University of Zurich

Department of Banking and Finance

03SM22MO0103: Statistical Foundations for Finance (Mathematical and Computational Statistics with a View Towards Finance and Risk Management)

Take Home Assignment 2

Submitted by:
Azamat Zhaksylykov 23-745-664
Nikolai Kurlovich 23-743-412
Ishaan Bhondele 23-745-862

Contents

Friday 15th December, 2023

Problem I

In this section, we conducted a comparative analysis of computational efficiency, specifically examining the performance of the t-iteration program in contrast to the BFGS algorithm. The objective of those algorithms was to get parameter estimation with 4-digit accuracy while manipulating tolerance. The subsequent code snippet is dedicated to determining the appropriate tolerance for the t-iteration program to meet the specified accuracy requirements.

```
function [tol_titer] = tolerance_titer(tol, X)
    if nargin < 2
        X = trnd(3, 250, 1);
    end
    if nargin < 1
        tol = 0.0000001;
    end

    df_prev = 0;
    mu_prev = 0;
    c_prev = 0;

    while true
        [df, mu, c, ~] = titer(X,0,tol);
        tol = tol + 0.000005;

        if round(df,4)==round(df_prev,4) && round(mu,4)==round(mu_prev,4)
            && round(c,4)==round(c_prev,4)
            break;
        end

        df_prev = df;
        mu_prev = mu;
        c_prev = c;
    end
    tol_titer = round(tol, 1, significant);
end
```

The subsequent code snippet is dedicated to determining the appropriate tolerance for the BFGS algorithm to meet the specified accuracy requirements.

```
function [tol_BFGS] = tolerance_BFGS(tol, X)
    if nargin < 2
        X = trnd(3, 250, 1);
    end
    if nargin < 1
        tol = 0.0000001;
    end

    df_prev = 0;
    mu_prev = 0;
    c_prev = 0;
    while true
        MLE_current = tlikmax(X, tol);
        tol = tol + 0.000005;

        if round(df_prev,4)==round(MLE_current(1),4)&& round(mu_prev,4)==
            round(MLE_current(2),4) && round(c_prev,4)==round(MLE_current(3)
            ,4)
            break;
        end

```

```
        df_prev = MLE_current(1);
        mu_prev = MLE_current(2);
        c_prev = MLE_current(3);
    end

    tol_BFGS = round(tol, 1, significant);
end
```

The MATLAB 'timeit' function was applied to the following function to measure the total running time of the titer algorithm.

```
function titer_time(k,df,tol,T)
    rng(1,twister);
    for i = 1:k
        X = trnd(df,T,1);
        tol_titer = tolerance_titer(tol,X);
    end
end
```

The MATLAB 'timeit' function was applied to the following function to measure the total running time of the BFGS algorithm.

```
function BFGS_time(k,df,tol,T)
    rng(1,twister);
    for i = 1:k
        X = trnd(df,T,1);
        tol_BFGS = tolerance_BFGS(tol,X);
    end
end
```

Then the total time required for each of the two algorithms is calculated.

```
function [t1_time,t2_time]=BFGSvsTITER(k,df,tol,T)

    if nargin < 4; T = 250; end
    if nargin < 3; tol = 0.0000001; end
    if nargin < 2; df = 3; end
    if nargin < 1; k = 1000; end


    t1_time=timeit(@() titer_time(k,df,tol,T));
    t2_time=timeit(@() BFGS_time(k,df,tol,T));

end
```

Based on 100 replications, and a requested parameter tolerance of 4 significant digits (tolerance level ensuring 4 digit accuracy was equal to 0.00001), the total time required for t-iteration program is $\mathbf{0.11482}$ seconds, and for the BFGS algorithm is $\mathbf{1.66767}$ seconds. Titer is more than 10 times faster than brute force MLE maximization using BFGS.

Problem II

The conditional distribution of a bivariate Student t

Part 1

We make a program that inputs the parameters of a bivariate Student t (2 location terms, and a 2X2 dispersion matrix, and the df parameter), and computes the parameters of the conditional distribution of X2 given X1. The computation is done according to Ding's paper. In the code snippet below; we define the exact inputs and get the relevant outputs.

```
function [u,conditional_location,new_df] = ding_conditional_distribution(X1
    ,location_terms,dispersion_matrix,df)
%computes parameters of conditional distribution of X2 | X1 as modelled by
    ding in his paper
%   Detailed explanation goes here
disp_inv = inv(dispersion_matrix);
u = location_terms(2) + dispersion_matrix(2,1) * disp_inv(1,1) * (X1 -
    location_terms(1));
new_df = df + 1;

d1 = (X1 - location_terms(1))' * disp_inv(1,1) * (X1 - location_terms(1));
conditional_location = (df + d1) * (1/new_df) * (dispersion_matrix(2,2) -
    dispersion_matrix(2,1) * disp_inv(1,1) * dispersion_matrix(1,2));

end
```

Part 2

We code a program to simulate a bivariate Student T. The inputs are: 2 location terms, 2 X 2 dispersion matrix, df value, and the number of bivariate vectors to simulate. The formula we used was
if $Z = (Z_1, Z_2, ..., Z_d,)$
$X = (X_1, X_2, ..., X_d,)' = \mu + \sqrt{G}Z$ then, X follows allows a d-variate multivariate Student'st distribution with ν degrees of freedom, location parameter μ, and dispersion matrix $\sum$

```
function [X] = simulate_multivariate_student_T_V2(v,n, location_terms,
    dispersion_matrix)
%MULTIVARIATE_STUDENT_T
%   X = u + sqrt(G) * Z
rng('default')  % For reproducibility
Z = mvnrnd([0, 0],dispersion_matrix,n);
Y = gamrnd(v/2,1,[n 1]) / (v/2);  G=1./Y;
%Y = gamrnd(v/2,2/v,[n 1]);  G=1./Y;

X = location_terms + sqrt(G) .* Z;


end
```

We use the surf function to make a bivariate 3D plot of this estimate. A notable observation is that the graph gets smoother and less `pointy` as the DOF increases. We simulate 500 values, the DOF and dispersion matrix are nu and Rho respecively.

```
Rho = [1 0.3; 0.3 1];
nu = 2;

data = simulate_multivariate_student_T_V2(nu, 500, [0 0], Rho);

% Plotting the estimated density in 3D
x_min = min(data(:, 1));
```

```
x_max = max(data(:, 1));
y_min = min(data(:, 2));
y_max = max(data(:, 2));
[x_grid, y_grid] = meshgrid(linspace(x_min, x_max, 100), linspace(y_min,
    y_max, 100));

% Evaluate the kernel density estimate
kde = ksdensity(data, [x_grid(:), y_grid(:)]);
density = reshape(kde, size(x_grid));

% Plotting the density estimate in 3D
figure;
surf(x_grid, y_grid, density, 'EdgeColor', 'none');
xlabel('X axis');
ylabel('Y axis');
zlabel('Density');
title('3D Plot of Estimated Density with DOF: ' + string(nu));

colorbar;
```

