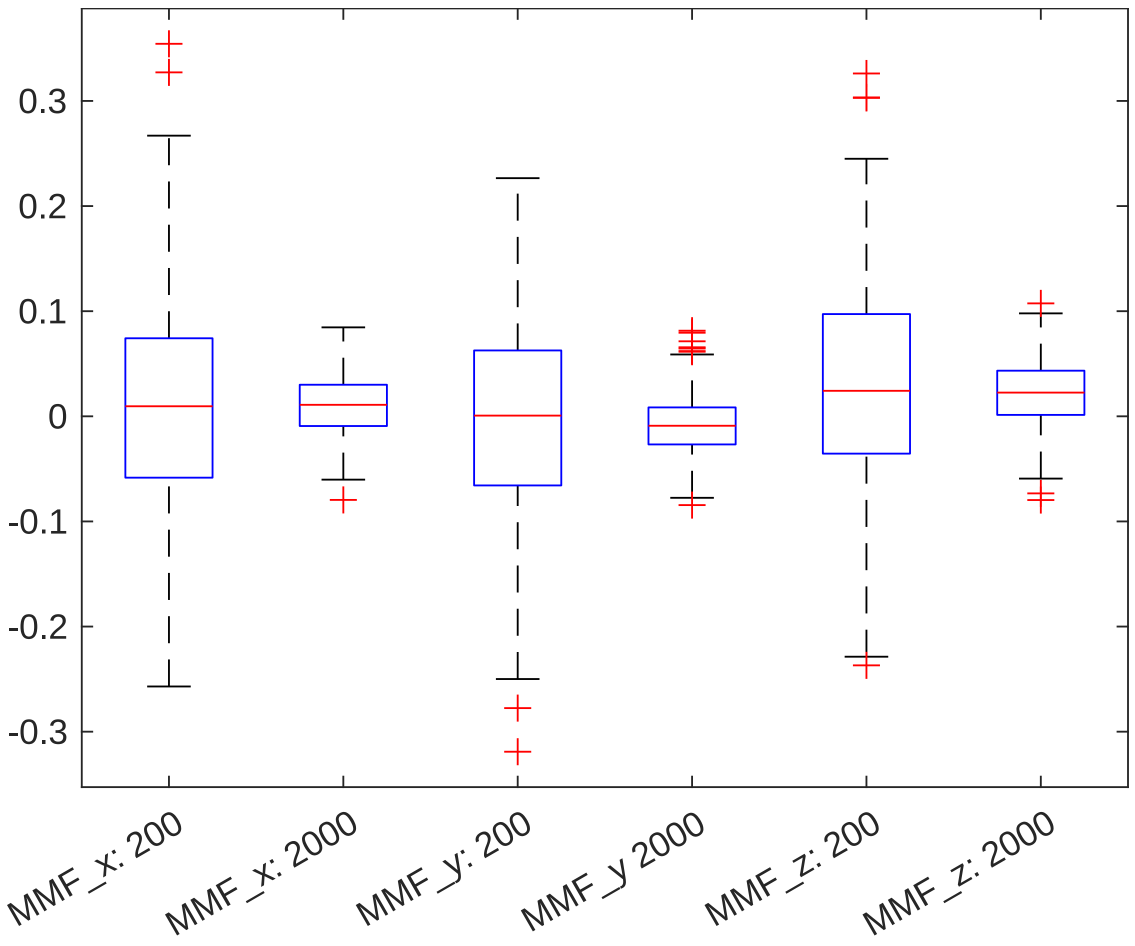
One last thing we did not mention so far are the ranges on the y-axes. First, and most logically, we wanted to use the largest range for all graphs. However, this led to some boxes being incredibly thin, so the visual comparison is rendered even more difficult than it already is. So, we resorted to using the automatically generated y-axis ranges and urge the reader to pay attention to the scales between graphs. We notice that the variances in for the df and location parameters in figures 1-5 have very similar ranges, indicating very similar variances. The estimates of Σ in figures 6-9, however, show larger variances, notably also more outliers. This indicates that the estimates of Σ are the most imprecise of all parameters. As mentioned above, the ECME method performs the worst in terms of estimation variance for Σ.

Figure 1: Estimates of *df* with MMF, MLE, ECME and Fast.



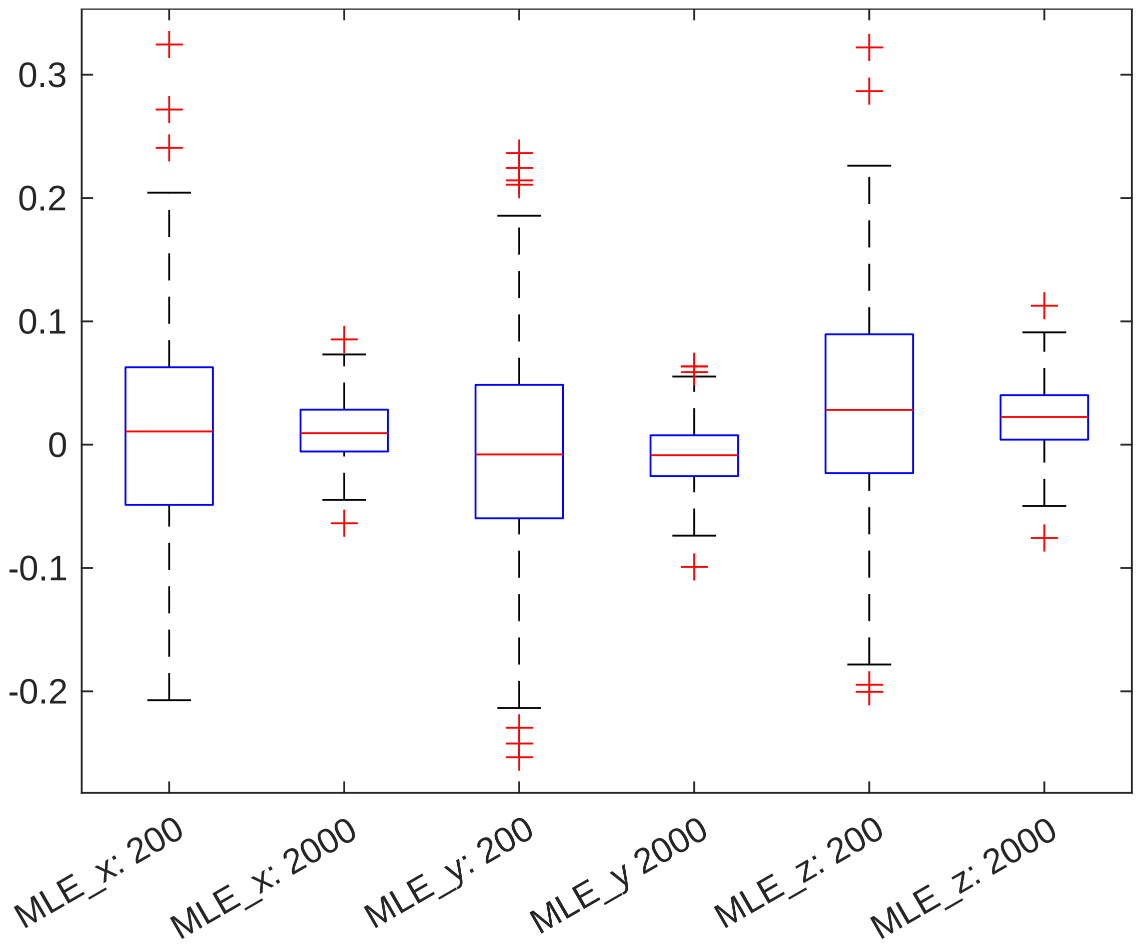
Note: The figure shows the boxplots for the estimates of the degrees of freedom (df) with MMF, MLE, ECME and Fast for the sample sizes T of 200 and 2000.

Figure 2: Estimates for Each Dimension of μ with MMF.



