Stable Paretian Distributions

February 12, 2026

Abstract

This report analyzes the symmetric stable Paretian distribution using numerical methods in MATLAB. In Part I, we compute the convolution of two independent stable random variables using four different methods: integral convolution, simulation, characteristic function inversion, and FFT. We compare their accuracy and computational speed. In Part II, we estimate the tail index α using the `stableregkw` estimator. We evaluate its performance for both stable and non-stable sums and verify the precision improvement with larger sample sizes using bootstrap confidence intervals. Finally, in Part III, we implement maximum likelihood estimation (MLE) for a mixture of two symmetric stable distributions and fit this model to the daily returns of 25 stocks from the Dow Jones Industrial Average index.

1 Introduction

The assignment is divided into three main parts. Part I covers the convolution of two independent symmetric stable distributions using four distinct computational approaches. Part II deals with the simulation-based estimation of the tail index using the `stableregkw` method. Part III focuses on the maximum-likelihood estimation for a single and a two-component symmetric stable model, with an application to DJIA daily returns data.

All computations were carried out in MATLAB R2024b, and the corresponding code snippets are shown directly within this report as well as in the appendix.

2 Part I: Convolution of Two Symmetric Stables

A univariate α–stable distribution $S_\alpha(\beta, c, \mu)$ is characterized by four parameters: the tail index $\alpha \in (0, 2]$, the skewness $\beta \in [-1, 1]$, the scale parameter $c > 0$, and the location parameter $\mu \in \mathbb{R}$. In this part of the assignment we use symmetric ($\beta = 0$), mean–zero ($\mu = 0$) parametrization for which the characteristic function can be written as

$$\varphi_X(t) = \exp(-c^\alpha |t|^\alpha), \tag{1}$$

and where $c = 1$ in part I.

We consider two independent random variables

$$X_1 \sim S_{\alpha_1}(0, 1, 0), \qquad X_2 \sim S_{\alpha_2}(0, 1, 0),$$

with tail indices $\alpha_1 = 1.3$ and $\alpha_2 = 1.7$, respectively. The stable family is closed under convolution only when the tail indices agree, i.e.,

$$X_1, X_2 \overset{\text{iid}}{\sim} S_\alpha \quad \Rightarrow \quad S = X_1 + X_2 \sim S_\alpha,$$

but if $\alpha_1 \neq \alpha_2$, S does not follow the stable distribution law. Still, the density f_S of

$$S = X_1 + X_2$$

is well defined and can be computed numerically. The first part of the assignment focuses on different approaches to compute the density of S. Our goal for this section is to compute and compare f_S over an appropriate grid (e.g. $x \in [-8, 8]$), using the following four approaches:

(a) the integral convolution formula based on the PDFs of the symmetric stable distributions, which are obtained via numerical inversion of their characteristic functions,
(b) simulation and kernel density estimation,
(c) numerical inversion of the characteristic function $\varphi_S(t)$,
(d) Fast Fourier Transform inversion of the characteristic function $\varphi_S(t)$.

Methods (a)–(d) should produce very similar densities, since they are simply different numerical

ways of computing f_S. The goal of this part is therefore to compare how these numerical methods behave and how close their results are, and to examine the shape of the distribution of S, especially in the tails.

2.1 Marginal Distributions

We calculate the PDFs for X_1 and X_2 on a grid x from -8 to 8. We use $N = 4000$ points, so the step size is $\Delta x \approx 0.004$.

The values from the `symstabpdf` function are normalized so that the discrete sum equals 1. If $p(x_i)$ is the value at point i (returned by the `stab` function), the normalized PDF $f(x_i)$ is:

$$f(x_i) = \frac{p(x_i)}{\sum_{j=1}^{N} p(x_j)\Delta x}. \tag{2}$$

We calculate the CDF by summing the normalized PDF values. For the k-th point on the grid, the CDF $F(x_k)$ as:

$$F(x_k) = \sum_{i=1}^{k} f(x_i)\Delta x. \tag{3}$$

This summation approximates the integral $\int_{-\infty}^{x} f(u)du$.