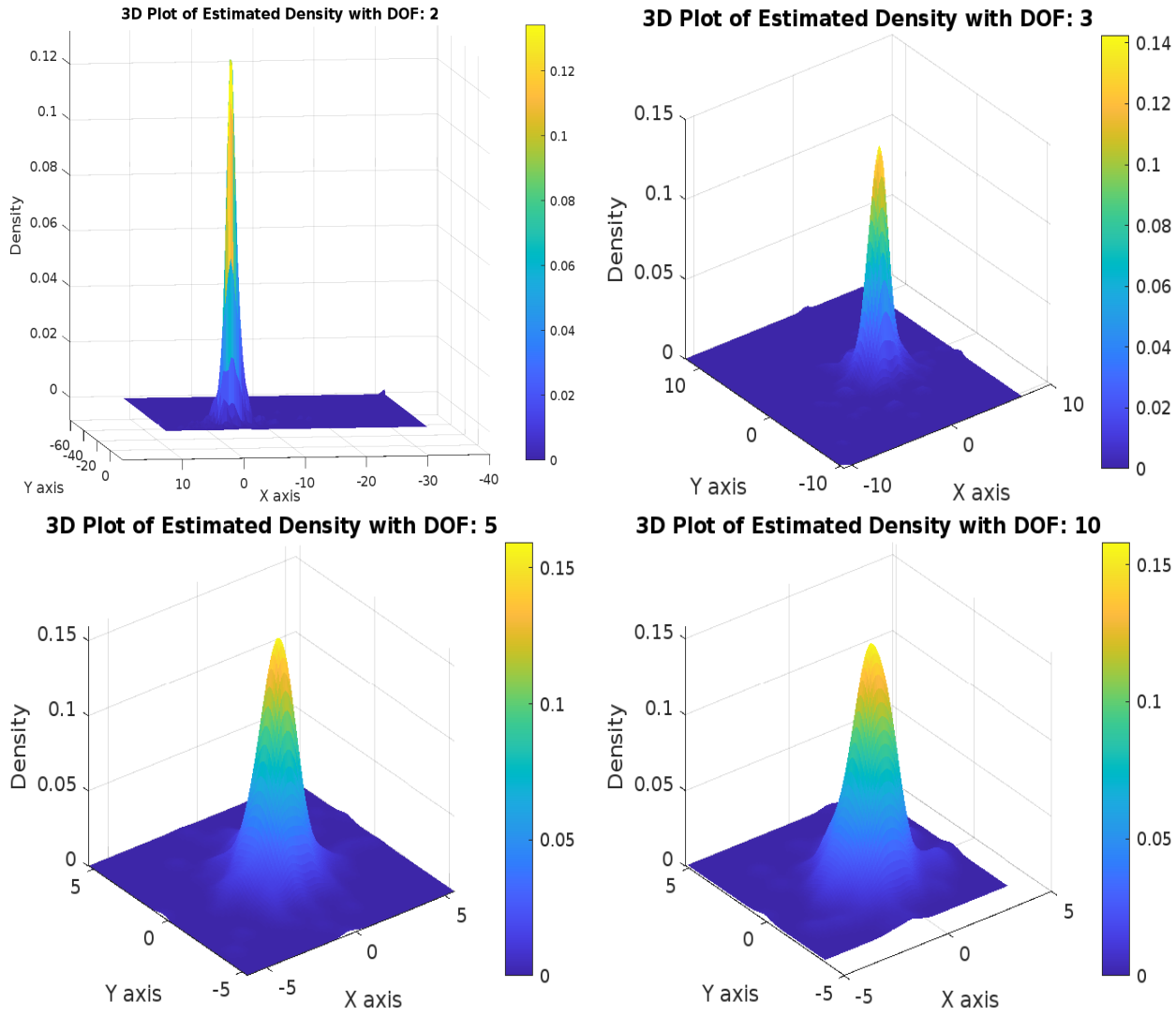


Figure 0.0.1: Bivariate Plots of different DOFs

Part 3

Bonus:In the multivariate Student t, even with identity dispersion matrix, the univariate margin distributions are *not* independent.
The Student's t-distribution is characterized by heavier tails compared to a normal distribution, especially for lower degrees of freedom. Even though the dispersion matrix is set to the identity matrix, the correlation between variables becomes more pronounced in the tails of the distribution.This means that extreme values in one variable are more likely to coincide with extreme values in another variable, leading to a lack of independence in the tails. We can see this more clearly in the formula $\frac{Z}{\sqrt{V}/\nu}$, where V is a chi squared distribution with ν Degrees of Freedom. For sufficiently small values of V; the distribution tends to take on larger values.
For this part, We define a function that inputs the n X 2 matrix of simulated bivariate Student t values, as well as a parameter, a scalar, called x2. There is a small value eps defined inside the function. The values that lie in the interval (x2-eps, x2 + eps) are extracted by simple if statements.

```
function [X1midX2] = conditional_subset(simulated_bivariate_values,x2)
eps = 0.0005;

%   splitting values
col1 = simulated_bivariate_values(:,1)';
col2 = simulated_bivariate_values(:,2)';
X1midX2 = [];
for i = 1:length(simulated_bivariate_values(:,1))

if x2 - eps < col1(i) && col1(i) < x2 + eps
    X1midX2(end + 1) = col1(i);
end

if x2 - eps < col2(i) && col2(i) < x2 + eps
   X1midX2(end + 1) = col2(i);
end
end
end
```

Part 4

Now we combine all of these defined functions. We Make plots to compare our estimated density with Ding's predicted density. A notable observation is that as we increase the DOF; The graphs start to look more like each other.
In the code; we can adjust the x2 scalar value and DOF value manually. We subsequently generate plots for DOF 5, 10, 15 with the X2 values -1, 0, 1, 2.

```
dispersion_matrix = [1 0.5; .5 2]; % defining a dispersion matrix
nu = 4; % defining a dof
location = [0 0]; % defining location vector

x2 = 0; % definite X2 to find a conditional subset
n = 1000000;
[X] = simulate_multivariate_student_T_V2(nu, n , location,
   dispersion_matrix);
[X1midX2] = conditional_subset(X, x2);
[f,xi] = ksdensity(X1midX2);

%generating theoretical values
[new_mean, new_scale, new_dof] = ding_conditional_distribution([0 0],
   dispersion_matrix, nu);

X1 = [-4:0.1:4];
a = (X1);
```

```
y1 = (tpdf(a, new_dof) - new_mean)/ new_scale;
figure;
plot(X1, y1) % plotting theoretical values
hold on
plot(xi, f) % plotting estimated values
[t,s] = title('Plot of Estimated & Theoretical Density','DOF: ' + string(nu
    ) + ', conditional subset scalar (x2): ' + string(x2), 'Color','blue');

legend(simulated, ding predicted)
hold off
```