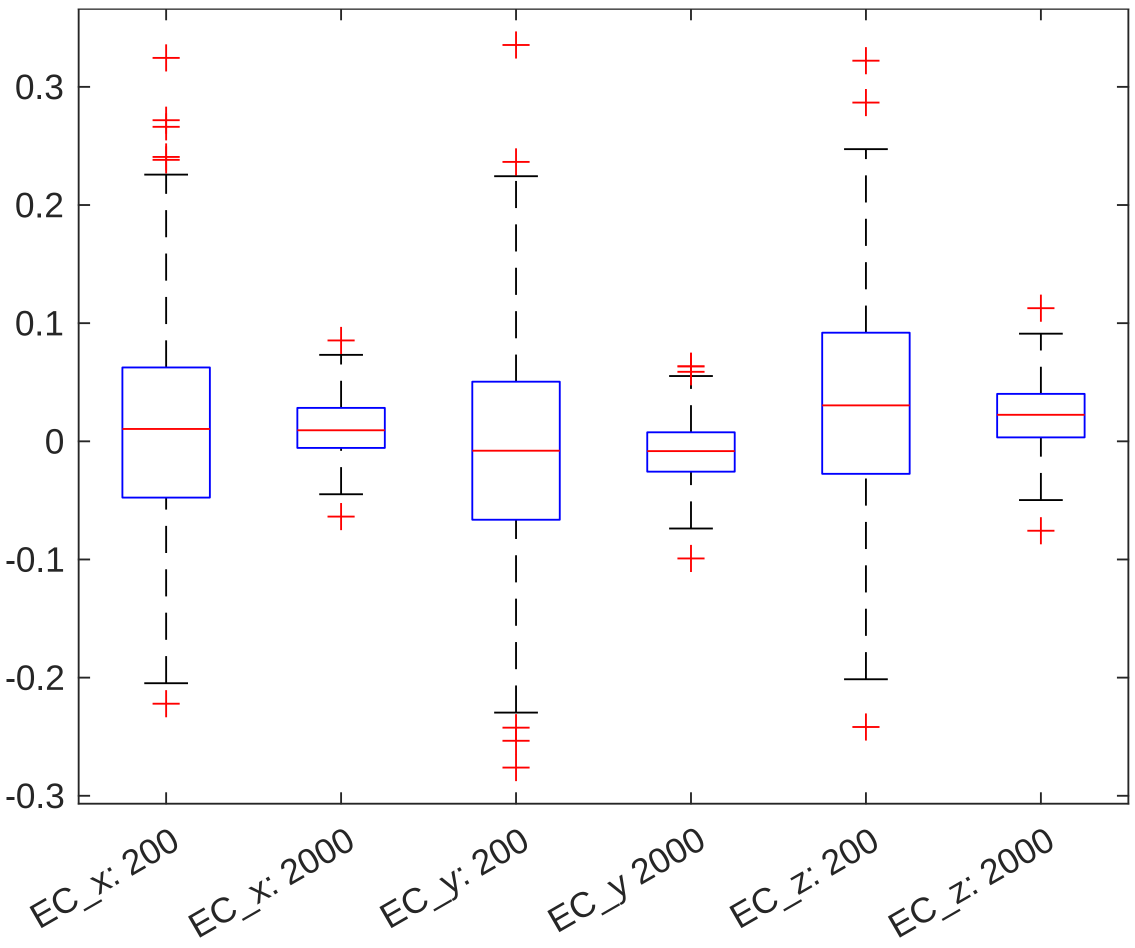
Note: The figure shows the boxplots for the estimates of each dimension of the location parameter μ with MMF for sample sizes T of 200 and 2000.

Figure 3: Estimates for Each Dimension of μ with MLE.



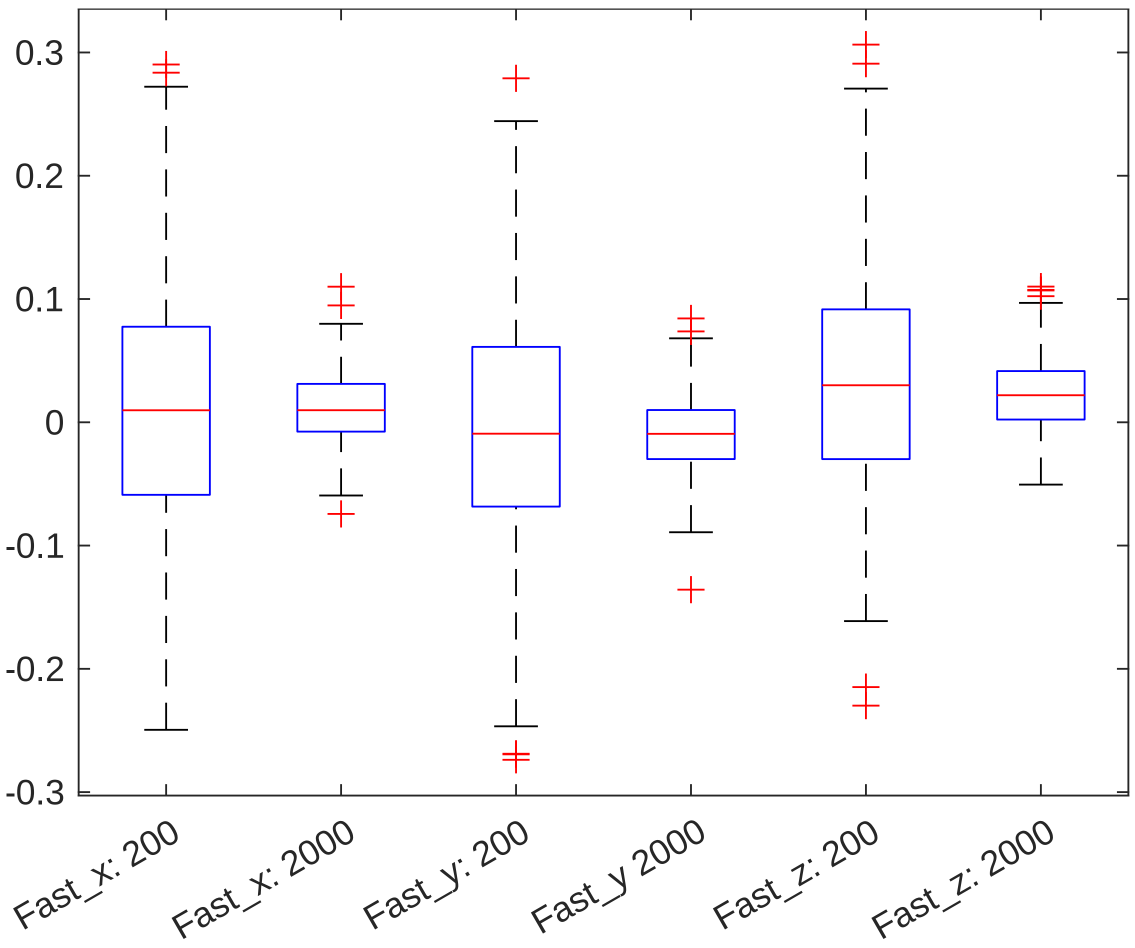
Note: The figure shows the boxplots for the estimates of each dimension of the location parameter μ with MLE for the sample sizes T of 200 and 2000.

Figure 4: Estimates for Each Dimension of μ with ECME.



