Statistical Foundation for Finance: Assignment 2

Zhichao Pan (23-746-993) and Christoffer Guttulsrød (23-717-416)

University of Zurich

14th December, 2023

Abstract

In this project, we first investigate the computational efficiency of two numerical optimization algorithms: BFGS and iterative methods, in the context of maximum likelihood estimation (MLE). We show that their performance are somehow parameter dependent. Secondly, it has been proved that the conditional distribution of multivariate student t distribution is also a student t distribution in past literature. We use simulation method to confirm this theory. Thirdly, we further investigate the MLE for bivariate student t distribution and the distribution of the linear combination of its two components. Lastly, we study two bivariate distributions of the non-elliptic kind: discrete mixed Laplace and non-central student t. Their non-normal and non-elliptic nature render them to be a good fit for modelling real-world financial data. We conduct model fitting with a Dow Jones Industrial Average (DJIA) data set and make a comparison of the two models.

Contents

1 Comparison of MLE with BFGS and Iterating Method

Determining the MLE for parameters of the underlying distribution in most cases is not viable in analytical way, and thus has to resort to numerical methods. There are two normal ways to obtain the MLE: 1) finding the roots of the objective function and 2) finding the maximum of the likelihood function. The intent of this section is to compare two of such numerical methods in terms of the execution time of the computer programs that implement them, which are, 1) iterative method and 2) BFGS algorithm.

Iterative method [Pao18] was invented and employed for finding the roots of multiple (non-linear) equations for several parameters, for example, the location-scale student t distribution which has three parameters: location μ, scale σ and degrees of freedom ν. Applying first order condition to determine the MLE for these parameters comes to three equations

$$\dot{l}_\theta(\mathbf{\Theta}; \mathbf{x}) = 0, \theta \in \mathbf{\Theta} = \{\mu, \sigma, \nu\} \tag{1}$$

The three equations need to be solved simultaneously in order to determine $\hat{\mathbf{\Theta}}_{ML}$. The idea of the iteration method is to solve one equation for an unknown at a time while fixing the other two, and keep iterating over the three equations until convergence. In this way, simple algorithm can be employed, such as bisection, because only a single equation with one unknown is to be handled at once. The BFGS method belongs to what is referred to as Quasi–Newton family. It doesn't compute the exact Hessian matrix, as opposed to Newton-Raphson-type methods, but use matrices approximating the Hessian and its inverse.

To make a meaningful direct comparison of the computational efficiency between the two algorithms, we first need to calibrate the parameter of convergence tolerance (tol) for both of the algorithms, such that the accuracy of the estimation is within 4 significant digits. To this end, we perform a search for the largest possible tolerance. At first tol is set as 1×10^{-7}, and a set of parameters, denoted as $\hat{\mathbf{\Theta}}_0$, is obtained. Then the value of tol is increased by some step size until at least one of the three ML estimators has the same 4 significant digits as $\hat{\mathbf{\Theta}}_0$. The optimal convergence tolerance obtained from this procedure are 2×10^{-4} and 3×10^{-4} for iterative and BFGS methods, respectively.

We then use the same set of simulated data, 1,000 i.i.d samples of size 250 drawn from location-scale t distribution with various constellations of parameters, to test the computational efficiency of two computer programs, namely custom MatLab program `titer` and the built-in `mle`, which are the implementation of the two methods. The observation is intriguing because the performance of `titer` is shape dependent, i.e. sensitive to the ν parameter of the underlying student t distribution, but not to the location and scale. The boundary is $\nu = 5$, below which, `titer` can outrun `mle`, and they flips otherwise. The result is listed in the table below:

df	titer (sec)	mle (sec)
2	7.861	9.621
3	8.660	9.607
4	9.379	9.609
5	10.11	9.573
6	10.67	9.580
9	11.77	10.07
12	12.30	10.39

Table 1: Recorded runtime of titer and mle against various degrees of freedom, and fixed $\mu = 1, \sigma = 2$.

As an aside, to call MatLab programs from Python requires the MatLab Engine API being installed on the system. Once available, import it in the Python environment and start the engine with

```
import matlab.engine
eng = matlab.engine.start_matlab()
```

Then call any MatLab function in the working directory, say `myFunc`, by

```
output = eng.myFunc(input, nargout= )
```

2 Conditional Distribution of Bivariate Student t Distribution

Multivariate t (MVT) distributions are playing an increasingly important role in statistical modelling, as they can provide a more realistic alternative to multivariate normal when working with real world data. This is particularly owing to its property of heavy tails. In this section we are to investigate the conditional distribution of MVT.

Suppose a p-dimensional random vectors $\mathbf{X}$ following a MVT distribution. We can partition it into two random vectors $\mathbf{X}_1, \mathbf{X}_2$ with dimensions p_1 and p_2 respectively, where $p_1 + p_2 = p$. Although Nadarajah and Kotz [NK05] successively derived the density function of $\mathbf{X}_2$ conditioned on $\mathbf{X}_1$, the distribution was not recognizable to them then. Howevever, Ding [Din16] claimed that the distribution of $\mathbf{X}_2|\mathbf{X}_1$ is in fact also a MVT distribution:

$$\mathbf{X}_2|\mathbf{X}_1 \sim t_{p_2}(\boldsymbol{\mu}_{2|1}, \frac{\nu + d_1}{\nu + 1}\boldsymbol{\Sigma}_{22|1}, \nu + p_1), \tag{2}$$

where

$$\begin{aligned} \boldsymbol{\mu}_{2|1} &= \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{X}_1 - \boldsymbol{\mu}_1), \\ d_1 &= (\mathbf{X}_1 - \boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{X}_1 - \boldsymbol{\mu}_1), \\ \boldsymbol{\Sigma}_{22|1} &= \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} \end{aligned}$$

Ding derived an intuitive and clean proof while averting the technical impediment of directly dealing with the complicated MVT joint p.d.f. The purpose of this section is to confirm the theory using Monte Carlo simulation. We will, nevertheless, restrict ourselves to bivariate t distribution only. In this case, the conditional distribution of X_2 given $X_1 = x_1$ reduces to a univariate location scale t distribution, with

$$\begin{aligned} mean &= \mu_2 + \frac{s_{21}}{s_{11}}(x_1 - \mu_1) \\ scale &= \frac{\nu + (x_1 - \mu_1)^2/s_{11}}{\nu + 1}(s_{22} - \frac{s_{12}s_{21}}{s_{11}}) \\ df &= \nu + 1 \end{aligned} \tag{3}$$

where $\Sigma = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}$ is the dispersion matrix or scale matrix, which is symmetric and positive definite. The off-diagonal entries s_{12}, s_{21} are associated to the correlation coefficient of X_1, X_2.