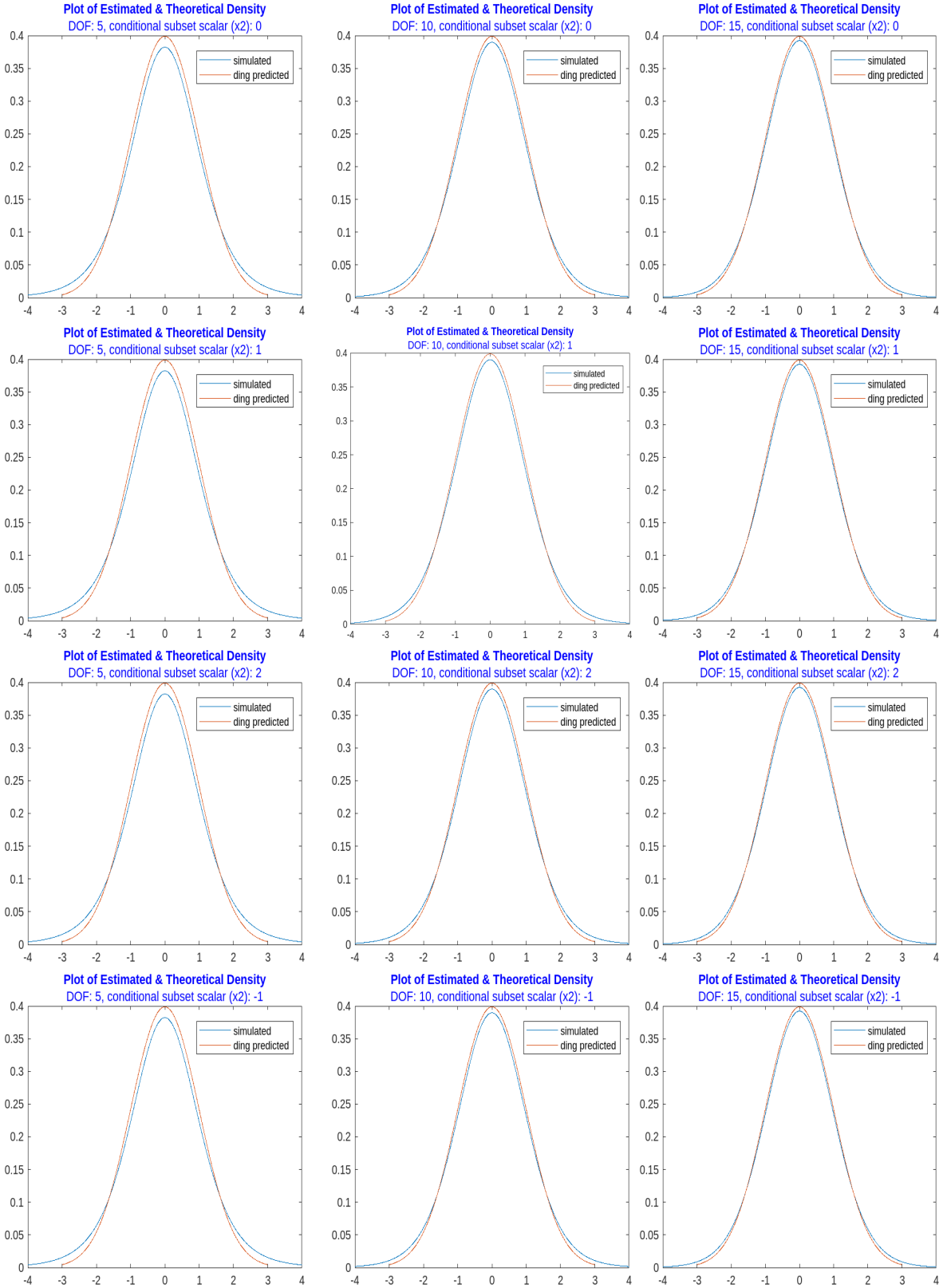


Figure 0.0.2: Plots of different var

Problem III

Part 1

In this section, we estimated the bivariate Student t with MLE. To do that MATLAB code from Prof. Paolella's time series book was implemented. The code snippet is given below.

```
function [param,stderr,iters,loglik,Varcov] = MVTestimation(x)
% param: (k, mu1, mu2, Sigma_11, Sigma_12, Sigma_22)
[nobs d]=size(x); if d~=2, error('not done yet, use EM'), end

if d==2
%%%%%%%% k mu1 mu2 s11 s12 s22
bound.lo= [ 0.2 -1 -1 0.01 -90 0.01];
bound.hi= [ 20 1 1 90 90 90];
bound.which=[ 1 0 0 1 1 1];
initvec =[2 -0.8 -0.2 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),
    einschrk(initvec,bound),opts);
V=inv(hess)/nobs; % Don't negate because we work with the negative of the
    loglik
[param,V]=einschrk(pout,bound,V); % transform and apply delta method to
    get V
param=param'; Varcov=V; stderr=sqrt(diag(V)); % Approximate standard
    errors
loglik=-fval*nobs; iters=theoutput.iterations;

function ll=MVTloglik(param,x,bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param),bound,999); end
[nobs d]=size(x); Sig=zeros(d,d); k=param(1); mu=param(2:3); % Assume d=2
Sig(1,1)=param(4); Sig(2,2)=param(6); Sig(1,2)=param(5); Sig(2,1)=Sig(1,2)
    ;
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
```

Subsequently, a bivariate Student t dataset was generated, and the accuracy of the parameter estimation algorithm was tested using simulated data. The validation process involved constructing a 6-element boxplot, where each element represents the difference estimated parameter and its true value. The following code was used to plot the boxplot.

```
function box_plot()
    param=zeros(100,6)
    for k=1:100
        Sigma= [1 0.8; 0.8 1];
        df=2;
        R=mvtrnd(Sigma,df,100);
        x=MVTestimation(R);
        param(k,1)=x(1)-df;
        param(k,2)=x(2);
        param(k,3)=x(3);
        param(k,4)=x(4)-Sigma(1,1);
        param(k,5)=x(5)-Sigma(1,2);
        param(k,6)=x(6)-Sigma(2,2);
    end
    boxplot(param,'Labels',{'df','mu_1','mu_2','sigma_11','sigma_12','
```

```
        sigma_22'});
    hold on;
    yline(0,'Color','r');
end
```

Following that, the presented boxplot illustrates the close alignment between the estimated parameters and their true values.

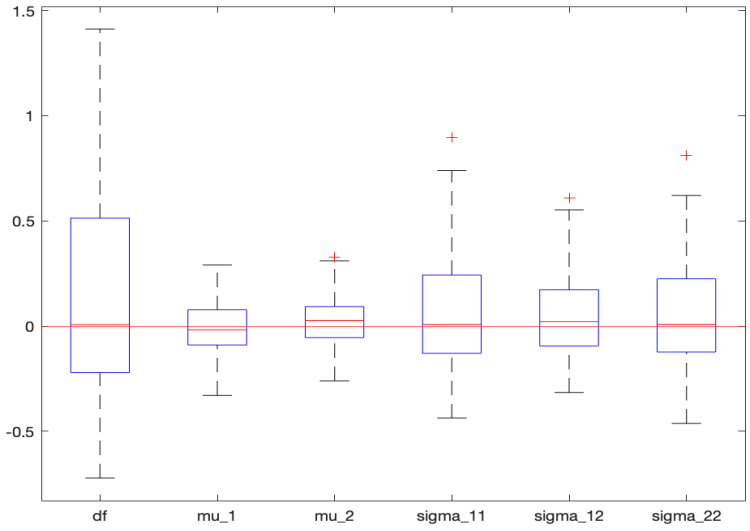


Figure 0.0.3: Estimated parameters minus their true values boxplot

Part 2

This section deals with fitting data to the bivariate IID model. We compare the estimated density (By using a kernel density function) and the theoretical density.
The theoretical density is calculated by using the inversion formula: $f_X(x) = \frac{1}{2\pi}\int_{-\infty}^{-\infty} e^{-itx}\phi_X(t)\,dt$
For a given Degree of Freedom (v), Sigma (σ), Mu (μ), we computed the characteristic function using the formula as follows:

$$\frac{K_{\frac{v}{2}}(v^{\frac{1}{2}} \mid t \mid \kappa)^{\frac{v}{2}}(v^{\frac{1}{2}} \mid t \mid \kappa)^{\frac{1}{2}}}{\gamma(v/2)2^{\frac{v}{2}-1}} e^{it\mu_s}$$

As you can see in the function below, we can input the subsequent parameters. The output is a set of x values and their respective y values. When we plot these, we get a distribution.

```
%calculate student T by characteristic function inversion formula
function [X ,f] = pdf_from_cf(v, min_x, max_x, n, sigma, a, mu)

f = zeros(n, 1);
X = linspace(min_x, max_x, n);
for i = 1:n
    x = X(i);
    %calculating characteristic function for Student T: Given DOF(v)
    % from Time Series book c.27
    kappa = sqrt(a' * sigma * a);
    mu_s = sum(a' * mu);
    phi = @(t) exp(1i * t * mu_s) .* (besselk(v/2, sqrt(v) * abs(t) * kappa
        ) .* ((sqrt(v) .* abs(t) * kappa).^(v/2)))/(2.^(v/2 - 1) .* gamma(v
        /2)) ;
    %defining function to intergrate for inversion theorem
```

```
    pdf_t = @(t, x) (1/(2 .* pi)) .* exp((-1i) .* t .* x ) .* phi(t);
    %integrating t from -inf to inf
    f(i) = integral(@(t) pdf_t(t, x), -inf,inf);
end

end
```