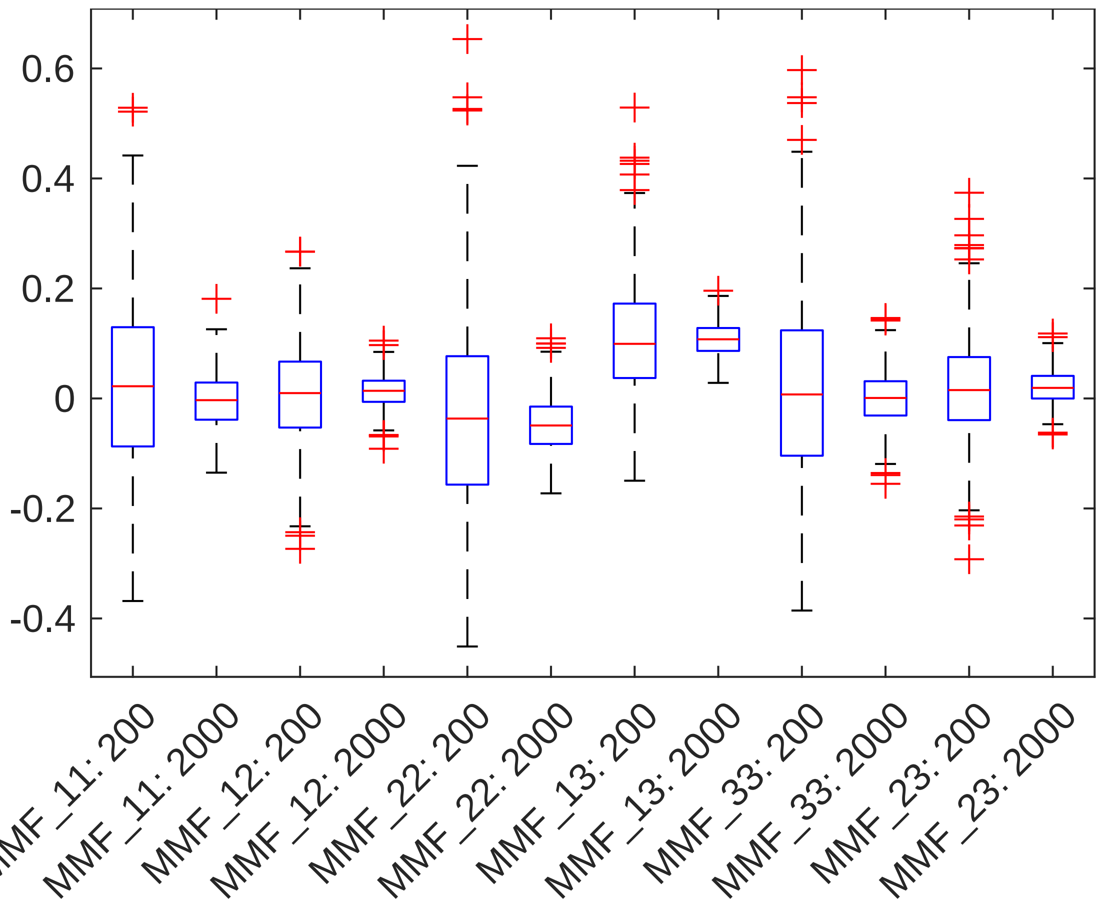
Note: The figure shows the boxplots for the estimates of each dimension of the location parameter μ with ECME for the sample sizes T of 200 and 2000.

Figure 5: Estimates for Each Dimension of μ with Fast.



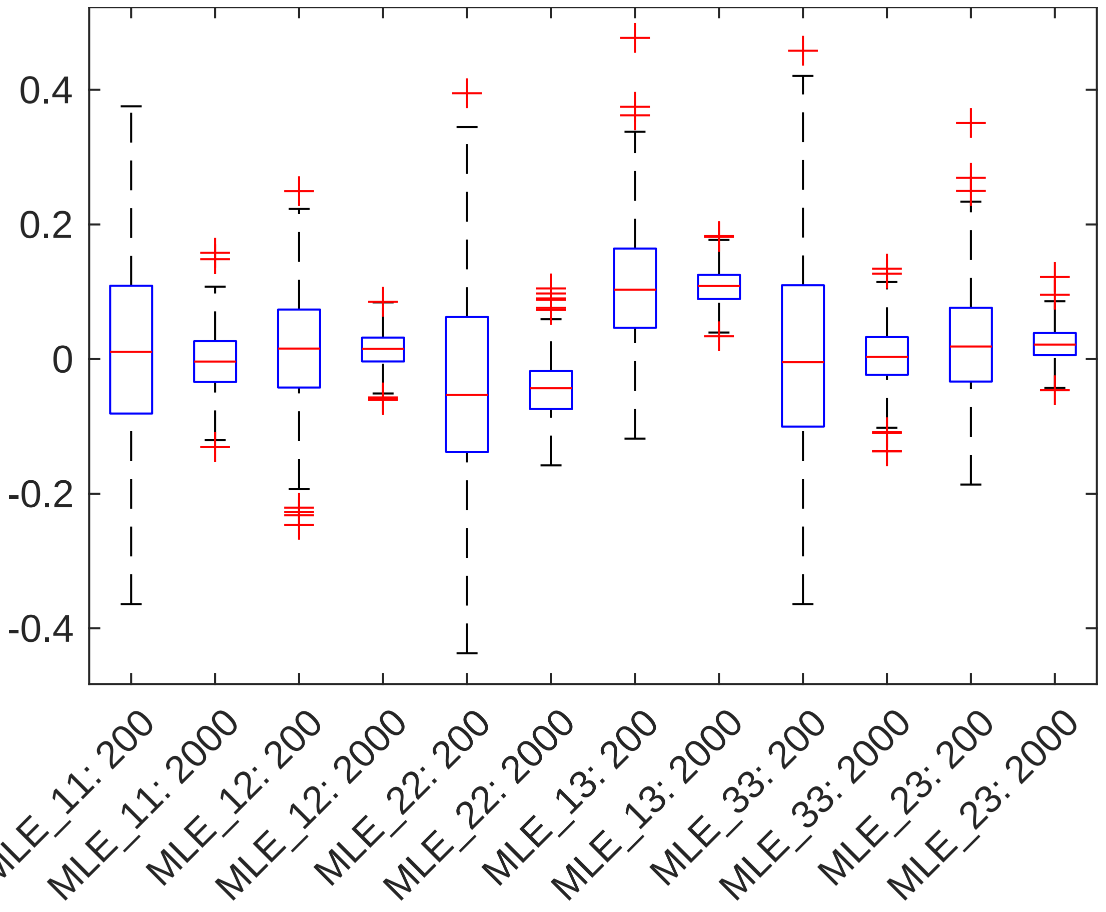
Note: The figure shows the boxplots for the estimates of each dimension of the location parameter μ with Fast for the sample sizes T of 200 and 2000.

Figure 6: Estimates for Each Relevant Index of Σ with MMF.



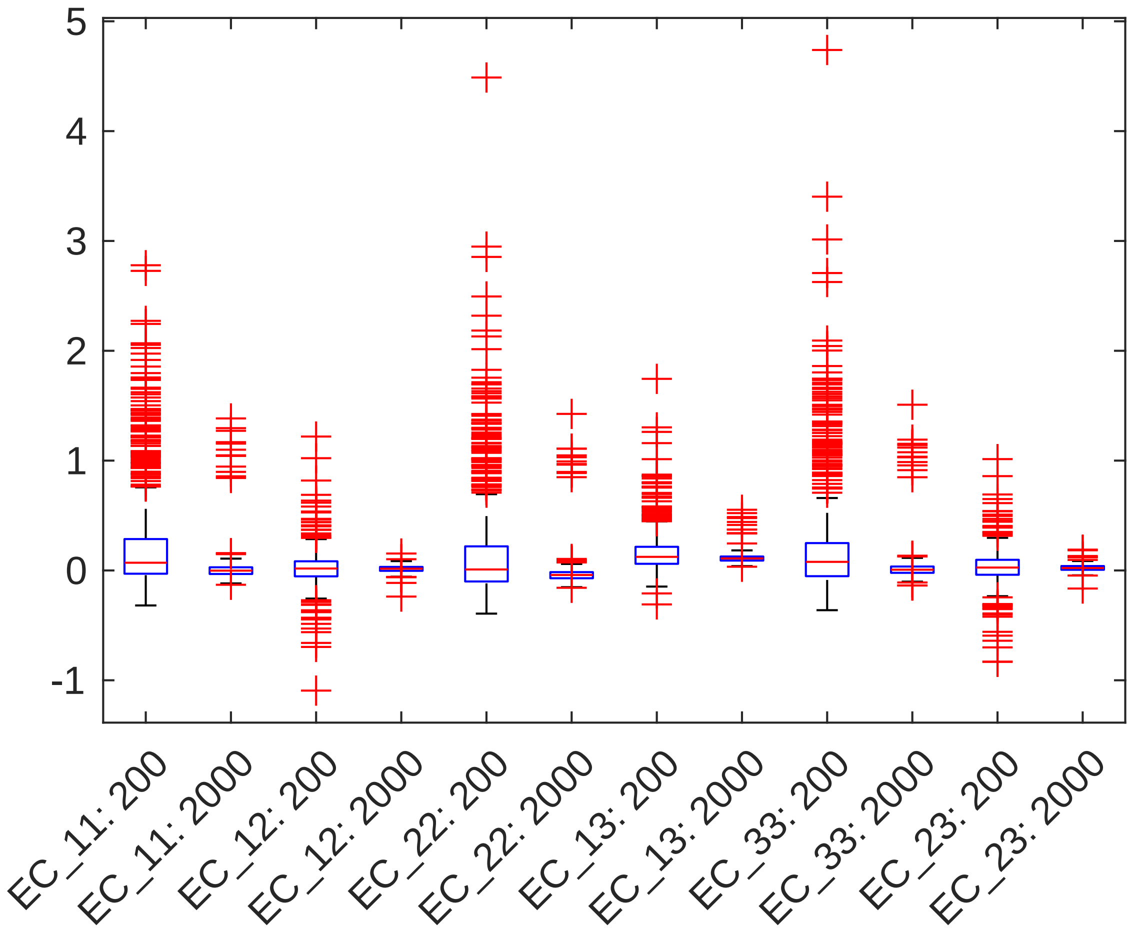
Note: The figure shows the boxplots for the estimates of each relevnat index of the scale parameter Σ with MMF for the sample sizes T of 200 and 2000.