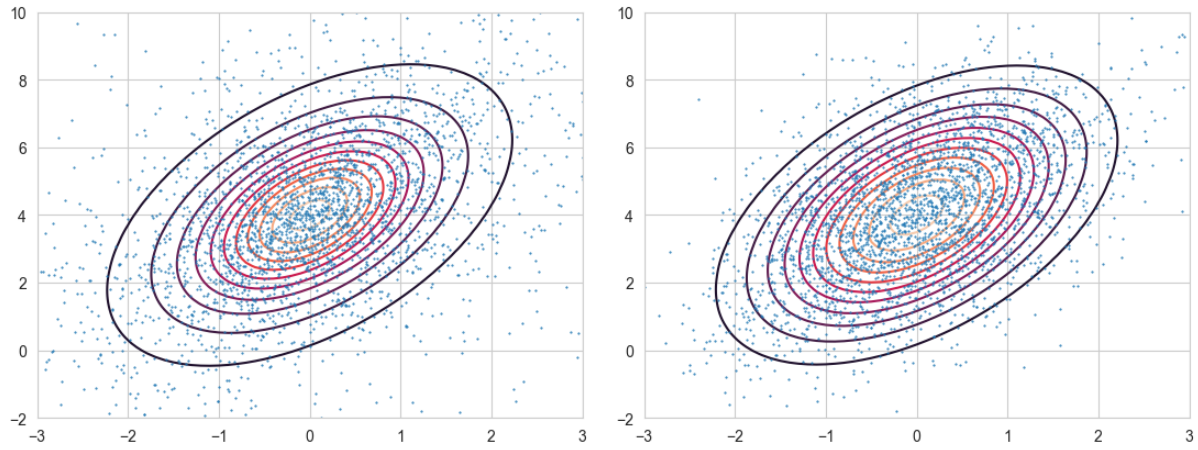
However, it is noteworthy that Σ being diagonal is a necessary condition for X_1, X_2 to be independent, but not sufficient. To see this, we can write down the simplest form of joint density function with $\mu_1 = \mu_2 = 0$ and Σ being the identity matrix:

$$f_{X_1,X_2}(x_1, x_2) = \frac{\Gamma\left((\nu + 2)/2\right)}{\nu\pi\Gamma(\nu/2)}\left(1 + (x_1^2 + x_2^2)/\nu\right)^{-(\nu+2)/2} \tag{4}$$

It is obviously impossible to factorize (4) into the product of the marginal densities of X_1 and X_2, therefore, $f_{X_1,X_2}(x_1, x_2) \neq f_{X_1}(x_1)f_{X_2}(x_2)$. We can also perceive this fact from the normal mixture representation of MVT:

$$\mathbf{X} = \boldsymbol{\mu} + \sqrt{G}\boldsymbol{Z} \tag{5}$$

where $\boldsymbol{Z} \sim N(\mathbf{0}, \boldsymbol{\Sigma})$, and G is a scalar random variable following the inverse gamma distribution $IGam(\nu/2, \nu/2)$. Since every component of X share the same G, they can never be independent even though Σ is diagonal. Independence is possible only when $\nu \to \infty$, as MVT converges to multivariate normal in distribution in the limiting case [Pao19].



(a) (b)

Figure 1: Simulation of 3,000 replications: (a) Bivariate t distribution with $\nu = 3$, (b) Bivariate normal, for both $\boldsymbol{\mu} = (0, 4)^T, \Sigma = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 4 \end{smallmatrix}\right]$.

2.1 Simulation of Bivariate t Conditional Distribution

Simulating MVT is straightforward with the `stats.multivariate_t.rvs` function in Python using the package SciPy. Figure 1 is the scatter plot of a random sample of size 3,000, with an overlaid contour plot of the density with true parameter values. To make a visual comparison, we place the same plots aside obtained from a bivariate normal random sample. It is clearly visible that the data points of the bivariate t are more scattering off the center than the bivariate normal.

Verifying the conditional distribution Eq. (2) via simulation method is not viable in theory, because MVT are continuous random variables, as such $X_1|X_2 = x_2$ is a "measure zero" event. We are thus to approximate the p.d.f. of $X_1|X_2$ by generating a large number of (X_1, X_2) pairs, and retaining those values of X_1 such that the corresponding values of X_2 falls within a small interval $(x_2 - \epsilon, x_2 + \epsilon)$. The set of the X_1 values obtained from this process can be considered as a sample drawn from the distribution $X_1|X_2 = x_2$, as long as ϵ is sufficiently small. As an initial attempt we choose $\epsilon = 1.5 \times 10^{-5}$, and x_2 to be the sample mean $\bar{X_2} = 1.587 \times 10^{-5}$, and $\nu = 6$, $\boldsymbol{\mu} = (0, 0)$, $\Sigma = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}$. With 100 million pairs, there are 1,132 of x_1 remaining in the interval. We then use both iterative method and MLE (with BFGS) to estimate the parameters (μ, σ, ν) for these numbers, assuming that they do follow some univariate student t distribution. The theoretical values of the parameters are calculated based on Eq. 2, with the following Python routine:

```python
def tConditional(X1, mu1, mu2, Sigma: np.ndarray, v):
    '''
     This function returns the parameters of the conditional distribution of
     ↪  X2|X1, where (X1, X2) are bivariate location scale t distributed random
     ↪  variables, based on Ding (2016).
    :param X1: value of X1, float
    :param mu1: location of X1, float
    :param mu2: location of X2, float
    :param Sigma: dispense matrix of X1 and X2, 2x2 ndarray
    :param v: degrees of freedom, same for the X1 and X2, float
    :return:
        muout: location of the resulting conditional t distribution
        cout: scale of the resulting conditional t distribution
        vout: degree of freedom of the resulting conditional t distribution
    '''
    d1 = (X1 - mu1)**2 / Sigma[0,0]
```

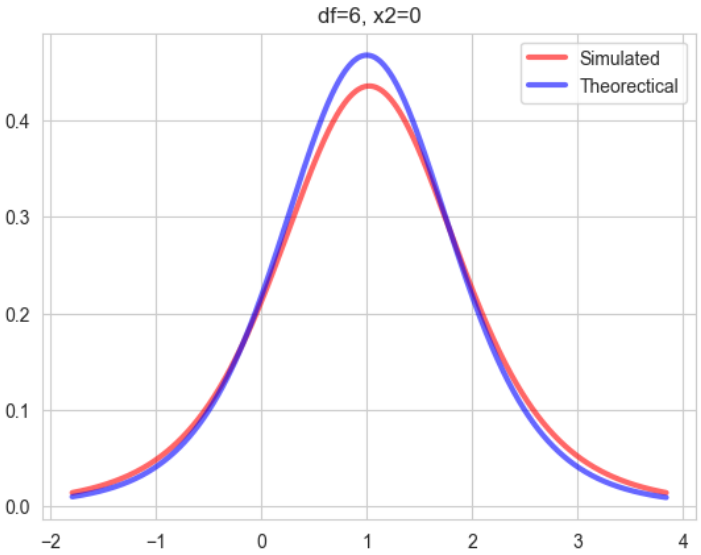


Figure 2: Kernel densities of the distributions based on theoretical and estimated parameters.

```
    muout = mu2 + Sigma[1,0] * (X1-mu1) / Sigma[0,0]
    fct = (v+d1)/(v+1)
    cout = fct * (Sigma[1,1] - Sigma[1,0]**2/Sigma[0,0])
    vout = v + 1
    return muout, cout, vout
```

2.2 Results and Discussion

The resulting estimations are listed in Table 2. The values given by `titer` and `mle` are identical up to the 5th decimal place, and they both underestimate the degrees of freedom. We also plot the kernel densities based on the theoretical and estimated values of the parameters, as shown in Figure 2. Holding the same location and scale parameters, we repeat the above process by varying the degrees of freedom ($nu = 1, 3, 5$) and x_2 values ($x_2 = 0.0, 0.6, 1.2, 2.0$), and present the resulting kernel density plots in Figure 3.