To get an estimated density we simulate random numbers and plot the estimated density using a kernel density function. As you can see in the code below; we have defined abstract parameters like the dispersion matrix, DOF and values for a. We use some random weights that add up to 1 for a.

```
%% Define parameters
mu = [0; 0];        % mean vector
sigma = [1 0.3; 0.3 2];     % covariance matrix (identity matrix for
    simplicity)
v = 3;              % degrees of freedom
a = [.3; .7];         % parameter vector

[xi , f] = pdf_from_cf(v, -4, 4, 10000, sigma, a, mu);

R = mvnrnd(mu,sigma,100000);
X = zeros(1, length(R(:,1)));
for i = 1: length(R(:,1))
X(i) =  a(1) * R(i,1) + a(2) * R(i,2);
end

[x_val, y_val] = ksdensity(X);


% Plot the 2D graph
figure;
plot(y_val, x_val, 'LineWidth', 2);
hold on
plot(xi, f)
title('Student''s t-Distribution PDF');
xlabel('x');
ylabel('PDF Value');
grid on;
hold off
```

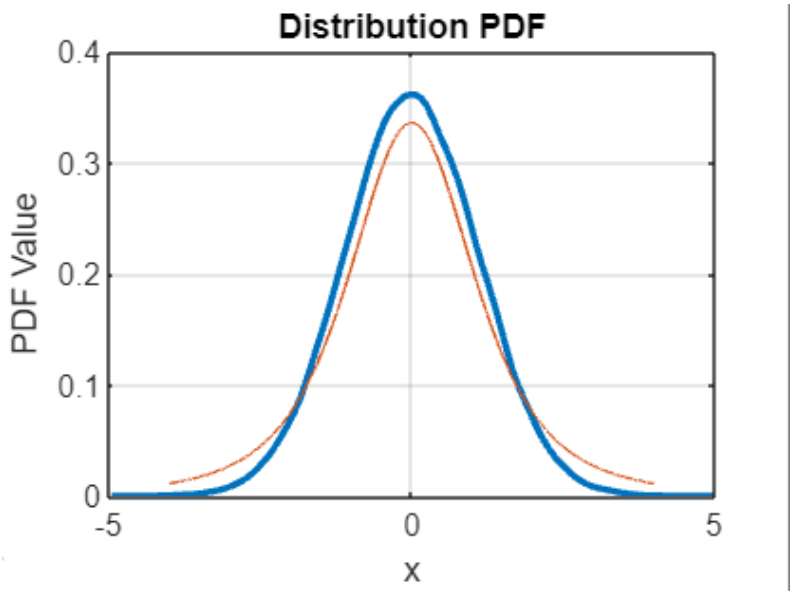


Figure 0.0.4: Estimated Density vs Theoretical Density

Problem IV

Part 1

In the first part we are interested in writing a program to simulate a 2-component, bivariate discrete mixture of Laplace-distributed random variables. We break this into two steps - first we explain how this can be accomplished and then we go to illustrate how this can be used in an attempt to model financial asset's returns.

Code Walkthrough

We first would like to be able to sample a bivariate Laplace random variable. Once this is accomplished, it won't be difficult to sample a mixture of those.

For that, we use the content presented on page 650 of the Time Series written by our professor, Dr. Paollela. The most important result if the following one: suppose we have $X \sim Lap_d(\mu, \Sigma, b)$. Then $X \mid G = g \sim N_d(\mu, g\Sigma)$ where $G \sim Gam(b, 1)$, where in both cases d is supposed to stand for dimensions, which for this problem will be taken to be equal to 2. This gives us a very efficient an simple way to sample bivariate Laplace-distributed random variables using functions already (efficiently) implemented in Matlab, namely those for sampling gamma-distributed and normal-distributed random variables. We first sample from gamma and then plug in the obtained value to obtain our Laplace distributed variable.

```
function y = randlap(n, mu, sigma, b)
    g = gamrnd(b, 1, [1 n]);
    y = zeros(n, 2);
    for i=1:n
        y(i, :) = mvnrnd(mu, g(i) * sigma);
    end
end
```

Note that we sample gamma-distrubuted random variable outside of the for-loop since this is further optimized by Matlab using vectorized operations. Having obtained this vector of gamma-distrubted random variables, we run a for-loop to create n-$Lap_2(\mu, \Sigma, b)$ random-variables.

The arguments for this program are obvious - n is the number of realizations needed and mu, sigma, b - correspond to the three parameters governing $Lap_2(\mu, \Sigma, b)$ distribution.

Now that we are able to sample bivariate Laplace random variables, it is time to use this in order to sample from a mixture of Laplace random variables. For this problem, we only handle a case when k - the number of components is equal to two. in this case, there is a single parameter w and weights of the components are given by w and $1 - w$. We take w to govern the weight put on the first component, in particular, meaning that for the second component the weight is given by $1 - w$. Before we present the actual code, we need to quickly mention a few helper functions, which we will need along with what we already came up with.

```
function C = randmultinomial(p)
    pp=cumsum(p);
    u=rand;
    C=1+sum(u>pp);
end
```

This function is extracted directly from the Prof. Dr. Paollela's textbook and its mechanic is simple - it takes a vector of probabilities p where p_i describes the probability of selecting i-th's component and outputs the number of the component which was randomly chosen. This is exactly what we need to be able to sample from a mixture. With, that, the following code accomplishes what we are after:

```
function y = sampleLapMix (n, lambda, mu1, sigma1, b1, mu2, sigma2, b2)
    pick = zeros(n, 2);
    % lambda - probability of picking the first comp.
    weights = [lambda 1-lambda];
    for i=1:n
        c = randmultinomial(weights);
        pick (i, c) = 1;
    end
```

```
9
10      Y = zeros(n, 2, 2);
11      disp(b1);
12      Y(:, 1, :) = randlap(n, mu1, sigma1, b1);
13      Y(:, 2, :) = randlap(n, mu2, sigma2, b2);
14
15      Y=pick.*Y;
16      y=squeeze(sum(Y, 2));
17  end
```

Let us give a quick overview of what's happening here. First, using randmultinomial function described above we create a $n \times 2$ matrix where $pick_i$ desribes which component was chosen - if the first is chosen, $pick_i = [1, 0]$ and as one might expect, when the second is chosen, $pick_i = [0, 1]$. Then we create matrix Y which stores n realizations of both the first and the second Laplace-distributed components. Using element-wise multiplication of $pick$ and Y, we turn unselected realizations to zero. Note that Y has dimensions $n \times k \times d$, where n - number of observations, k- number of components in the mixture and d - number of dimensions of the Laplace distributed random variables. Thus, after picking, we must perform summation along the second dimension, since in our case it is responsible for the division along components.

Now we have the code which allows to sample a 2-components mixture of bivariate random variables.

Application Example

Now we would like to use this code in order to model some financial asset's returns. We assume that we have two assets. The task boils down to selecting a reasonable set of parameters. We would like first component to capture a more standard, non-extreme behaviour and the second component will be responsible for describing a more unusual, extreme events. Usually, such events are negative ones. From this formulation, it is almost immediate that we may wanna set w (the weight of the first component) to some high value, 0.9 - 0.95.

Also, we will assume that all returns are in percentages and are annual. Thus, for a more generally good market conditions, we would take $\mu_1 = [7, 4]$, i.e. our assets generate 7% and 4% annual returns. For the bad times, let us take $\mu_2 = [-6, -3]$. If one looks at expressions for the standard deviations of the gamma distribution as well as how g is used in the expression to generate laplace distributed random variables, it becomes clear that with an increase in b, $Lap_b(\mu, \Sigma, b)$ sees an increase in its variance. We already made our first financial asset to possess higher returns on average. However, general finance theory predicts us that oftentimes higher returns come from a higher exposure to systematic risk, i.e. there is higher risk associated with this asset. To model this, we will take $b_1 = 3$ and $b_2 = 2$.