Figure 7: Estimates for Each Relevant Index of Σ with MLE.



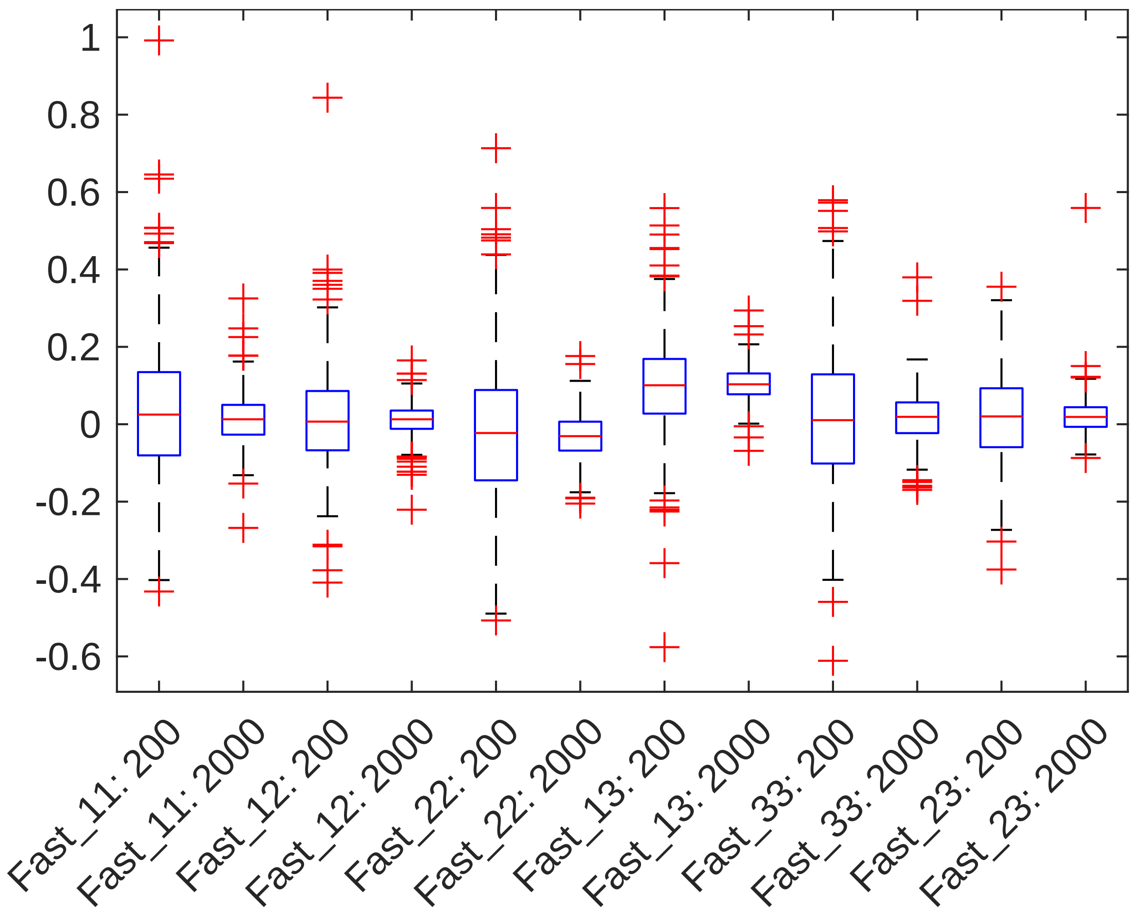
Note: The figure shows the boxplots for the estimates of each relevant index of the scale parameter Σ with MLE for the sample sizes T of 200 and 2000.

Figure 8: Estimates for Each Relevant Index of Σ with ECME.



Note: The figure shows the boxplots for the estimates of each relevnat index of the scale parameter Σ with ECME for the sample sizes of 200 and 2000.

Figure 9: Estimates for Each Relevant Index of Σ with Fast.



Note: The figure shows the boxplots for the estimates of each relevnat index of the scale parameter Σ with Fast for the sample sizes T of 200 and 2000.

2 Usage of Weighted Likelihood

For the second question, we read "Chapter 13: Weighted Likelihood" from "Linear Models and Time-Series Analysis"[10]. We first provide a quick summary of the techniques used in order to justify the need for weighted likelihood estimation.

There are two ways to improve forecast in time-series models without much effort. One can use shrinkage on the one hand to obtain estimators with lower mean-squared errors despite finite amount of data. On the other hand, weighted likelihood results in improved forecasting and accounts for the fact that the proposed model is incorrect due to mis-specification. As we learnt, using an i.i.d setting for financial asset returns results in a mis-specified model, wrongly assuming identically distributed data.
Each observation in traditional likelihood-based interference is equally weighted in the likelihood. This approach is only optimal when the data-generating-process (dgp) is correctly specified. For a financial asset return model this does not hold, since the dgp is rather complicated, making it nearly impossible to specify a proper model. Furthermore, as we aim to construct a forecast at future time T+1 with time-series models, possibly more recent observations contain more information about the distribution at T+1, than more distant observations, and might therefore need to receive more weight[11]. This method would improve the forecasting power of a financial asset return model. Giving more weights to more recent observations, the weighted likelihood accounts for non-linearities in the true dgp. Thus, using a linear time-series model with weighted likelihood is a reasonable approximation to the true dgp of financial asset returns, which is complicated and nonlinear, especially for a multivariate sequence.
For a set of observations v, vector of weights $w = (w_1, ..., w_v)$ is used and standardized to a sum (such as v or 1). A simple hyperbolic weighting scheme looks like this:

$$w_t \propto (v - t + 1)^{\rho - 1}, \quad \sum_{t=1}^{v} w_t = 1, \tag{3}$$

where ρ is a single parameter dictating the shape of weighting function, i.e. $\rho < 1$ gives more recent observations relatively more weight, $\rho > 1$ gives more recent observations relatively less weight, $\rho = 1$ is equal to the standard, equally weighted likelihood.
Both shrinkage and weighted likelihood can and should be used in conjunction as they address different problems of the estimation.

We structure this section like we did with question 1. In section 2.1, we provide the weight-adapted code for the MMF and the same for the MLE in section 2.2 In section 2.3, we provide the code to simulate non-identically distributed MVT random variates and estimate them for different values of ρ over 500 repetitions. Lastly, in section 2.4 we show the resulting boxplots and discuss the results.

[10]Paolella, M. S. Linear models and time-series analysis: regression, ANOVA, ARMA and GARCH. John Wiley & Sons (2019)

[11]Hildreth-Houck find that random disturbance shows certain distributional properties. Secondly, a factor's influence may systematically vary with time. Robert Engle (2001) states that more recent events are more relevant and thus should have more weight in the model.