Our investigation of the bivariate t conditional distribution via simulation method to some extend agrees with the theory that $X_1|X_2$ is a student t distribution, whose parameter is given by Eq. 2. From Figure 3 we can see that the theoretical and simulated density align with each other better at larger degrees of freedom. The value of X_2 on which X_1 is conditioned on also has an effect on the discrepancy, but it is more likely due to the fluctuation of randomness coming with the simulation and the standard error of the MLE method.

	μ	σ	ν
Theoretical	1.000	0.8229	7
Iterative	1.023	0.8756	5.617
MLE	1.023	0.8756	5.618

Table 2: Theoretical and estimated values of the parameters of $X_1|X_2 = x_2$.

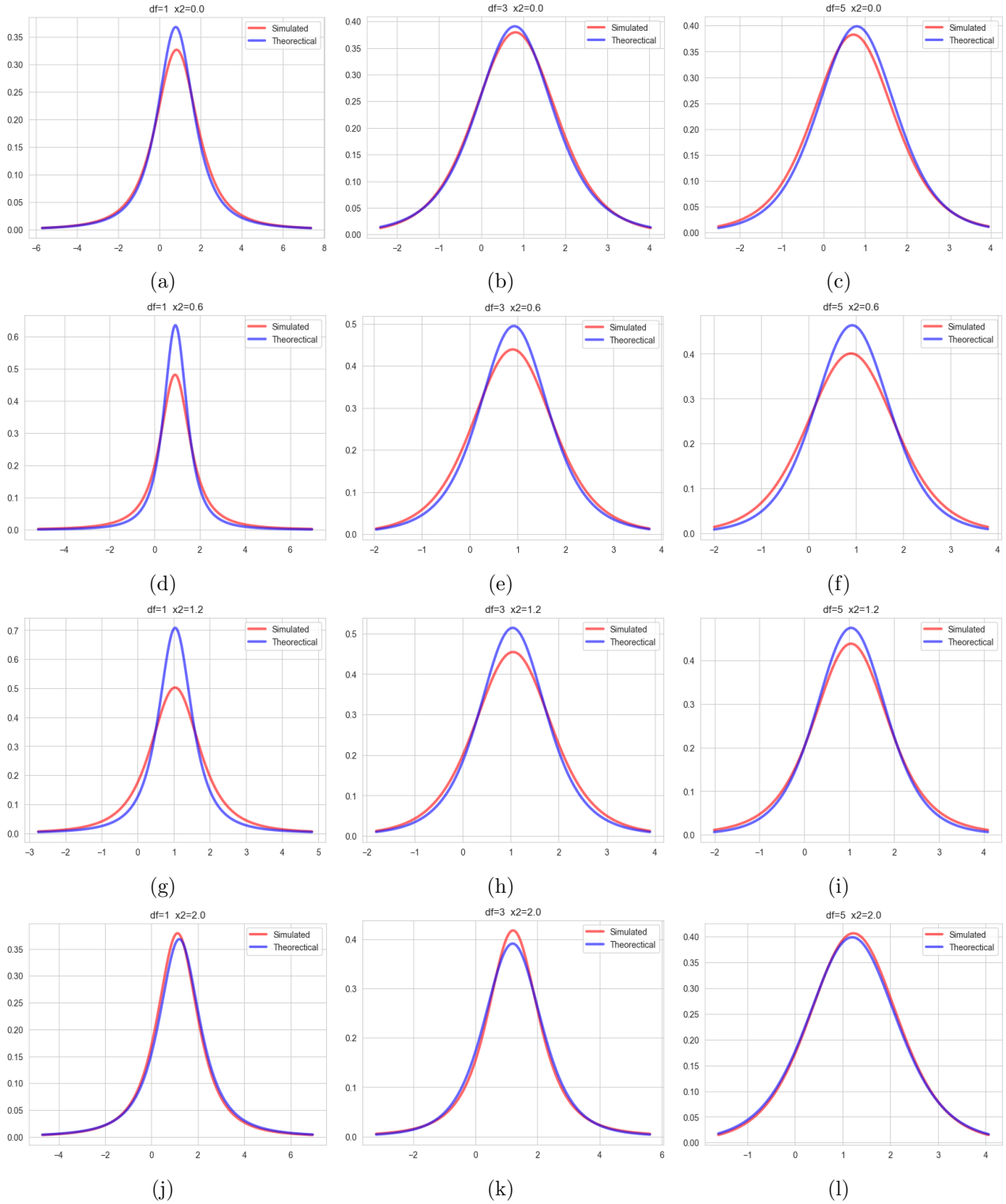


Figure 3: Kernel densities of the distributions with various degrees of freedom and X_2 values.

3 MLE of Bivariate t Distribution and Distribution of Linear Combination of Two Jointly Distributed Random Variables

This section continues to study the estimation of bivariate location scale t distribution with MLE. We will also further investigate the distribution of the linear combination of the two components of a bivariate t distribution, which has an immediate application in modelling the return of asset portfolio.

3.1 Estimation of Bivariate t Distribution

For a biviarte location scale t distribution, there are a total of six parameters to be estimated, namely, $\nu, \mu_1, \mu_2, \sigma_1, \sigma_2$ and σ_{12}, the number is well in the "conform zone" of MLE method. We will look into a MatLab program `MVTestimation`that has been well developed in [Pao19] (Page 527) and test its estimating accuracy with simulated bivariate t data. The program calls the MatLab built-in function `fminunc` to do the optimization job, which employs the BFGS algorithm by default. The program imposes a so-called box constraint on the estimators to make sure that estimated values lie within the theoretically allowed range, for example, $\hat{\nu}_{ML}$ should always be greater than zero. The imposition is done through another MatLab routine `einschrk` developed in the author's another book [Pao18] (Page 142). The idea is to transform a parameter θ into $\phi = \sqrt{\frac{b-\theta}{\theta-a}}$, when the constraint to be imposed is $a < \theta < b$. The program evaluates the p.d.f of a bivariate location scale t distribution based on

$$f_{\mathbf{X}}(x1, x2; \mu_1, \mu_2, \Sigma, \nu) = \frac{f_{\mathbf{T}}(t_1, t_2; R, \nu)}{\sigma_1 \sigma_2}, \tag{6}$$

where as previously discussed, Σ is the dispersion or scale matrix. $R = \begin{bmatrix} 1 & \rho_{12} \\ \rho_{21} & 1 \end{bmatrix}$ is the correlation matrix.

Again, we will make use of simulation to test the aforementioned programs, with a choice of the following parameters: $\nu \in \{1, 3, 6, 9\}$, $\mu_1 \in \{1.0, 4.0\}$, $\mu_2 = 2.0$, $c_1 \in \{0.5, 1.0\}$, $c_2 \in \{1.5, 2.0\}$, $\rho \in \{0.2, 1.0, 4\}$. The combinations of such parameters form a total of 128 constellation, with each of which we draw a sample of size 1,000 from the corresponding bivariate location scale t distribution. The relative errors between the MLE and the true values for each of the six parameters are presented in Figure 4 in the form of boxplot. The median of the relative errors are all sitting very close to zero, confirming the accuracy of the MLE with the MatLab programs discussed above. The precision of the estimators for the parameters are different though.