Now it remains to specify Σ_1 and Σ_2. In both we will take the variance of the first component to be larger than that of the second. What's crucial is the covariance term. In our task description, we have *the Σ_2 matrix has much larger diagonal elements than Σ_1* and this is true. Volatility and negative movements in the markets tend to be highly correlated. What is true to that is also that correlation across assets tends to significantly rise too - when things go downhill, really everything goes downhill. Thus, we also will take covariance elements in the second dispersion matrix to be much larger than that of the first. Thus, we suggest the following values:

$$\Sigma_1 = \begin{bmatrix} 4 & 0.36 \\ 0.36 & 2 \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 6 & 1 \\ 1 & 2 \end{bmatrix}$$

This is translated into matlab code as follows:

```
lambda = 0.9;
mu1 = [7 4];
mu2 = [-6 -3];
sigma1 = [4, 0.36; 0.36, 2];
sigma2 = [6, 1; 1, 1];
b1 = 3;
b2 = 2;

outcomes = sampleLapMix(10000, lambda, mu1, sigma1, b1, mu2, sigma2, b2);
```

We can create several visualizations of this sample (the code of visualization will be omitted so that we don't fill this report with clatter too much). We first start with heatmaps:

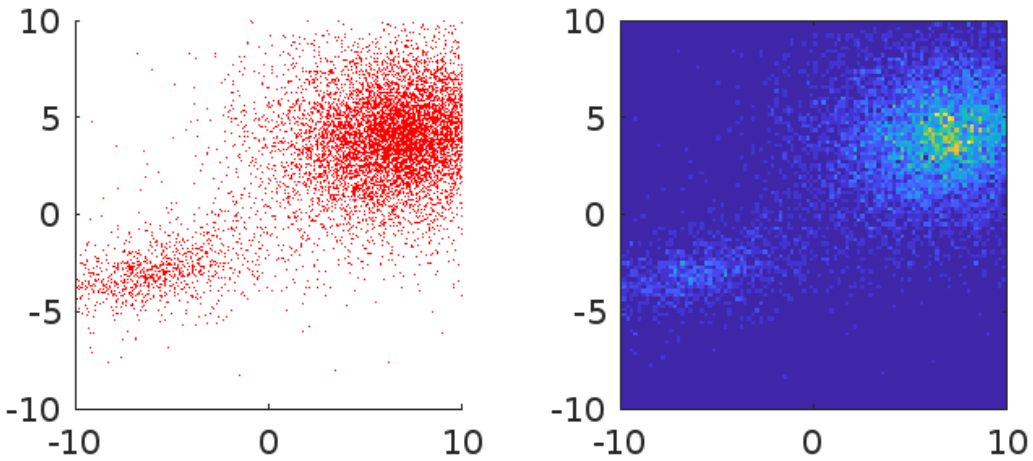


Figure 0.0.5: Two heatmaps showing the sample generated.

Both show the exact same thing, but just with a bit different coloring scheme. This heatmap suggests the our simulation procedure is more or less correct. First, we observe a much greater share of realizations to be clustered around μ_1, meaning that our w parameter indeed correctly affects our simulations proceduare and things are as we expect them to be. Also, both clusters indeed happen to be where we expect them to be - around μ_1 and μ_2. Also, as we predicted, the first component has a higher variance (which we plot on x-axis). Overall, this figure is very convincing - points really draw the shape that we expected them to draw.

Since the task also asked for a 3d plot, we will produce it too. However, now it is a bit more challenging to observe relations that we clearly saw using our heatmaps above:

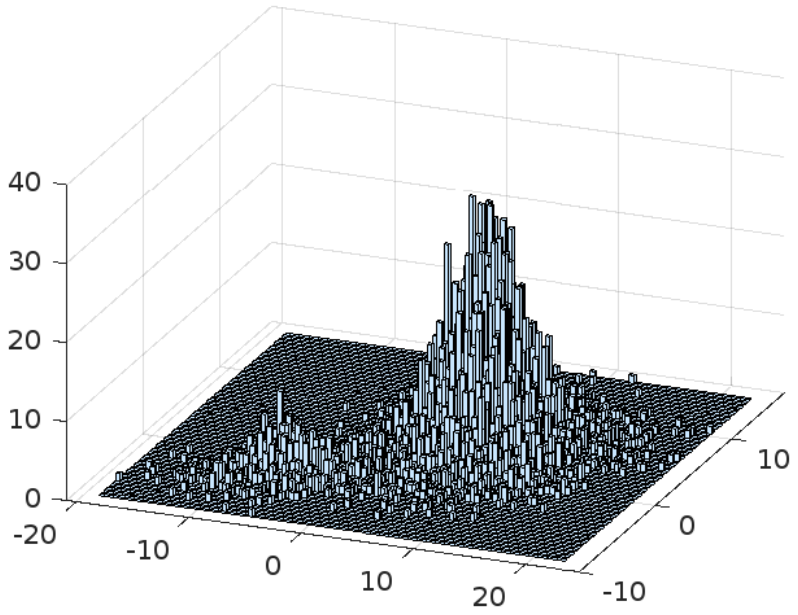


Figure 0.0.6: Histogram of returns

Note that the z-axis here is not supposed to represent proportion and therefore it shouldn't be interpreted as an estimate of empirical distribution function. Here, we focus solely on the visual image of the plot obtained. Still, it is quite clear that we have more realizations coming from the first component. It is also visible where the two clusters are and they are there we expected them to be. Finally, variance in both directions and components also seems to be in line with our expectations.

Part 2

N.B.: Everything related to the non-central t-distribution is put in part 3.

Code Walkthrough

In this problem we would like to make a program to obtain MLE estimates of a 2-mixture of bivaraite Laplace-distributed random variables. For that, we need to be able to compute its log-likelihood. Then, once this is done, we will plug this function into matlab optimizer to get our estimates. First, we start with the pdf.

```
function y = mulLapPdf (x, mu, sigma, b)
    % compute pdf of a bivariate Laplace distribution
    d = length(x);
    m = (x - mu)' / sigma*(x - mu);
    coef = (2 / gamma(b)) / (sqrt(det(sigma)) * (2 * pi)^(d/2));
    core = (m/2)^(b/2 - d/4) * besselk(b - d/2, sqrt(2*m));
    y = coef .* core;
end
```

This code directly implements equation 14.31 of the professor's textbook. important thing here is that we use / operator instead of *inv() as it is numerically more stable and efficient. Also, due to issues in naming conventions, it is not obvious whether the function we use for the modified Bessel function of the third kind is really what we think it is. From the textbook, we want to use a matlab function to evaluate the value of

$$K_v(x) = \frac{1}{2}\int_0^\infty u^{v-1}\exp\left[-\frac{x}{2}\left(\frac{1}{u}+u\right)\right]\mathrm{d}u$$

We need to manually check that this integral and the Matlab function we use coincide. Integrals can be evaluated numerically without much trouble, so we also make the following small program to verify that everything is okay:

```
bessels = zeros(10, 1);
integs = zeros(10, 1);

for x = 1:10
    f = @(u) u.^(2-1) .* exp(-(x/2).*(u + 1 ./ u));
    integs(x) = 0.5 .* integral(f, 0, 10000);
    bessels(x) = besselk(-2, x);
end

disp(integs');
disp(bessels');
```

Here nu is taken to be equal to 2. Checking for just one value is sufficient - it is highly unlikely that we randomly stumble upon a magic value of ν such that testing only with it we would make a wrong conclusion. The output we get is as follows:

```
    1.6248    0.2538    0.0615    0.0174    0.0053    0.0017    0.0006
       0.0002    0.0001    0.0000

    1.6248    0.2538    0.0615    0.0174    0.0053    0.0017    0.0006
       0.0002    0.0001    0.0000
```

We see, that both are exactly equal meaning that we are correct in our use of Besselk matlab function.