2.1 Weighted MMF

In the following section we chose ρ to be the parameter dictating the hyperbolic decay from equation 3. The following code extends the code for the MMF in section 1.1 by allowing for the ρ parameter to be passed as an argument. This allows varying weights, depending on the decay rate ρ (line 34). Nota bene, the code in 1.1 already used weights, however, they were just taken to be random. Now we assume they have a structure determined by ρ.
The authors of the paper on the MMF algorithm specify that they allow for arbitrary weights that sum up to 1. The code given in listing 13.1. of the Time-Series book for the calculation of the weights using hyperbolic decay, multiplies the weight vector with T in the last step. This gives the result, that the weights don't sum up to 1. At first, we thought this might be a problem, however, it turns out that the output is not altered by including the factor T.

```
1  % ======================Weighted MMF Function=====================
2  % This calls the relevant MMF function and removes outliers. As ...
       taken from the statistical inference book.
3  % ================================================================
4
5  % This calls the relevant MMF function and removes outliers. As ...
       taken from the statistical inference book.
6  function [ solvec , crit, iter] = mvtMMFweighted(y, d, rho, tol, ...
       maxit)
7  if nargin < 4 , maxit=5e4 ; end , if nargin < 3, tol =1e-6; end
8  outlier = 0;
9  while 1
10     [solvec, crit, iter] = MMF(y, d, rho, tol, maxit);
11     if all (¬isnan ( solvec ) ) , break
12     else
13         y=sort(y) ; left =y(2)-y(1) ; right =y(end)-y ( end-1) ;
14         if left > right , y=y ( 2:  end ) ; else y=y ( 1:  end-1) ...
               ; end
15         ' removing an outlier '; outlier = outlier +1;
16     end
17 end
18 end
19
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 % this function computes the MMF of the MVT
22 function [solvec, crit, iter] = MMF(y, d, rho, tol, maxit)
23 % initialise the parameters, with df = epsilon > 0
24 mu_old = zeros(d,d); df_old = zeros(d,1);
25 mu_old(1,:) = mean(y); sigma_r = cov(y); df_old(1) = 2;
26 iter = 0; crit = 0;
27 len_y = size(y); n = len_y(1);
28
29 old = [mu_old sigma_r df_old]; new = zeros(d, 2*d+1);
30
31 % Initialise random weight vec: simulate len(y) random numbers and
32 % normalise them via division by sum
33 T=length(y); tvec=(1:T); omega=(T-tvec+1).^(rho-1); ...
       w=(omega'./sum(omega))';
34
35 % Begin iteration for the EM
```

```
while 1
    iter= iter +1;
    % Define parameters for iteration
    mu_r = old(1, 1:d) ; sigma_r = old(:, d+1:2*d); df_r = ...
        old(1, 2*d+1);

    % Append all △_ri and gam_ri values
    △_r = zeros(1,n);
    gam_r = zeros(1,n);

    % Loop over all n samples
    for i = 1:n

        % E-step: compute the weigths
        % △:
        dif1 = y(i,:)-mu_r;
        tim1 = mtimes(dif1, inv(sigma_r));
        △_r(i) = mtimes(tim1, dif1');

        % gamma (matrix of over Nxd)
        gam_r(i) = (df_r + d)/(df_r + △_r(i));

    end

    % M-step:
    % new mu:
    mu_rplus1 = mtimes((w.*gam_r),y) ./ sum(w.*gam_r);
    new(1, 1:3) = mu_rplus1;

    % new sigma:
    temp1 = (w.*gam_r).*y';
    sigma_rplus1 = mtimes(temp1,y) ./ sum(w.*gam_r);
    new(1:3, 4:6) = sigma_rplus1;

    % Find 0's of the function for df
    temp = (df_r+d)./(df_r+△_r);
    df_fun = @(x) phi(x/2) - phi((x+d)/2) + ...
        sum(w.*(temp-log(temp)-1));

    % Find the 0 of the function and append it to array
    df_rplus1 = fzero(df_fun, [1 20]);
    new(1, 2*d+1) = real(df_rplus1);

    % Stopping criteria
    crit = max ( abs ( old(1,end) - new(1,end) ) ) ; solvec=new ...
        ; if any ( isnan ( solvec ) ) , break , end
    if ( crit < tol ) || ( iter ≥ maxit ) , break , end
    old = new;

end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Used to compute phi in the fzero-function
function phi = phi(y)
    y = abs(y);
    phi = psi(y) - log(y);
end
```

2.2 Weighted MLE

This section, similarly to the previous section, augments the codes for the MLE in section 1.2 by allowing for weights determined by the decay rate ρ. The relevant changes mainly affect line 36. To stay consistent with the implementation of the weighted MFF in section 2.1, we include the factor T in the calculation of the weights.

```
 1 % =======================MMF Function=================================
 2 % This function calculates the MLE as taken from the statistical ...
       inference book.
 3 % ==================================================================
 4
 5 function [param,stderr,iters,loglik,Varcov] = ...
       MVTestimation_3dweighted(x,rho)
 6 [nobs, d]=size(x);
 7
 8 %%%%%%%% k mu1 mu2 mu3 s11 s12 s22 s13 s33 s23
 9 bound.lo= [ 0.2 -1 -1 -1 0.01 -90 0.01 -90 0.01 -90];
10 bound.hi= [ 20 1 1 1 90 90 90 90 90 90];
11 bound.which=[ 1 0 0 0 1 1 1 1 1 1];
12 initvec =[1 -0.8 -0.2 -0.5 20 0 20 0 20 0];
13
14 maxiter=300; tol=1e-7; MaxFunEvals=length(initvec)*maxiter;
15 opts=optimset('Display','iter','Maxiter',maxiter,'TolFun',
16 tol,'TolX',tol,'MaxFunEvals',MaxFunEvals,'LargeScale','Off');
17 [pout,fval,¬,theoutput,¬,hess]= ...
18 fminunc(@(param) ...
       MVTloglik(param,x,bound,rho),einschrk(initvec,bound),opts);
19 V=inv(hess)/nobs; % Don't negate because we work with the ...
       negative of the loglik
20 [param,V]=einschrk(pout,bound,V); % transform and apply ∆ method ...
       to get V
21 param=param'; Varcov=V; stderr=sqrt(diag(V)); % Approximate ...
       standard errors
22 loglik=-fval*nobs; iters=theoutput.iterations;
23
24 function ll=MVTloglik(param,x,bound,rho)
25 if nargin<3, bound=0; end
26 if isstruct(bound), param=einschrk(real(param),bound,999); end
27 [nobs, d]=size(x); Sig=zeros(d,d); k=param(1); mu=param(2:d+1);
28 Sig(1,1)=param(d+2); Sig(2,2)=param(d+4); Sig(1,2)=param(d+3); ...
       Sig(2,1)=Sig(1,2);
29 Sig(3,3)=param(d+6); Sig(1,3)=param(d+5); Sig(3,1)=Sig(1,3); ...
       Sig(2,3)=param(d+7);
30 Sig(3,2)=Sig(2,3);
31
32 if min(eig(Sig))<1e-10, ll=1e5;
33 else
34     pdf=zeros(nobs,1);
35     for i=1:nobs, pdf(i) = mvtpdfmine(x(i,:),k,mu,Sig); end
36     T=length(x); tvec=(1:T); omega=(T-tvec+1).^(rho-1); ...
           w=T.*omega'/sum(omega);
37     llvec=log(pdf); ll = -mean(w.*llvec); if isinf(ll), ll=1e5; end
38 end
39
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 a correlation (not dispersion) matrix.
% So, just need to do the usual scale transform. 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);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 e r r o r s
    trans=2*pin .* (lo-hi ) ./ (1+pin .^2) .^2;
    d=(1-welche) + welche .* trans ; % either unity or Δ method .
    J=diag (d) ; Vout = J* Vin * J ;
end
```