3.2 Distribution of Linear Combination of Two Jointly Distributed Random Variables

The sum of two student t random variables, whether independent or not, is not a student t. The distribution of the sum, as a random variable, is of value to be investigated. Imagine the returns of two securities, which are surely correlated and whose distribution can be modelled by a bivariate t random vectors $\mathbf{X} = (X_1, X_2)^T$. The linear combination of the two components $P = \alpha X_1 + (1-\alpha) X_2$ thus represents the return of a portfolio constituted by the two securities, where $0 \leq \alpha \leq 1$ is the weight.

We draw 100,000 independent random samples from a bivariate location scale t distribution with parameters: $\alpha = 0.2, \nu = 6, \boldsymbol{\mu} = (0.0, 10.0)^T$, scale $\mathbf{c} = (1.0, 2.0)^T$, and correlation coefficient $\rho = 0.6$, and plot their histogram in Figure 5 with the empirical kernel density.

We can retrieve the real p.d.f of the random variable P with the above parameters by inverting the characteristic function, derived in [Pao19]:

$$\varphi_P(t) = e^{it\mu} \frac{K_{\nu/2}(\nu^{1/2}|t|\kappa)(\nu^{1/2}|t|\kappa)^{\nu/2}}{2^{(\nu/2)-1}\Gamma(\nu/2)}, \tag{7}$$

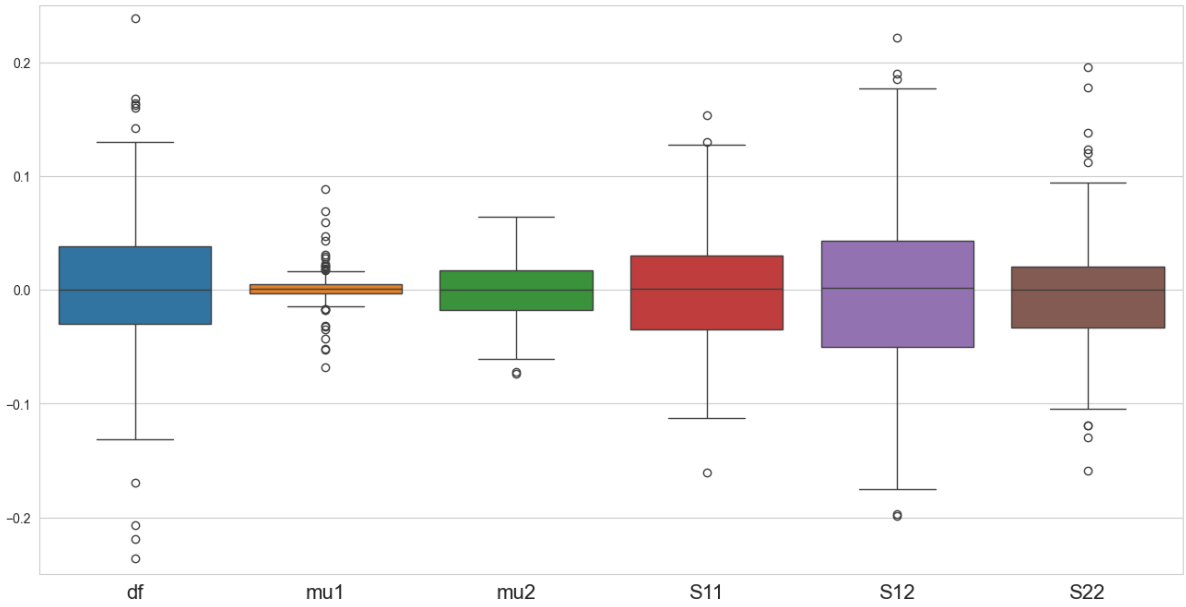


Figure 4: Boxplots of the relative errors between the ML estimators and the true values of the six parameters of the bivirate location scale t distribution.

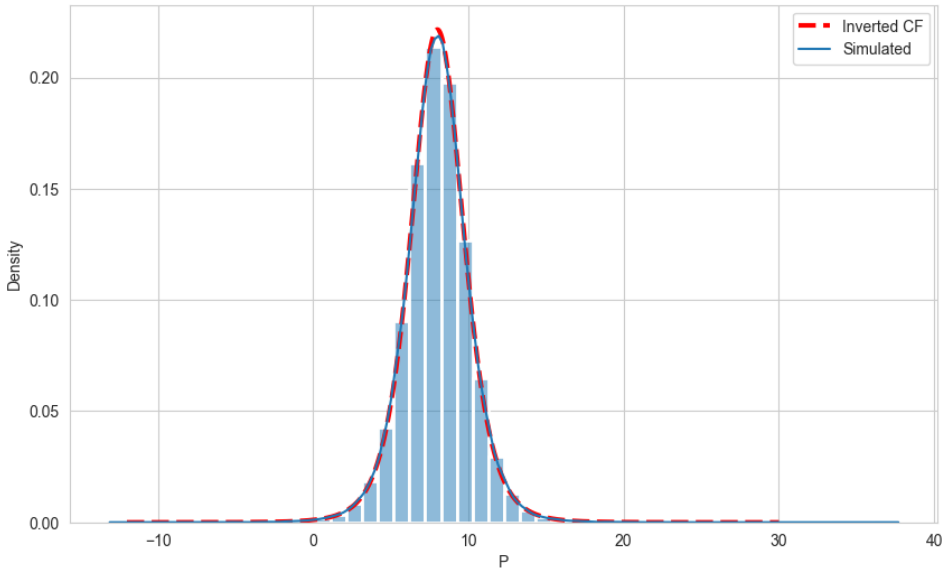


Figure 5: Histogram and empirical kernel density of $P = \alpha X_1 + (1-\alpha)X_2$ simulated by 100,000 indenpent samples (blue solid), overlaid by the true density obtained from inversion of the characteristic function (red dashed).

where $\mu = \alpha\mu_1 + (1-\alpha)\mu_2$, $\kappa = ||\Sigma^{1/2}(\alpha, 1-\alpha)^T||$.

We employ the fast Fourier transform (FFT) to conduct the inversion of Eq. 7. The implementation of the algorithm in Python is listed below:

```
def sum_mvt_IFFT(muvec, avec, Sigma, v, ell=-6, dx=0.001, T=2**16):
    # Inverser fast Fourier transsformation for student t
    # lowlim is the lower limit in finite integral approximation to the c.f.
    # use a smaller value of lowlim for low dof
    if df < 0:
        raise ValueError("Degree of freedom should be postive.")
    if isinstance(df, float) and (not df.isinteger()):
        raise ValueError("Degree of freedom should be an integer.")
    t = np.arange(-T/2, T/2)
    if np.any(t==0):
        t[np.where(t==0)] = 1e-8
    s = 2*np.pi*t/(T*dx)
    phi = sum_mvt_cf(s, muvec, avec, Sigma, v)
    g = phi * np.exp(-1.j*s*ell)
    P = ifft(g)
    pdf = P/dx
    x = np.arange(dx, (T+1)*dx, dx) + ell
    if max(np.imag(abs(pdf))) > 1e+8:
        raise ValueError("The value of PDF contains imaginary numbers.")
    return np.flip(abs(pdf)), x
```

We observe from Figure 5 that the true density obtained from characteristic function inversion overlays the kernel from simulation closely. The distribution of P is symmetric about the center close to $\mu_2 = 10.0$ (the weight of X_2 is set to be 0.8).