Now we use this to write a function that returns pdf of a mixture - this is extremely easy and is solved with just one line of code using the definition. We get:

```
function y = mixMulLapPdf (x, lambda, mu1, sigma1, b1, mu2, sigma2, b2)
    %fprintf(» y = lambda .* mulLapPdf(x, mu1, sigma1, b1) + ...+ (1 -
       lambda) .* mulLapPdf(x, mu2, sigma2, b2);end
```

Having done that, we now move to implementation an objective function. Since matalb optimizer minimizes a given function and we want to maximize, we will set the objective function to be equal to the negative log-likelihood, so that minimizing it we maximize log-likelihood. Also, its input is the parameter vector, so we need to make appropriate transformations to turn it into the format that our pdf function written above can take. We make the following code:

```
function ll = MLloglik (param, x, bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param), bound, 999); end
[nobs, d] = size(x);
sigma1 = zeros(2, 2); sigma2 = zeros(2, 2); lambda=param(1); mu1=param(2:3)
    ;
sigma1(1, 1) = param(4); sigma1(2,2) = param(6); sigma1(1, 2) = param(5);
sigma1(2,1) = sigma1(1, 2); b1 = param(7); mu2 = param(8:9); sigma2(1, 1) =
     param(10); sigma2(2,2) = param(12);
sigma2(1,2) = param(11); sigma2(2,1) = sigma2(1,2); b2 = param(13);
% disp(b1);
if (min(eig(sigma1)) | min(eig(sigma2))) < 1e-10, ll = 1e5; else
    pdf = zeros(nobs, 1);
for i=1 : nobs, pdf(i) = mixMulLapPdf(x(i, :)', lambda, mu1', sigma1, b1,
    mu2', sigma2, b2);end
    llvec=log(pdf); ll=-mean(llvec); if isinf(ll), ll=1e5; end
end

end
```

We have 13 parameters in total - 6 for each Laplace component and 1 for the weight. Now it's time to write an actual program which performs the optimization.

```
function [param,stderr,iters,loglik,Varcov] = MLestimation(x)
% param:  (lambda, mu1_1, mu1_2, sigma1_11, sigma1_12, Sigma1_22, b1, mu2_1
    , mu2_2, sigma2_11, sigma2_12, Sigma2_22, b2)

[nobs d]=size(x);
mn = mean(x);
mn1 = mn(1);
mn2 = mn(2);
sd = mean(std(x));
%%%%%%%%       lamb mu11 mu12  s111   s112  s122  b1 mu21 mu22 s211 s212
    s222   b2
bound.lo    = [  -1   -1   -1  0.01    -10 0.01  0.1   -1   -1 0.01  -10
    0.01  0.1];
bound.hi    = [   1    1    1    10     10   10   10    1    1   10   10
    10   10];
bound.which = [   1    0    0     1      1    1    1    0    0    1    1
       1    1];
initvec     = [ 0.5  mn1  mn2     1      0    1   sd  mn1  mn2    1    0
       1   sd];
maxiter=500; tol=1e-6; MaxFunEvals=length(initvec)*maxiter;
opts=optimset('Display','iter','Maxiter',maxiter,'TolFun',tol,'TolX',tol,
    ...
'MaxFunEvals',MaxFunEvals,'LargeScale','Off');
[pout,fval,~,theoutput,~,hess] = fminunc(@(param) MLloglik(param, x, bound)
    , einschrk(initvec,bound), opts);
V=inv(hess)/nobs;
%Don't negate because we work with the negative of the loglik
[param,V]=einschrk(pout,bound,V);
% transform and apply delta method to get V
param=param'; Varcov=V; stderr=sqrt(diag(V));
% Approximate standard errors
```

```
loglik=-fval*nobs; iters=theoutput.iterations;
end
```

Note that we are quite smart with the initial values here. From the sample, it costs almost nothing to compute its mean and standard deviation. We use those values to have a better initial vector for our optimization. However, since we are dealing with the mixture, this heavily inflates the sample covariance matrix and its use has little value for us. For that reason for covariance matrix we stick with a fairly conservative initial value - a simple identity matrix. Also note that we specifiy relevant constraints for each parameter. Means do not have those. Variances must remain positive and b should also be only positive. To ensure, that these constraints are always met, we utilize the einschrk function which is fully and without any modification is taken from our professor's textbook.

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

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

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

Example Run

Now having done that, we are ready to run the code and see how it performs. We will use our function for sampling that we wrote in subproblem 1 to generate a sample and then will try to obtain its MLE estimate. In the problem definition, we were asked to run this with a sample of size 10000. We would like to see, however, how our estimates would fluctuate with a change in this value. Thus, we run our simulations for a sample size of 5, 10, 25 thousand realization. We use the following simple code (for the case when $n = 25000$) to do that:

```
clear;
rng(42);
sample = sampleLapMix(25000, 0.3, [4; 5], [2 0.5; 0.5 2], 2, [-3; -4], [1
    0; 0 1], 6);

[param,stderr,iters,loglik,Varcov] = MLestimation(sample);
disp(param');
disp(stderr');
```

As you see, for the sake of this simulation, our choice of parameters for this study is the following: $w = 0.3$, $\mu_1 = [4, 5]$, $\mu_2 = [-4, -4]$, $\Sigma_1 = [2, 0.5; 0.5, 2]$, $\Sigma_2 = [1, 0; 0, 1]$, $b_1 = 2$, $b_2 = 6$.

The following table reports this result. Note that the notation μ_2^1 is supposed to mean the second component of the mean vector belonging to the first mixture component. Also, sometimes running the code we get results where the labels are switched - i.e. instead of $w = 0.3$ we get $w = 0.7$ and first 6 parameter values describe the second component. This is completely reasonable since labels can always be switched and we need to keep this in mind. The following results are reported so that w is always closer to 0.3.

	$n = 5000$		$n = 10000$		$n = 25000$		
Parameter	Est.	SE	Est.	SE	Est.	Se	True Value
$\lambda(\equiv w)$	0.3053	0.0067	0.3053	0.0048	0.2977	0.0030	0.3
μ_1^1	4.0945	0.0472	4.0050	0.0325	4.0165	0.0207	4
μ_2^1	5.1072	0.0462	4.9740	0.0337	4.9964	0.0215	5
Σ_{11}^1	1.9930	0.2635	2.1237	0.1946	1.9900	0.1185	2
Σ_{12}^1	0.4561	0.0892	0.5439	0.0719	0.4881	0.0431	0.5
Σ_{22}^1	1.8907	0.2495	2.2330	0.2078	2.0936	0.1273	2
b^1	2.0187	0.2114	1.8741	0.1348	1.9454	0.0922	2
μ_1^2	-2.9917	0.0416	-2.9900	0.0297	-3.0273	0.0186	-3
μ_2^2	-3.9818	0.0416	-4.0051	0.0293	-3.9875	0.0185	-4
Σ_{11}^2	1.1689	0.1842	1.0778	0.1236	0.9624	0.0764	1
Σ_{12}^2	0.0189	0.0226	0.0029	0.0143	-0.0123	0.0080	0
Σ_{22}^2	1.1390	0.1814	1.0252	0.1182	0.9367	0.0748	1
b^2	5.2362	0.7806	5.7113	0.6220	6.2797	0.4768	6

As we can see, we get values that are pretty close to the real, true parameter values. Also, ML estimates are known to be asymptotically normal given enough smoothness conditions. If we use the standard errors reported, we could construct confidence intervals at each sample size and what we would then observe is that the true values are always within these intervals at all sample sizes. Finally, standard errors indeed get smaller as sample size increases, meaning that our ML estimates approach the true values. All in all, the table above suggests that we can be fairly certain that our program correctly performs the estimation of parameters in question.