2.3 Code for Output

First, we simulate a sequence of MVT random variables in order to obtain independently, both not identically distributed random variates from a MVT distribution, which exhibit time variation in the df parameter. We follow the suggestions in the assignment and create a grid of T values between 6 and 3 to obtain a sequence of decaying parameters for the df. Next, we simulate one 3-D MVT realization for every value of the df parameter, with the same true values for μ and for Σ. We then loop over values of $\rho = 0.2, 0.3, ..., 1$ and the number of repetitions for both methods of estimation, MMF and MLE. Finally, we plot the outcomes and report the computing time. The code is structured in a way, such that both methods are computed in one run. Since we also want to know the computing time, we simply comment out the respective lines for the other method.

```
% ===========================Exercise 2==============================
% We simulate 200 and 2000 random variates of a multivariate t ...
    distribution and estimte the parameters via MMF and MLE. The ...
    outputs are plotted as boxplots.
% ==================================================================


% Start Timer
tStart = cputime;


% Define variables
T = 200; d=3;
rhovec = 0.2:0.1:1; rholen = length(rhovec);
MMF1=[]; MLE1=[];
MMF2=[]; MLE2=[];
MMF3=[]; MLE3=[];
MMF4=[]; MLE4=[];
MMF5=[]; MLE5=[];
MMF6=[]; MLE6=[];
MMF7=[]; MLE7=[];
MMF8=[]; MLE8=[];
MMF9=[]; MLE9=[];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generate a random n x n matrix
sigma = -1 + 2.*rand(d,d);
% % construct a symmetric matrix
sigma = triu(sigma.',1) + tril(sigma);
% % Adding perturbations along the diagonal of a PSD matrix will ...
    make it PD
sigma = sigma + d*eye(d)
% Check if sigma is PD
%{
symm = issymmetric(sigma)
d = eig(sigma)
isposdef = all(d>0)
%}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


df_estimateMMF = [];
df_estimateMLE = [];

for i = 1:rholen, rho = rhovec(i)

    for rep = 1:1, rep
        % Create linspace for decay
        grid = linspace(6,3,T);

        % Create array for appending of decayed y
        y_decay = zeros(T,d);

        % Calculate y for decaying df
        for t=1:T, y_decay(t,:) = MVTrandom(sigma, grid(t), 1, ...
            [0 0 0]); end

        % Estimate parameters for MMF
        EM = mvtMMFweighted(y_decay, d, rho,  1e-3, 100);

        % Estimate parameters for MLE
        [param,stderr,iters,loglik,Varcov] = ...
            MVTestimation_3dweighted(y_decay, rho);

        % Append df parameters to array for plotting
        df_estimateMMF(end+1) = EM(1,7);
        df_estimateMLE(end+1) = param(1);
    end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Boxplots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For the MMF estimates. Substract the true value is not ...
    possible, because
% the parameters vary over the dataset.


MMF1 = df_estimateMMF(1:rep)';
MMF2 = df_estimateMMF(rep+1:2*rep)';
MMF3 = df_estimateMMF(2*rep+1:3*rep)';
MMF4 = df_estimateMMF(3*rep+1:4*rep)';
MMF5 = df_estimateMMF(4*rep+1:5*rep)';
MMF6 = df_estimateMMF(5*rep+1:6*rep)';
MMF7 = df_estimateMMF(6*rep+1:7*rep)';
MMF8 = df_estimateMMF(7*rep+1:8*rep)';
MMF9 = df_estimateMMF(8*rep+1:9*rep)';


% For the MLE estimates.
MLE1 = df_estimateMLE(1:rep)';
MLE2 = df_estimateMLE(rep+1:2*rep)';
MLE3 = df_estimateMLE(2*rep+1:3*rep)';
MLE4 = df_estimateMLE(3*rep+1:4*rep)';
MLE5 = df_estimateMLE(4*rep+1:5*rep)';
MLE6 = df_estimateMLE(5*rep+1:6*rep)';
MLE7 = df_estimateMLE(6*rep+1:7*rep)';
MLE8 = df_estimateMLE(7*rep+1:8*rep)';
MLE9 = df_estimateMLE(8*rep+1:9*rep)';



% Create the two boxplots as subplots

group1 = [ones(size(MMF1)); 2 * ones(size(MMF2)); 3 * ...
    ones(size(MMF3)); 4 * ones(size(MMF4));
     5 * ones(size(MMF5)); 6 * ones(size(MMF6)); 7 * ...
         ones(size(MMF7)); 8 * ones(size(MMF8));
     9 * ones(size(MMF9))];
%
```

```
group2 = [ones(size(MLE1)); 2*ones(size(MLE2)); ...
    3*ones(size(MLE3)); 4*ones(size(MLE4));
     5*ones(size(MLE5)); 6*ones(size(MLE6)); 7*ones(size(MLE7)); ...
         8*ones(size(MLE8));
     9*ones(size(MLE9))];


figure(1)
boxplot([MMF1; MMF2; MMF3; MMF4; MMF5; MMF6; MMF7; MMF8; MMF9], ...
    group1);
set(gca,'XTickLabel',{'MMF_0.2','MMF_0.3', ...
    'MMF_0.4','MMF_0.5','MMF_0.6','MMF_0.7','MMF_0.8',
'MMF_0.9','MMF_1'})
ylim([0 10])

figure(2)
boxplot([MLE1; MLE2; MLE3; MLE4; MLE5; MLE6; MLE7; MLE8; MLE9], ...
    group2);
set(gca,'XTickLabel',{'MLE_0.2','MLE_0.3', ...
    'MLE_0.4','MLE_0.5','MLE_0.6','MLE_0.7','MLE_0.8',
'MLE_0.9','MLE_1'})
ylim([0 10])


% End Timer
eEnd = cputime-tStart
```

2.4 Output

Before analysing the results, we compare the calculation speeds for the two methods in table 2. As expected, we clearly see that the brute force MLE takes much longer:

Table 2: Comparison of the Computation Time for One Run of the Weighted MMF vs. MLE Method.

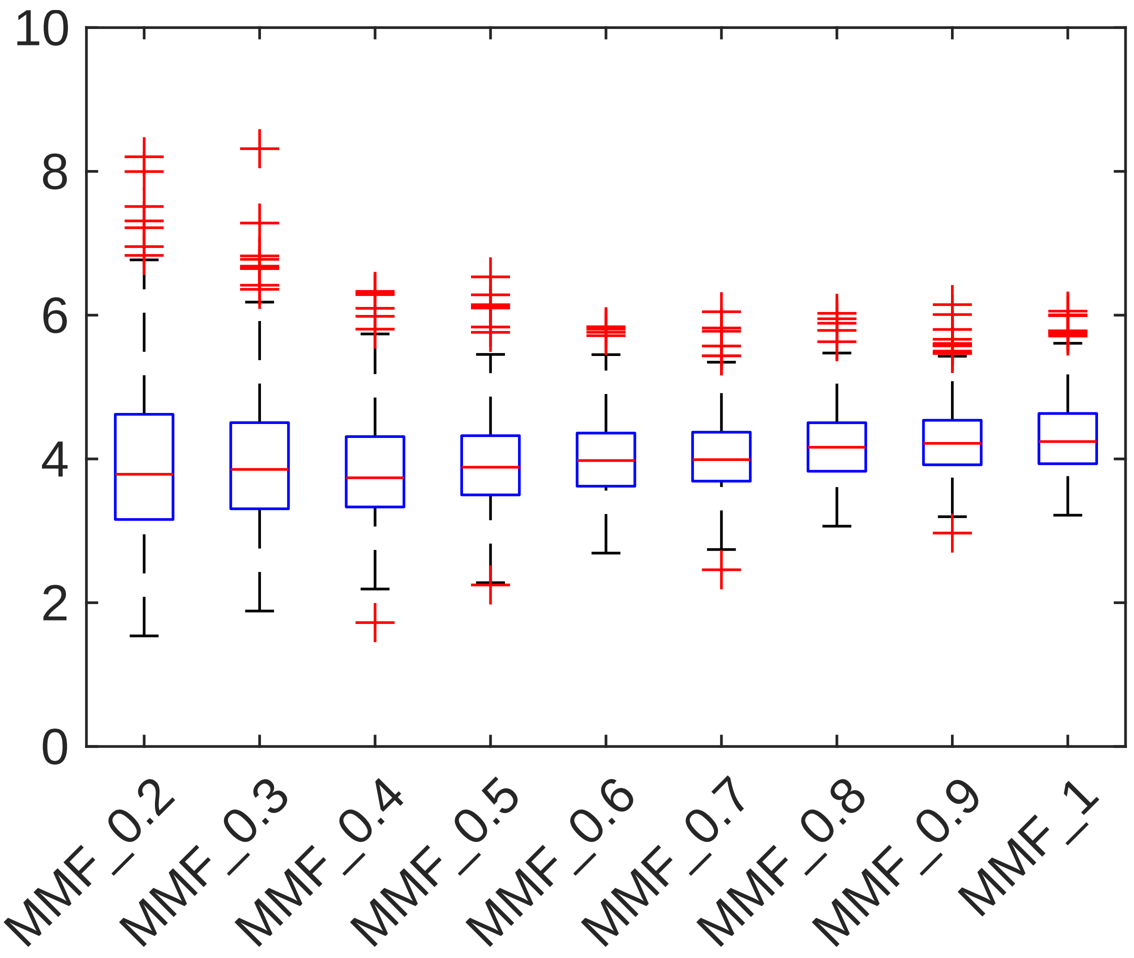
	Time (in sec.)	Normalized Time (in sec.)
Weighted MMF	0.0195	1
Weighted MLE	2.3302	119.2328

Note: The column 'Normalized Time' computes the relative times w.r.t. to the time for the weighted MMF.