4 Non-elliptic Distributions for Financial Modelling

Random variables can be categorized as symmetric and asymmetric. Typical examples of the former are the continuous normal, student t, Cauchy, and the discrete binomial. The gamma family, including exponential and chi-squared, are all asymmetric. When it comes to the multivariate setting, some joint distributions such as normal and student t preserves such symmetry. The notion of ellipticity is therefore a natural extension from reflection symmetry to rotational symmetry in the higher dimensions.

While elliptic distributions have played an important role in portfolio risk analysis and optimization, interest for the application of non-elliptic ones is increasingly growing. It is the fundamental assumption for the use of elliptic distributions that the return of every asset in the portfolio is symmetrically distributed, all of the marginal distributions share the same tailing behavior for both end. For instant, MVT has a unique degrees of freedom. There are, nonetheless, material evidence that real-world asset returns are not only non-normal, but even non-elliptic [PPW21, Pao19], dismissing this assumption as unrealistic.

In this section, we particularly look into two non-elliptic distributions: the 2-component discrete mixture of bivariate Laplace distribution (denoted as Mix_2Lap_2), and the bivariate non-central location scale student t distribution (denoted as MVNCT_2).

4.1 Simulation and MLE of Mix_2Lap_2

Mix_2Lap_2 is defined in terms of its p.d.f as [Pao19]

$$f_{Mix_2Lap_2}(y_1, y_2; \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma_1, \Sigma_2, \lambda_1, \lambda_2, b_1, b_2) = \sum_{j=1}^{2} \lambda_j f_{Lap}(y_1, y_2; \boldsymbol{\mu}_j, \Sigma_j, b_j), \tag{8}$$

where λ_j is the mixing weight, f_{Lap} denotes the joint p.d.f of the bivariate Laplace distribution with 2-dimensional location vector $\boldsymbol{\mu}_j$, dispersion matrix Σ_j and scale parameter b_j.

There is no built-in function generating bivariate Laplace random numbers either in MatLab or in SciPy. Yet note that the conditional distribution of a multivariate Laplece random vector $\mathbf{Y}$ on a gamma random variable is a multivariate normal distribution. Specifically, $\mathbf{Y}|G = g \sim N_2(\boldsymbol{\mu}, g\Sigma)$, with $G \sim Gam(b, 1)$. Taking advantage of this fact, we can simulate the bivariate Laplace in a 2-step indirect process: firstly generate a gamma random number g; secondly generate bivariate normal random vectors with the g as a parameter. Figure 6 showcases a simulation of 50,000 independent random vectors out of Mix_2Lap_2, along with the marginal histogram of each component.

We adapt the MatLab program `MVTestimation`, discussed in Chapter 4, for computing the MLE of Mix_2Lap_2. The adapted program `MixLapEstimation` is listed in Appendix A. The evaluation of the joint p.d.f. of a multivariate Laplace distribution involve a Bessel function in integral form:

$$K_z(x) = \frac{1}{2} \int_0^\infty u^{z-1} exp\left[-\frac{x}{2}\left(\frac{1}{u} + u\right)\right] du, z \in \mathbb{R}, x \in \mathbb{R}_+. \tag{9}$$

It is referred to as modified Bessel function of the **third** kind, and is supposed to linked to the built-in MatLab function `besselk`. The only concern is that in the documentation of `besselk` it is referred to as modified Bessel function of the **second** kind. To assure ourselves, we evaluate the integral in Eq. 9 numerically at 200 points within interval $(1.0, 5.0)$ for 4 different z values $(1, 2, 3, 5)$. Table 3 reports the absolute greatest difference among the evaluation points for each z value.

We produce a large simulation of 50,000 independent samples for the MLE of the Mix_2Lap_2 distribution. The outcome of the MLE along with the standard error are presented in Table 4.

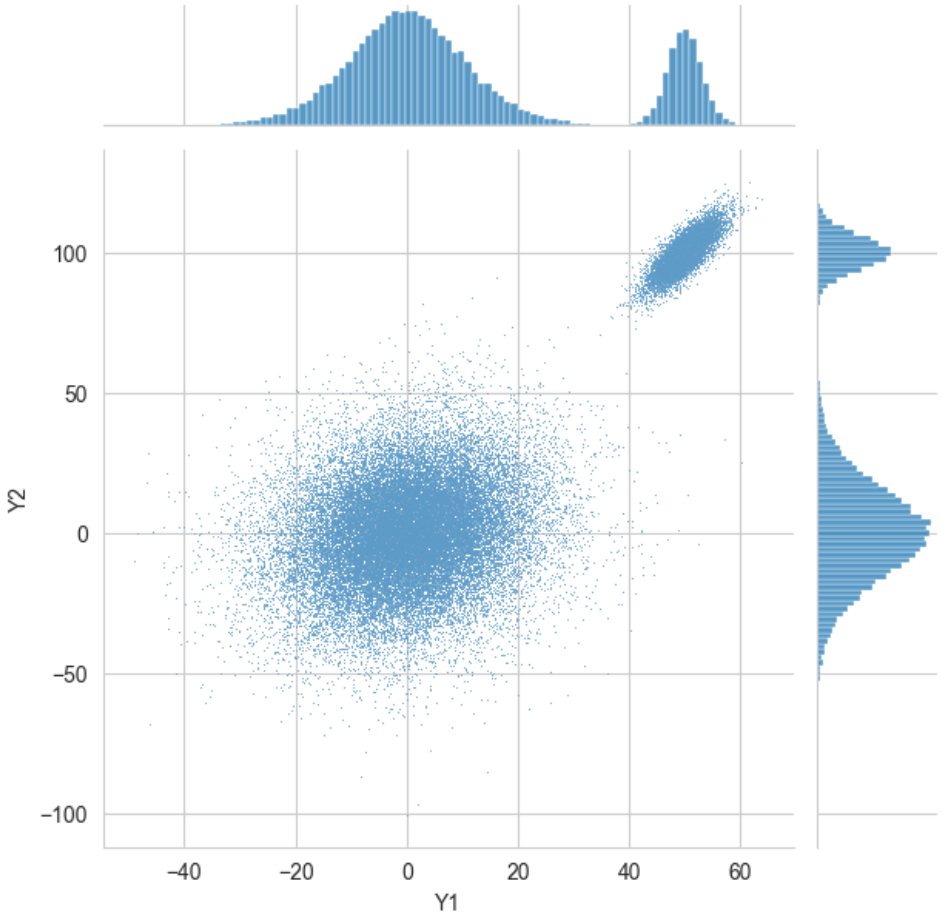


Figure 6: A realization of Mix_2Lap_2 with 50,000 replications, $\boldsymbol{\lambda} = (0.2, 0.8)^T$, $\boldsymbol{\mu}_1 = (50, 100)^T$, $\boldsymbol{\mu}_2 = (0, 0)^T$, $\Sigma_1 = \left[\begin{smallmatrix} 1 & 1.6 \\ 1.6 & 4 \end{smallmatrix}\right]$, $\Sigma_2 = \left[\begin{smallmatrix} 25 & 8 \\ 8 & 64 \end{smallmatrix}\right]$, $\mathbf{b} = (10, 5)^T$.

z	max. abs. difference
1.0	3×10^{-16}
2.0	9×10^{-15}
3.0	4×10^{-13}
5.0	4×10^{-13}

Table 3: Confirmation of the kind of Bessel function.