Part 3

Multivariate NCT distribution MLE

We need to make something similar to what we did in part 2 but now for the multivariate noncentral t-distribution. Good thing is that we already have approximation of its log likelihood available to us. The code below is taken without any modification directly from our professor's textbook:

```
function pdfln = mvnctpdfln(x, mu, gam, v, Sigma)
% x        d X T matrix of evaluation points
% mu, gam  d-length location and noncentrality vector
% v is df; Sigma is the dispersion matrix.
[d,t] = size(x); C=Sigma; [R, err] = cholcov(C, 0);
assert(err == 0, 'C is not (semi) positive definite');
mu=reshape(mu,length(mu),1); gam=reshape(gam,length(gam),1);
vn2 = (v + d) / 2; xm = x - repmat(mu,1,t); rho = sum((R'\xm).^2,1);
pdfln = gammaln(vn2) - d/2*log(pi*v) - gammaln(v/2) -...
    sum(slog(diag(R))) - vn2*log1p(rho/v);
if(all(gam == 0)),return;end
idx = (pdfln >= -37); maxiter=1e4; k=0;
if(any(idx))
gcg = sum((R'\gam).^2); pdfln = pdfln - 0.5*gcg; xcg = xm' * (C \ gam);
term = 0.5*log(2) + log(xcg) - 0.5*slog(v+rho');
term(term == -inf) = log(realmin); term(term == +inf) = log(realmax);
logterms = gammaln((v+d+k)/2) - gammaln(k+1) - gammaln(vn2) + k*term;
ff = real(exp(logterms)); logsumk = log(ff);
```

```
while(k < maxiter)
    k=k+1;
    logterms = gammaln((v+d+k)/2) - gammaln(k+1) - gammaln(vn2) + k*term(idx);
    ff = real(exp(logterms-logsumk(idx))); logsumk(idx)=logsumk(idx)+log1p(ff);
    idx(idx) = (abs(ff) > 1e-4);if(all(idx == false)),break,end
end
pdfln = real(pdfln+logsumk');
end
end
```

Now we construct the objective function. Similar considerations as those described in part 2 apply.

```
function ll = NCTloglik (param, x, bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param), bound, 999); end
mu=param(1:2); gam = param(3:4); v = param(5); Sigma = zeros(2,2); Sigma(1, 1) = param(6);
Sigma(2, 2) = param(8); Sigma(1, 2) = param(7); Sigma(2,1) = Sigma(1, 2);
if (min(eig(Sigma))) < 1e-10, ll = 1e5; else
    llvec = mvnctpdfln(x', mu, gam, v, Sigma); ll=-mean(llvec); if isinf(ll), ll=1e5; end
end
end
```

The einschrk function used here is exactly the same as what was described in part 2. This allows us to make the estimation function we are after:

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

[nobs d] = size(x);
vin = nobs - 1;
mn = mean(x);
mn1 = mn(1);
mn2 = mn(2);
vr = cov(x);
s11 = vr(1, 1);
s12 = vr(1, 2);
s22 = vr(2, 2);
%%%%%%%%        mu1  mu2 gam1  gam2      v sg11 sg12 sg22
bound.lo    = [  -1   -1   -4    -4    1.1 0.01  -50 0.01];
bound.hi    = [   1    1    4     4    100   50   50   50];
bound.which = [   0    0    1     1      1    1    1    1];
initvec     = [ mn1  mn2    0     0     10    1    0    1];
maxiter=1000; tol=1e-6; MaxFunEvals=length(initvec)*maxiter;
opts=optimset('Display','iter','Maxiter',maxiter,'TolFun',tol,'TolX',tol,...
'MaxFunEvals',MaxFunEvals,'LargeScale','Off');
disp(initvec);
[pout,fval,~,theoutput,~,hess] = fminunc(@(param) NCTloglik(param, x, bound), einschrk(initvec,bound), opts);
V=inv(hess)/nobs;
%Don't negate because we work with the negative of the loglik
[param,V]=einschrk(pout,bound,V);
% transform and apply delta method to get V
param=param'; Varcov=V; stderr=sqrt(diag(V));
% Approximate standard errors
loglik=-fval*nobs; iters=theoutput.iterations;
end
```

Again, we are trying to be smarter with the initial vector here. Estimating mean and variance from the sample is quite easy and takes very little computational power. Thus, we can set our initial estimates of μ to be equal to the sample mean. We don't have any guess on what noncentrality parameter is equal to, so it is reasonable to leave it at 0, i.e. that there is no noncentrality in the distribution at all. For degrees of freedom we set the maximum at 100 for two reasons. First, there is hardly any difference in obtained values past that. Secondly, our optimization method is gradient based and it's in general not a great idea to have some parameter values many magnitudes larger than others - this can lead to problems. Thus, we leave our maximum at 100 and allow our optimization algorithm to reach this value if necessary, starting initially at 10. Dispersion matrix is initially set to be equal to the identity matrix.

AIC, BIC, and the Combined Program

For this subproblem we also need to be able to compute Akaike information criterion (AIC) and Bayesian information criterion (BIC). The formulas to obtain those are quite straightforward:

$$AIC = 2k - 2\ln(\hat{L})$$
$$BIC = k\ln n - 2\ln(\hat{L})$$

where k is the number of estimated parameters in the model, n is the sample size, and $\hat{L}$ is the maximized value of the likelihood function for the model. Note that ideally $\hat{L}$ is supposed to mean global maximum. However, many, but not all likelihood functions are log-concave. Thus, running our optimization algorithm we currently are not guaranteed to have obtained global maximum. Still, we can be fairly certain that what we do get is still a very good representation of what the optimal MLE estimate is equal to. Thus, with this remark, and recalling that our previous programs actually already compute the value of the log-likelihood, we can write the following two programs:

```
function [param, aic, bic] = fitMML(x)
    [param,stderr,iters,loglik,Varcov] = MLestimation(x)
    [nobs d] = size(x);

    aic = 2 * length(param) - 2 * loglik;
    bic = length(param) * log(nobs) - 2 * loglik;
end


function [param, aic, bic] = fitNCT(x)
    [param,stderr,iters,loglik,Varcov] = NCTMLE(x)
    [nobs d] = size(x);

    aic = 2 * length(param) - 2 * loglik;
    bic = length(param) * log(nobs) - 2 * loglik;
end
```

Here we quite easily extract k - by simply taking the length of the parameter vector. n is also extracted in a straightforward way by using function size and taking the first component. Finally, log-likelihood is the output of our model.

Example Run

Here we perform a test run of the code we wrote above to see that it more or less makes sense, sort of a basic sanity check before we try to use it in part 4.

```
rng(42);
sample = sampleLapMix(5000, 0.3, [5; 5], [2 0.5; 0.5 2], 2, [-3; -3], [1 0;
    0 1], 6);

[paramMML, aicMML, bicMML] = fitMML(sample);
[paramNCT, aicNCT, bicNCT] = fitNCT(sample);

disp(aicMML);
```

```
disp(aicNCT);

disp(bicMML);
disp(bicNCT);
```

Basically we make a sample and then fit both our mixture of multivariate Laplace (MML) and multivariate noncentral t-distribution (NCT) models. It is intentional that we generate data like that - this way we surely know that we should expect MML to perform better, since it is a natural choice in this case.

We get the following results:

$$AIC_{MML} = 50574 \quad AIC_{NCT} = 54026$$
$$BIC_{MML} = 50658 \quad BIC_{NCT} = 54078$$

Both AIC and BIC unanimously agree that MML performs better here. And this is despite the fact that this distribution has more parameters to fig meaning that it is more vulnerable to the fact that we chose only 5000 as our sample size (from our runs in part 2 and sample errors that we obtained it should be clear that 10000 or less is not ideal for getting reliable estimates of a 13-parameter model like this. Still, MML is a clear winner here (since for both of them less means better).

Part 4

We first quickly discuss how pairs are selected. For that, we need to say which two indices are chosen. There are 25 stocks, so 25 columns in total and 25 possible choices. Also, we need to ensure that we don't call the same index twice within the same pair since in that case this is not really a pair. The following program accomplishes that.