Figures 10 and 11 show our results for the estimation of the df using the weighted MMF and MLE respectively, with 200 samples and 500 repetitions. We decided to use fewer samples, so that the variance, i.e. length of boxplots, would be greater and the differences thus more easily visible to the naked eye. The outputs are presented as a function of ρ, where $\rho < 1$ gives more ever more weight to recent observations, as described in section 2 This means that we effectively use "less" data the smaller ρ, resulting in higher variance, due to the down-weighting of older observations. This means, we should expect longer boxplots for smaller values of ρ. Using the same argument, the smaller ρ is, the closer we expect the centre of the boxplot to be to 3,

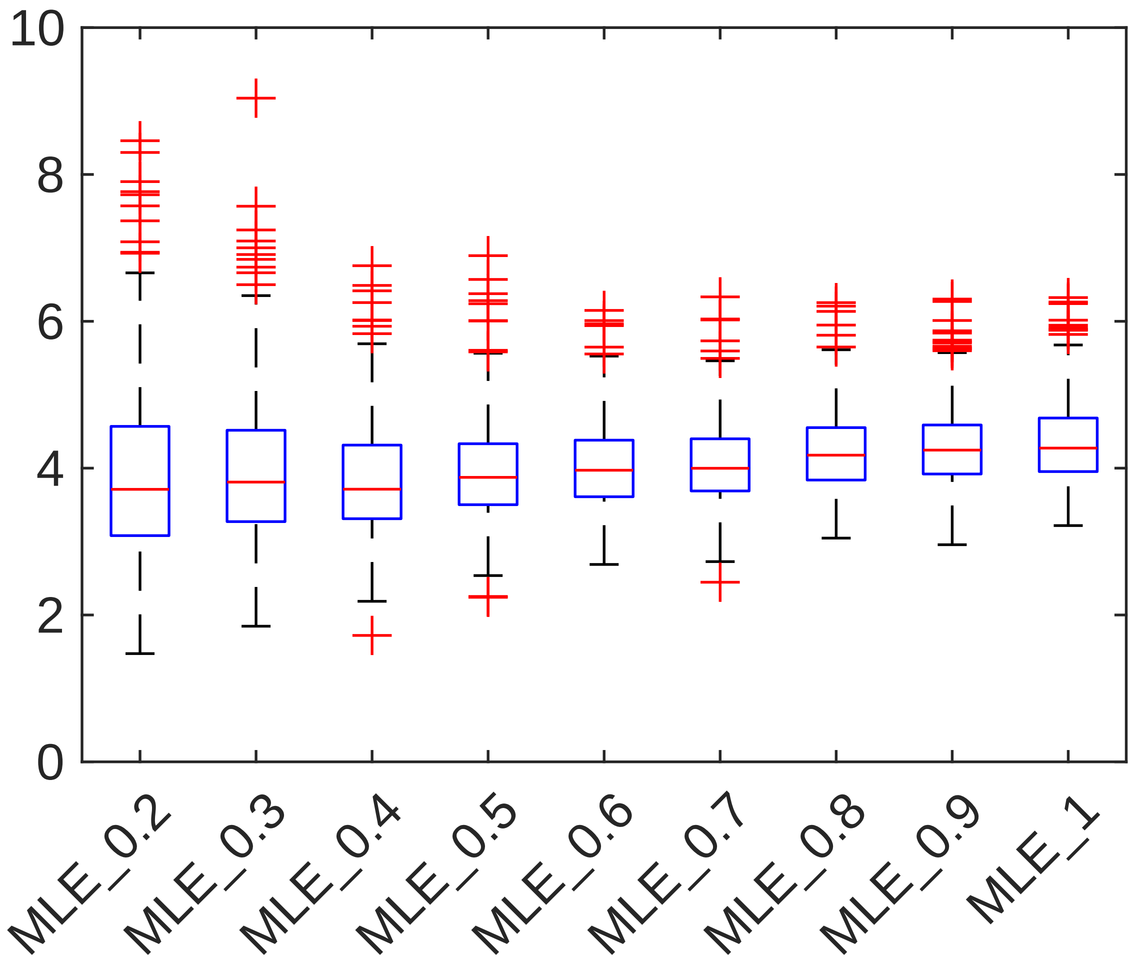
the final value of df or the value it converges to. Thus, the estimate would exhibit less bias. This stems from the fact that, if we were to use our results in forecasting as described in the introduction in section 2, we want the estimated df to be closer to the final value of 3, which is why we put more weight on more recent observations. Higher values of ρ, which results in more equal weights, would give a more biased estimate that is around the average of 6 and 3, so around 4.5.
Looking at the *variance* in both figures, we see that the length of the boxplots decreases with increasing ρ, as expected. In the first question, the length of the boxplots depended on the sample size, which is not varying in this question. As was the dominant picture in the first question, the two methods yield very similar results.
The *bias* for both methods behaves according to our expectations, namely increasing with ρ to a value around 4.5. The leftmost centre should be the least biased, i.e. closest to the final value 3. This is absolutely the case, with the centres in both cases being below 4. While the trend of the bias certainly behaves the way we expect it to, the pattern could definitely be clearer. We could not resist exploring this avenue and present the results for the same question, but with a higher sample size (T=500, instead of T=200) in the Appendix (3.1). We wanted to try even higher sample sizes, like $T = 2000$, however, our computers crashed twice (!) after 10 hours of calculations, so we stopped our attempts. For the sake of completeness, we still report the figure with 500 samples. We can somewhat see that the boxplots are a bit shorter in general, as expected when using larger sample sizes. The pattern of the bias is not exacerbated very much, but it must be when using even more samples. We remember that the differences in question 1 between 200 and 2000 samples were quite big already. This concludes our analysis.

Figure 10: Estimates of *df* for Different Values of ρ Using MMF.



Note: The figure shows the boxplots for the estimates of the degrees of freedom (df) for value of $\rho = 0.2, 0.3, ..., 1$ using the MMF method.

Figure 11: Estimates of df for Different Values of ρ Using MLE.

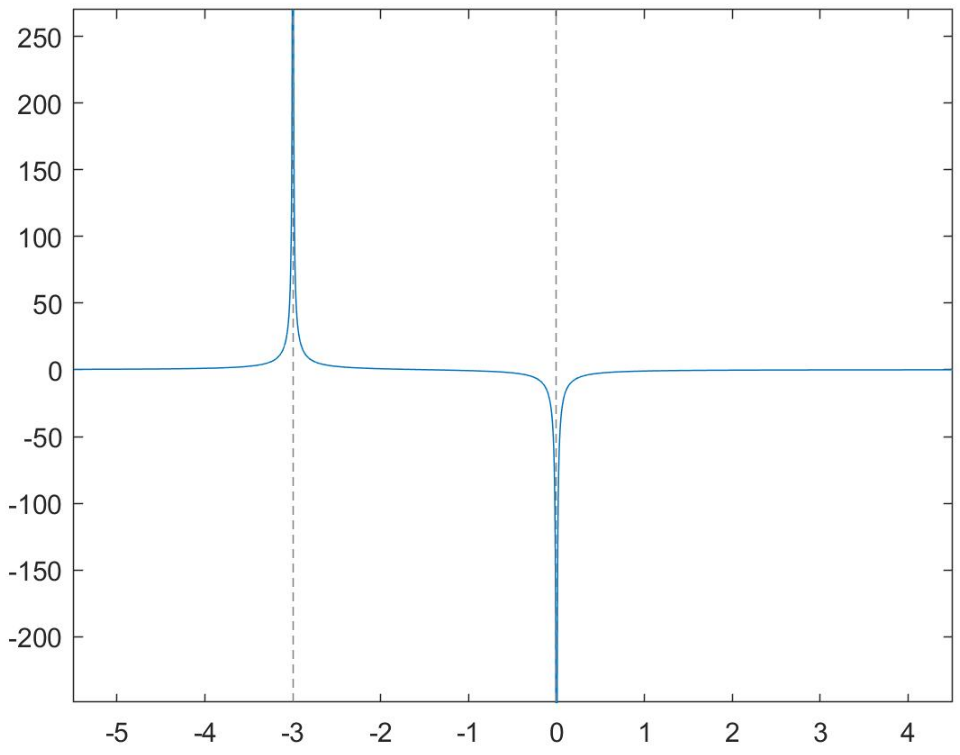


Note: The figure shows the boxplots for the estimates of the degrees of freedom (df) for value of $\rho = 0.2, 0.3, ..., 1$ using the MLE method.