	μ_{11}	μ_{21}	μ_{12}	μ_{22}	$S1_{11}$	$S1_{12}$
True	50.0	100.0	0.0	0.0	1.0	1.6
MLE	49.97	99.91	0.0329	0.002	1.133	1.822
Std. Err.	0.031	0.063	0.054	0.088	0.037	0.062
	$S1_{22}$	$S2_{11}$	$S2_{12}$	$S2_{22}$	b_1	b_2
True	4.0	25.0	8.0	64.0	10.0	5.0
MLE	4.573	23.94	7.565	62.50	8.910	5.160
Std. Err.	0.149	1.02	0.379	2.65	0.291	0.208

Table 4: MLE and standard errors of the 14 parameters of Mix_2Lap_2.

	ν	μ_1	μ_2	S_{11}	S_{22}	R_{12}	γ_1	γ_2
True	3	3.0	1.0	1.0	1.0	0.5	1.0	3.0
MLE	1.925	2.425	-0.7735	0.9456	1.237	0.5660	1.741	3.997
Std. Err.	0.008	0.024	0.019	0.005	0.005	0.005	0.032	0.004

Table 5: MLE and standard errors of the 8 parameters of bivariate NCT.

4.2 Simulation and MLE of MVNCT$_2$

Just like the multivariate student t distribution, MVNCT can also be constructed as a continuous mixture of multivariate normals [Pao19]:

$$\mathbf{X} = \boldsymbol{\mu} + \sqrt{G}\mathbf{S}(\boldsymbol{\gamma} + \mathbf{R}^{1/2}\mathbf{Z}), \tag{10}$$

where $G \sim IGam(\nu/2, \nu/2)$ is an inverse gamma random variable independent of $\mathbf{Z} \sim N_d(\mathbf{0}, \mathbf{I})$. The vector $\boldsymbol{\gamma}$ determines the degree of noncentrality, and diagonal matrix $\mathbf{S} = diag(\boldsymbol{\sigma})$. This representation suggests that the conditional distribution of MVNCT given G is multivariate normal. Indeed, $\mathbf{X}|G = g \sim N_d(\boldsymbol{\mu} + g\boldsymbol{\gamma}, g\boldsymbol{\Sigma})$, therefore, analogous to the bivariate Laplace, we also take the 2-step method to perform the simulation for bivariate NCT, implemented by the following Python program. Figure 7 showcases a simulation of 3,000 independent samples, overlaid with the contour plot, from which the non-ellipticity of the distribution can be evidently seen.

```python
def simMVNCT(muvec, gam, v, Sigma, n=1000):
'''
This function generates n 2-dim random vectors of bivariate noncentral
↪  location-scale t distribution.
:param muvec: location,
:param gam: noncentrality
:param Sigma: scale matrix
:param v: degrees of freedom
:param n: int
:return: n x 2 numpy.ndarray
'''
eigvals = np.linalg.eig(Sigma)[0]
if np.min(eigvals) < 1e-10:
    raise ValueError("The scale matrices have to be positive definite.")
Y = np.zeros((n,2))
for i in range(n):
    g = stats.gamma.rvs(v/2, scale=2/v)
    if abs(g) < 1e-6:
        g = 1e-6;
    g = 1/g;
    Y[i] = multivariate_normal.rvs(mean=muvec+g*gam, cov=g*Sigma)
return Y
```

We again adapt the MatLab program `MVTestimation` for computing the MLE of bivariate NCT. The adapted program `MVNCTestimation` is listed in Appendix A. The evaluation of the joint p.d.f. of a bivariate NCT is accomplished by MatLab program `mvnctpdfln`, given in [Pao19] (Page 532). Table 5 reports the MLE outcome and standard errors for the bivariate NCT with 50,000 independent samples.

4.3 Fitting Asset Return Dataset with Mix$_2$Lap$_2$ and MVNCT$_2$

We will further investigate the Mix$_2$Lap$_2$ and MVNCT$_2$ distribution by model fitting with real-world data. The employed data set consists of the log percentage daily returns of 25 selected stocks on DJIA from Jan 1990 to Oct 2021. The data set contains exactly 8,000 rows. Figure 8 presents the basic descriptive statistics of the first 8 stocks. It is noticeable that 9 out of the 25 stocks has zero median.

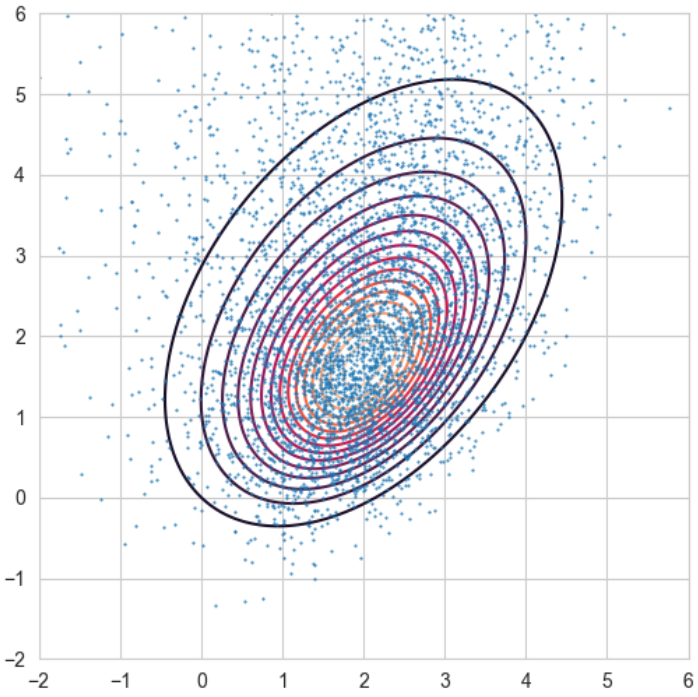


Figure 7: A realization of MVNCT_2 with 5,000 replications, $\boldsymbol{\mu} = (2, 0)^T$, $\Sigma = \left[\begin{smallmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{smallmatrix}\right]$, $\boldsymbol{\gamma} = (0, 2)^T$, $\nu = 5$.

	0	1	2	3	4	5	6	7
count	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000
mean	0.078523	0.043948	0.037785	0.050535	0.037189	0.040402	0.070512	0.031179
std	2.805319	2.224262	2.098916	2.026456	1.650040	1.878604	1.986848	1.744228
min	-73.124703	-19.352330	-27.244427	-15.685887	-25.006226	-20.288783	-33.868146	-16.889004
25%	-1.217343	-0.918200	-0.960986	-0.991115	-0.795864	-0.884856	-0.879657	-0.789085
50%	0.038773	0.000000	0.000000	0.000000	0.030789	0.000000	0.053083	0.022599
75%	1.398769	1.042033	1.060951	1.105389	0.884153	0.943888	1.030065	0.858658
max	28.679560	19.788523	21.767751	14.555559	20.490370	14.818051	14.215094	12.363560

Figure 8: Descriptive statistics of the first 8 stocks out of the DJIA data set.