```
function [num1, num2] = selectTwoNumbers()
    num1 = randi([1, 25]);
    num2 = randi([1, 25]);

    while num2 == num1
        num2 = randi([1, 25]);
    end
end
```

The mechanics of this function are quite simple. We first simply draw two integers within $[1, 25]$ without any considerations whether we get the same or not. Then, if they are not, we are done. If they in fact are, we redraw the second integer until we get it different from the first. This way our distribution remains uniform and we ensure that these pairs always have different two integers within them.

Now that the sampling procedure is described, it is time to describe how the overal program functions. We decided to break it down into three functions: one serves for generating the table of AIC and BIC values while the other two can be called in order to generate relevant plots. It wasn't mentioned explicitly what plot is needed, but it

```
function tble = DijaMLE (df)
    reps = 50;
    tble = zeros(reps, 4);
    for rep = 1:reps
        [num1, num2] = selectTwoNumbers();
        sample = df(:, [num1, num2]);
        [paramMML, aicMML, bicMML] = fitMML(sample);
        [paramNCT, aicNCT, bicNCT] = fitNCT(sample);
        tble(rep, :) = [aicMML bicMML aicNCT bicNCT];
    end
end
```

For plots we use the following functions:

```
function plots (tble)
figure;
aicDiff = tble(:, 1) - tble(:, 3);
```

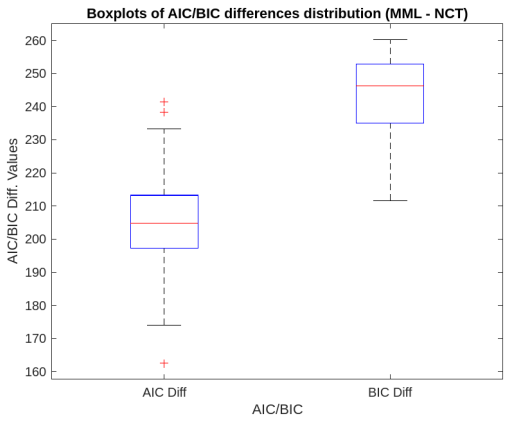
```
bicDiff = tble(:, 2) - tble(:, 4);
boxplot([aicDiff bicDiff], 'Labels', {'AIC Diff', 'BIC Diff'});
title('Boxplots of AIC/BIC differences distribution (MML - NCT)');
xlabel('AIC/BIC');
ylabel('AIC/BIC Diff. Values');
end
```

We would first like to explain why our choice of plots is like this. Running the first program above, we obtain a 50×4 matrix containing AIC and BIC values for each pair of stocks we sampled. We should first point out that absolute values of AICs and BICs carry no meaning and there is no point in dedicating our precious y-axis to showing what values exactly they are equal to. Thus, we transform the data by computing their differences. Then, lower value means that one model is (at least in-sample) is better than the other. Comparing it with 0 makes y-axis at least carry some meaning. Then, as we are limited on computation power and only 50 samples of these AIC and BIC values are available, it is crucial for us to understand how confident we can be in the conclusions that we can draw from looking at their differences. One simple way to do that is to encode the dispersion of their distribution in our visualization. Naturally, these considerations make us gravitate towards going with a boxplot type of chart. Thus, we construct a boxplot of showing differences between respective AIC values and BIC values.

The total output we are after is produced by simply running

```
load(DJIA30stockreturns.mat);
tble = DijaMLE(DJIARet);
plots(tble);
```

Then we get the following plot:



There are several things that we can observe. First of all, the difference in BIC values tends to be greater than the difference in AIC values but what's much more important is that both difereneces are positive. Since in our construction we subtracted AIC (and BIC) of a noncentral t-distribution from that of a mixture of multivariate Laplace distributions, this means that AIC (and BIC) of NCT is lower than that of the MML. In addition to that, looking at the dispersion of the values we obtained, it is also evident that we can be quite confident that these differences are indeed positive. This means that our results suggest that multivariate noncentral t-distribution does a better job at modelling financial asset's returns.

We have several ideas why we obtained such results. First of all, one plausible explanation lies in the heavy-tailed nature of financial data, a characteristic that is often better captured by the t-distribution. Stock returns are known for their occasional extreme movements, outliers, and fat tails, which may not be fully accounted for by a Laplace distribution. The noncentral t-distribution, with its ability to accommodate heavier tails, provides a more flexible framework for modeling extreme events and outliers commonly observed in financial markets.

Moreover, the multivariate noncentral t-distribution allows for the incorporation of a noncentrality parameter, representing the presence of a nonzero mean in the distribution. This feature is particularly relevant in financial modeling, as it acknowledges the existence of a potential market anomaly or abnormal behavior.

Bonus: Weighted Likelihood

The whole idea here is to give higher weight to more recent observations - this is quite relevant in finance where indeed more recent time-steps incorporate most relevant information which affects returns. This can be done by slightly modifying the log-likelihood function as shown below:

```
T=length(x); tvec=(1:T); omega=(T-tvec+1).^(rho-1); w=T*omega'/sum(omega);
ll = -mean(w.*llvec);
```

here ρ is a hyperparameter which governs how strongly we value more recent events. Recall that previously our code simply took the negative of the mean of the $llvec$. Now it applies weights before taking the mean.

In particular, in our case this change can be incorporated as follows. The code below is for computing log likelihood of the 2-mixture of bivariate Laplace distribution. We assume that rho is stored globally as a hyperparamter in the enviornment that can be accesess by our function.

```
function ll = MLloglik (param, x, bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param), bound, 999); end
[nobs, d] = size(x);
sigma1 = zeros(2, 2); sigma2 = zeros(2, 2); lambda=param(1); mu1=param(2:3)
    ;
sigma1(1, 1) = param(4); sigma1(2,2) = param(6); sigma1(1, 2) = param(5);
sigma1(2,1) = sigma1(1, 2); b1 = param(7); mu2 = param(8:9); sigma2(1, 1) =
     param(10); sigma2(2,2) = param(12);
sigma2(1,2) = param(11); sigma2(2,1) = sigma2(1,2); b2 = param(13);
% disp(b1);
if (min(eig(sigma1)) | min(eig(sigma2))) < 1e-10, ll = 1e5; else
    pdf = zeros(nobs, 1);
for i=1 : nobs, pdf(i) = mixMulLapPdf(x(i, :)', lambda, mu1', sigma1, b1,
    mu2', sigma2, b2);end
    T=length(x); tvec=(1:T); omega=(T-tvec+1).^(rho-1); w=T*omega'/sum(
        omega);
    llvec=log(pdf); ll = -mean(w.*llvec); if isinf(ll), ll=1e5; end
end
end
```

Similarly this change can be implemented for the log-likelihood of the noncental t-distribution.

```
function ll = NCTloglik (param, x, bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param), bound, 999); end
mu=param(1:2); gam = param(3:4); v = param(5); Sigma = zeros(2,2); Sigma(1,
     1) = param(6);
Sigma(2, 2) = param(8); Sigma(1, 2) = param(7); Sigma(2,1) = Sigma(1, 2);
if (min(eig(Sigma))) < 1e-10, ll = 1e5; else
    T=length(x); tvec=(1:T); omega=(T-tvec+1).^(rho-1); w=T*omega'/sum(
        omega);
    llvec = mvnctpdfln(x', mu, gam, v, Sigma); ll = -mean(w.*llvec); if
        isinf(ll), ll=1e5; end
end
end
```

After this, we can use these functions in the same they are used in other parts of this problem. It would be interesting to draw comparison how this affects out-of-sample performance, since this change is likely to improve our model - it is based on a sound reasoning. However, at this point we limit ourselves to what has been done so far.