3 Appendix

3.1 Plot of *df* Function

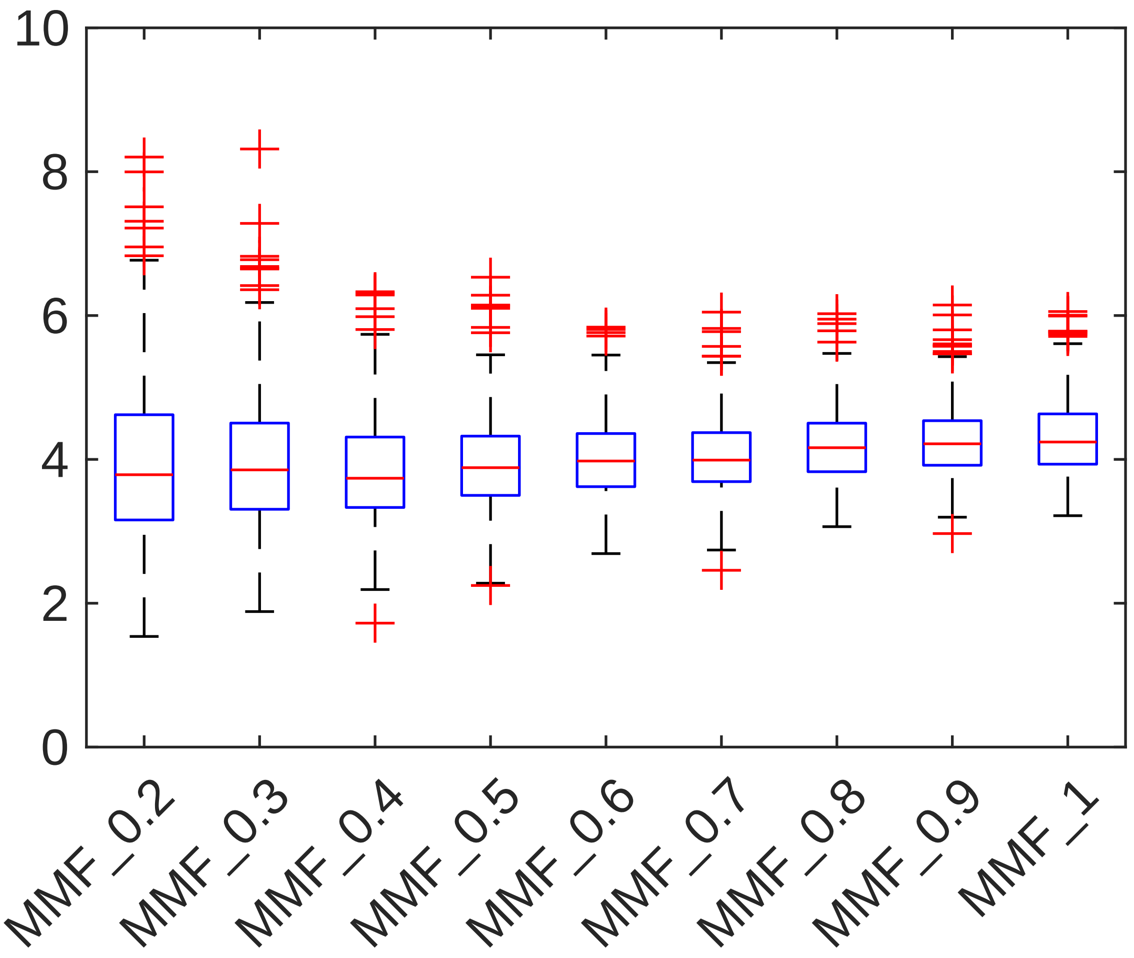
Figure 12: Plot for the *df* Function from the MMF Algorithm.



Note: This is just a plot of a single iteration in the MMF algorithm. The function does not change much in form over the iterations, so one snapshot should be enough.

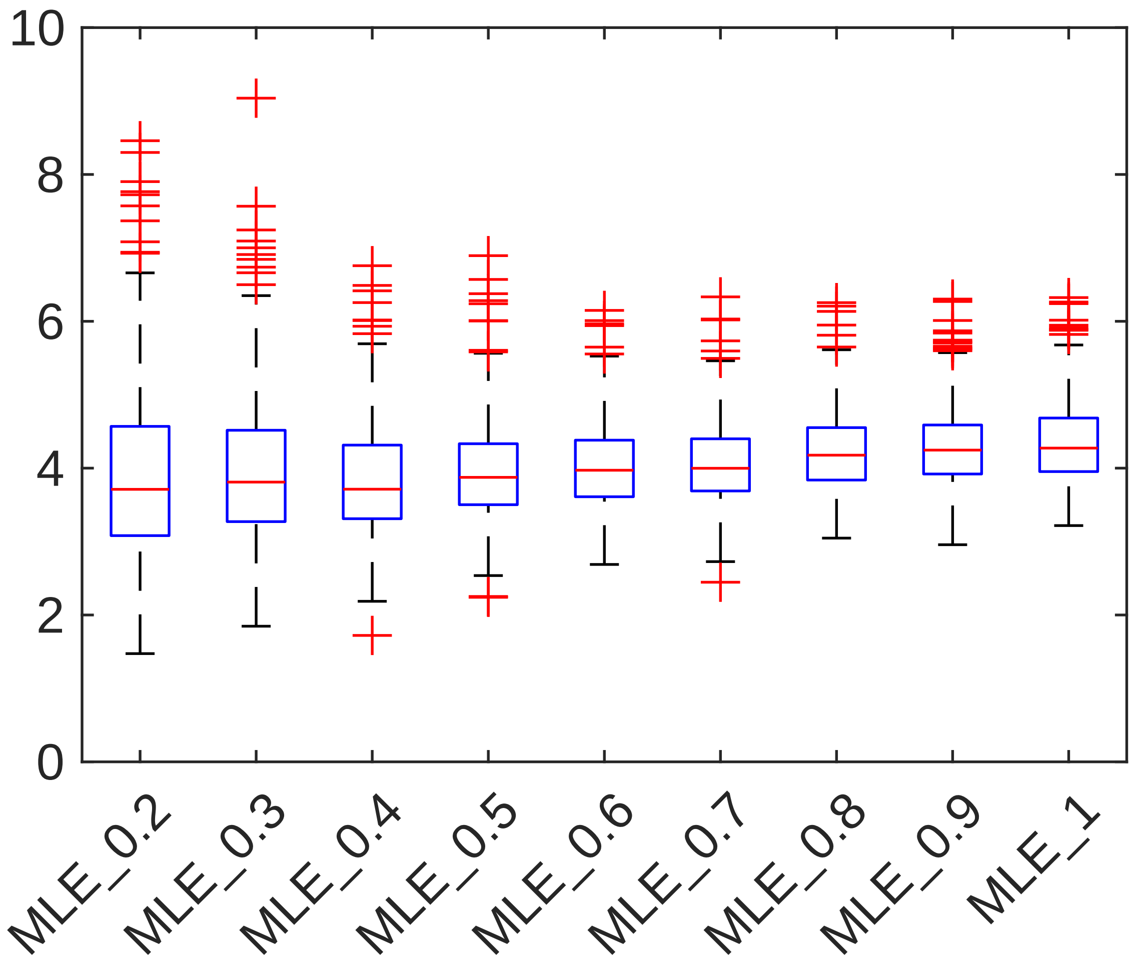
3.2 Weighted Estimates with Higher Sample Size

Figure 13: Estimates of *df* for Different Values of ρ Using MMF (T=500).



Note: The figure shows the boxplots for the estimates of the degrees of freedom (df) for value of $\rho = 0.2, 0.3, ..., 1$ using the MMF method.

Figure 14: Estimates of df for Different Values of ρ Using MLE (T=500).



Note: The figure shows the boxplots for the estimates of the degrees of freedom (df) for value of $\rho = 0.2, 0.3, ..., 1$ using the MLE method.