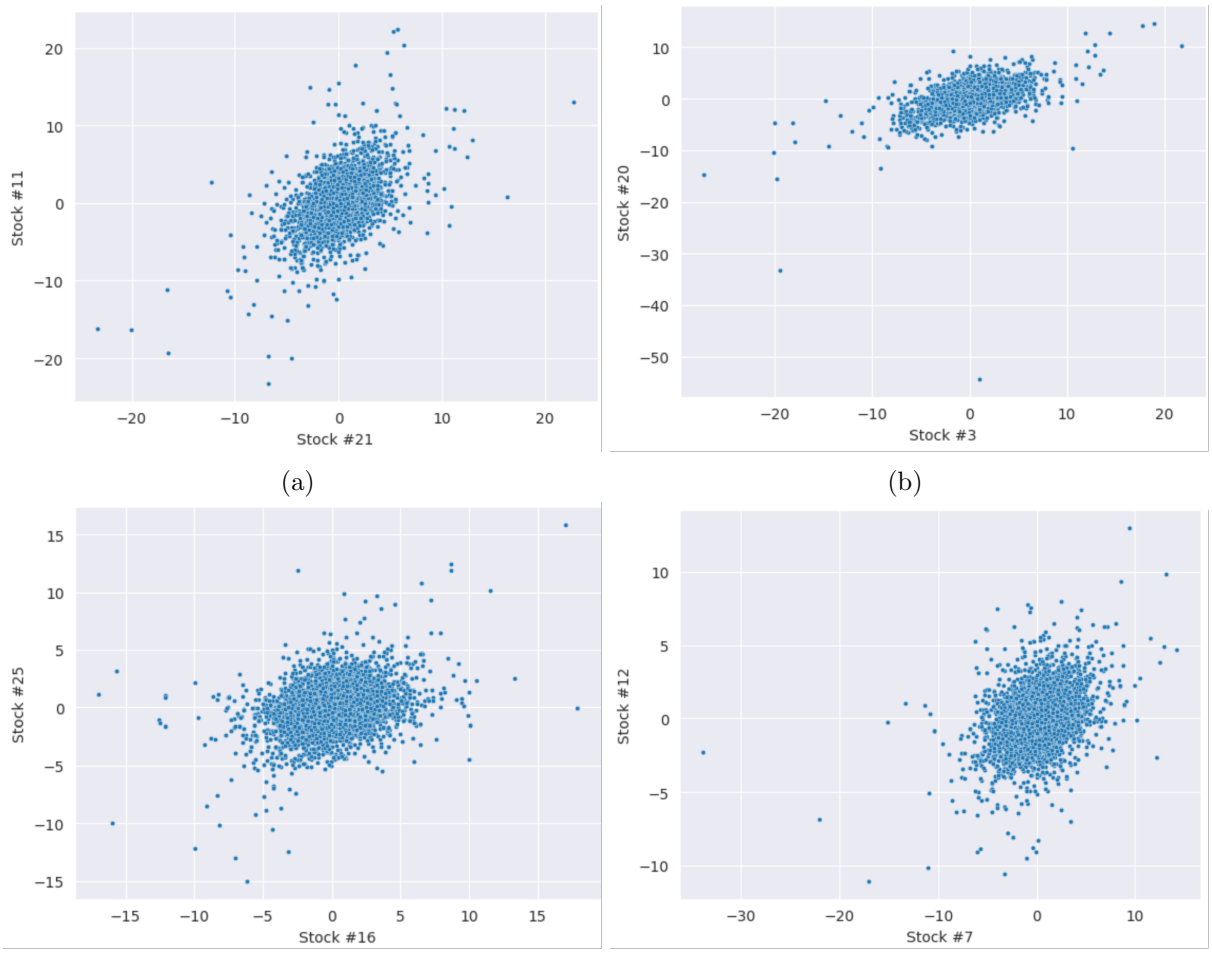


Figure 9: Scatterplots of the daily returns of 4 pairs of stocks.

Since in this study we restrict ourselves to bivariate case only, we will pick up pairs of stocks at random to form an array of shape $(8,000, 2)$ and to fit the bivariate models. This is equivalent to randomly draw 2 numbers out of the range of integers $[1, 25]$ without replacement, which can be easily accomplished in one-line code with the Numpy package of Python:

```
a1, a2 = numpy.random.choice(ns, 2, replace=False)
```

4 pairs of the stocks are chosen to show their scatterplots in Figure 9.

We will employ the information criterion (derived from information theory) to compare the two models, specifically, Akaike Information Criteria (AIC), which is the earliest one to be proposed, and Bayesian Information Criteria (BIC), the later modified version:

$$AIC = 2(k - LL) \tag{11}$$

$$BIC = kln(n) - 2LL, \tag{12}$$

where k is the number of parameters of a model, and LL the maximal values of log-likelihood function, and n the number of observations. A model with lower values of AIC and/or BIC is more favorable. Based on the definition 12, both criterion penalize the complexity of a model in terms of the number of parameters - an idea analogous to adjusted-R^2. Given that, we anticipate that the MVNCT$_2$ model is of an inherent advantage of the Mix$_2$Lap$_2$, simply because its number of parameters (8) is less than that of Mix$_2$Lap$_2$ (12).

The above discussed process is repeated for 50 times, for each time the AIC and BIC are calculated after the model fitting with a pair of stock data. This is accomplished by the following Python function:

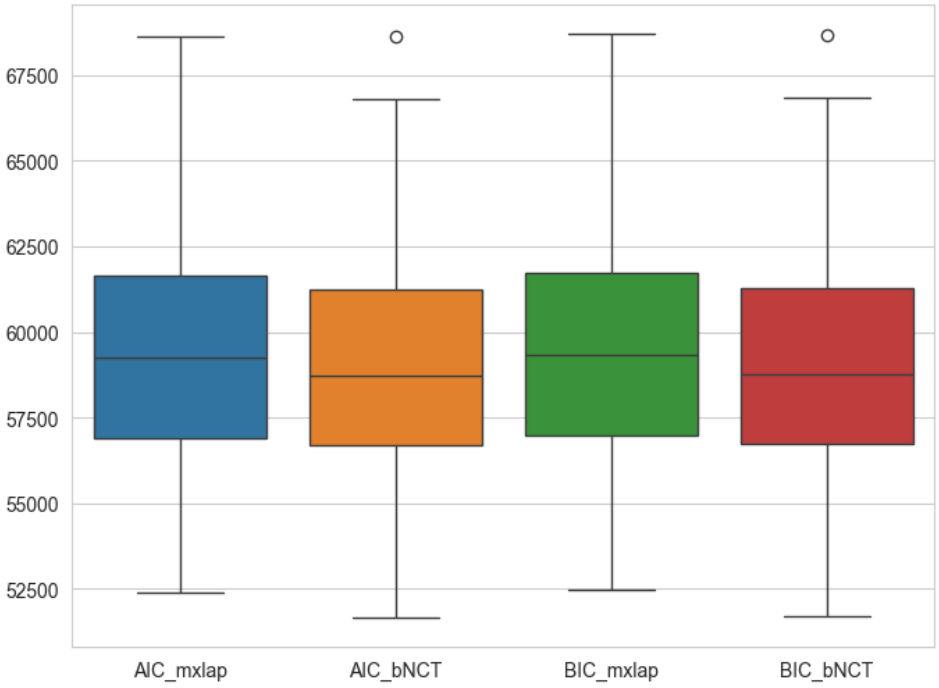


Figure 10: AIC and BIC values of the mixed Laplace and bivariate NCT models.

```
def models_AIC_BIC(data, rep):
'''
This function fits input data set with the mixed Laplace and Noncentral student t distributions, and compute the AIC and BIC for each model.
:param data: input data set, numpy ndarray (n, 2)
:param rep: number of replications
:return: AIC and BIC for each replication
'''
N = data.shape[0]
nstock = data.shape[1]
k1, k2 = 12, 8  # number of estimated params for Mix2Lap2 and Bi-NCT respectively
AIC = np.zeros((rep, 2))
BIC = np.zeros((rep, 2))
X = np.zeros((N, 2))
for i in range(rep):
    the
    X[:,0] = data[:,j1]
    X[:,1] = data[:,j2]
    ll_mixlap = eng.MixLapEstimate(X, nargout=4)[-1]
    ll_bNCT = eng.MVNCT2estimation(X, nargout=4)[-1]
    AIC[i,0] = 2*(k1 - ll_mixlap)
    AIC[i,1] = 2*(k2 - ll_bNCT)
    BIC[i,0] = k1*np.log(N) - 2*ll_mixlap
    BIC[i,1] = k2*np.log(N) - 2*ll_bNCT
result = pd.DataFrame(np.concatenate((AIC, BIC), axis=1),
                      columns=["AIC_mxlap", "AIC_bNCT","BIC_mxlap", "BIC_bNCT" ])
return result
```

Figure 10 confirms our previous speculation that the bivariate NCT model would outperform the mixed Laplace, however, not significantly. According to AIC (BIC), the bivariate NCT is preferred in 88% (90%) of all the cases.

References

[Din16] Peng Ding. On the conditional distribution of the multivariate t distribution, 2016.

[NK05] Saralees Nadarajah and Samuel Kotz. Mathematical properties of the multivariate t distribution. *Acta Applicandae Mathematica*, 89(1):53–84, Dec 2005.

[Pao18] Marc S Paolella. *Fundamental Statistical Inference: A Computational Approach*. John Wiley & Sons, 2018.

[Pao19] Marc S Paolella. *Linear Models and Time-Series Analysis Regression, ANOVA, ARMA, and GARCH*. John Wiley & Sons, 2019.

[PPW21] Marc S. Paolella, Paweł Polak, and Patrick S. Walker. A non-elliptical orthogonal garch model for portfolio selection under transaction costs. *Journal of Banking Finance*, 125:106046, 2021.

A Code Listing

```
function [param,stderr,iters,loglik,Varcov] = MixLapEstimation(x, initvec)
% param: (mu11, mu21, mu12, mu22,
%         S1_11, S1_12, S1_22, S2_11, S2_12, S2_22,
%         b1, b2, lmda)
[nobs, d]=size(x);
if d~=2
    error('Not done yet, use EM instead.')
end

if d==2
% mu11, mu21, mu21, mu22,
% S1_11, S1_12, S1_22, S2_11, S2_12, S2_22,
% b1, b2, lmda
bound.lo = [-1 -1 -1 -1 ...
            0.01 -90 0.01 0.01 -90 0.01 ...
            0.01 0.01 0];
bound.hi = [1 1 1 1 ...
            90 90 90 90 90 90 ...
            20 20 1];
bound.which = [0 0 0 0 ...
               1 1 1 1 1 1 ...
               1 1 1];

    if nargin < 2
        initvec =[-0.8 -0.2 -0.8 -0.2 20 2 10 20 2 10 8 4 0.5];
    end
end

maxiter=500; 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) MixLaploglik(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 = MixLaploglik(param, x, bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param),bound,999); end
[nobs, d] = size(x);
Sig1 = zeros(d,d);
Sig2 = zeros(d,d);
mu1 = param(1:2); % Assume d=2
mu2 = param(3:4);

Sig1(1,1)=param(5); Sig1(1,2)=param(6);
Sig1(2,2)=param(7); Sig1(2,1)=Sig1(1,2);

```

```
Sig2(1,1)=param(8); Sig2(1,2)=param(9);
Sig2(2,2)=param(10); Sig2(2,1)=Sig2(1,2);

b1 = param(11);
b2 = param(12);
lmda = param(13);

if min([eig(Sig1), eig(Sig2)], [], "all") < 1e-10, ll = 1e5;
else
pdf = zeros(nobs,1);
for i = 1:nobs
    pdf(i) = lmda*mvlpdf(x(i,:), mu1, Sig1, b1) + ...
    (1-lmda)*mvlpdf(x(i,:), mu2, Sig2, b2);
end
llvec=log(pdf); ll=-mean(llvec); if isinf(ll), ll=1e5; end
end

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

function y = mvlpdf(x, mu, Sigma, b)
% Joint PDF of multivariate Laplace based on III. (14.31)
d = length(x);
x = reshape(x,d,1);
mu = reshape(mu,d,1);
m = (x-mu)' * (Sigma\(x-mu));
y = exp(log(2) + (b/2 - d/4) * log(m/2) + log(besselk(b-d/2, sqrt(2*m))) - ...
    log(det(Sigma))/2 - log(2*pi)*d/2 - gammaln(b));
```

```
function [param,stderr,iters,loglik,Varcov] = MVNCTEstimate(x, initvec)
% param: (mu1, mu2, gam1, gam2, v, S11, S12, S22)
[nobs, d]=size(x);
if d~=2
    error('Not done yet, use EM instead.')
end

if d==2
% mu1, mu2, gam1, gam2, v, S11, S12, S22
bound.lo = [-10 -10 -4 -4 1 0.01 -90 0.01];
bound.hi = [10 10 4 4 20 90 90 90];
bound.which = [1 1 1 1 1 1 1 1];

if nargin < 2
    initvec =[0.1 0.1 0.1 0.8 3 10 2 10];
end
end

maxiter=500; tol=1e-10;
MaxFunEvals = length(initvec)*maxiter;
```

```
opts = optimset('Display', 'iter', 'Maxiter', maxiter, 'TolFun', tol, ...
    'TolX', tol, 'MaxFunEvals', MaxFunEvals, 'LargeScale', 'Off');

[pout, fval, ~, theoutput, ~, hess] = ...
fminunc(@(param) MVNCTloglik(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 = MVNCTloglik(param, x, bound)
if nargin<3, bound=0; end
if isstruct(bound), param=einschrk(real(param),bound,999); end
d = length(x(1,:));
Sig = zeros(d,d);
mu = param(1:2); % Assume d=2
gam = param(3:4);
v = param(5);
Sig(1,1)=param(6); Sig(1,2)=param(7);
Sig(2,2)=param(8); Sig(2,1)=Sig(1,2);

if min(eig(Sig)) < 1e-10, ll = 1e5;
else
llvec = mvnctpdfln(x', mu, gam, v, Sig);
ll=-mean(llvec);
if isinf(ll), ll=1e5; end
end

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