Figure 1 shows the results. The distribution with $\alpha = 1.3$ (red) has a lower peak and wider tails than the distribution with $\alpha = 1.7$ (blue), which is consistent with the theoretical properties of stable laws.

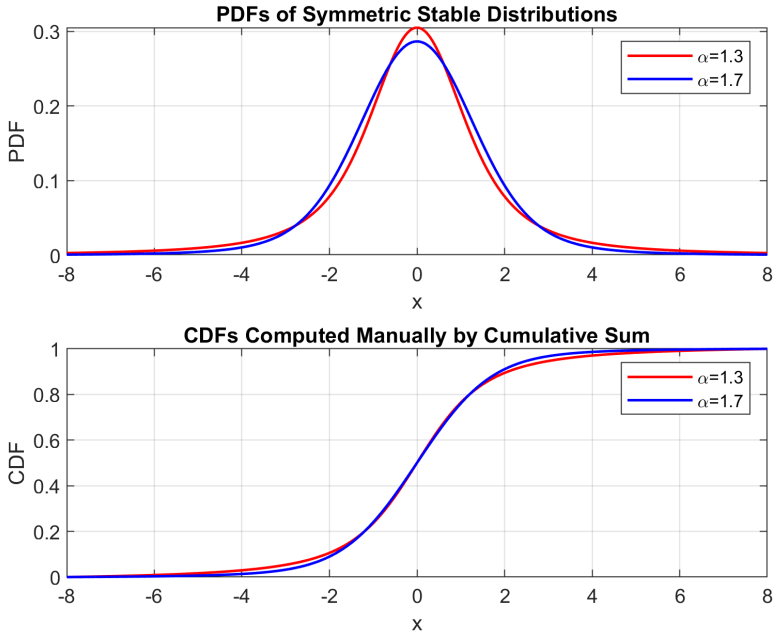


Figure 1: **Top**: PDFs of X_1 and X_2 on $[-8, 8]$. **Bottom**: CDFs computed by the cumulative sum of the PDF values.

2.2 (a) Analytical convolution / Convolution via integral

For independent X_1 and X_2, the density of the sum is

$$f_S(s) = \int_{-\infty}^{\infty} f_{X_1}(s-u)\, f_{X_2}(u)\, du.$$

We evaluate f_{X_1} and f_{X_2} on a grid using the numerical stable PDF routine `symstabpdf`, which implements the inversion of the characteristic function as described above, and then compute the convolution at a point s_i numerically with the built-in function `integral`, passing as integrand the product $f_{X_1}(s_i - u) f_{X_2}(u)$.

Matlab code

Below we show the Matlab code used to generate the densities f_{X_1}, f_{X_2}, and f_S.

Listing 1: Main routine for the convolution of two symmetric stable PDFs.

```
a1 = 1.3;
a2 = 1.7;
pdf1 = symstabpdf(range_s, a1);
pdf2 = symstabpdf(range_s, a2);

fS = convolution_sum_integral(range_s, a1, a2);

function fS = convolution_sum_integral(range_s, a1, a2)
    fS = zeros(size(range_s));

    for k = 1:length(range_s)
        s = range_s(k);
        integrand = @(x) symstabpdf(x, a1) .* symstabpdf(s - x, a2);
        fS(k) = integral(integrand, -Inf, Inf, 'RelTol', 1e-5, '
           AbsTol', 1e-8);
    end
end
```

Listing 2: Computation of the symmetric stable Paretian PDF.

```
function f = symstabpdf(x, alpha)
    xcol = x(:);
    fcol = asymstabpdf(xcol, alpha, 0, 0);
    f = reshape(fcol, size(x));
end
```

Listing 3: Asymmetric stable PDF via numeric integration (symmetric case $\beta = 0$ used).

```
function f = asymstabpdf(xvec, a, b, plotintegrand)
if nargin < 4
    plotintegrand = 0;
end
if a == 1
    error('Case alpha = 1 not implemented.');
end

xl = length(xvec);
f  = zeros(xl, 1);

for k = 1:xl
    x = xvec(k);
    if x < 0
        f(k) = stab(-x, a, -b);
    else
        f(k) = stab(x,  a,  b);
    end
end
end
```

Listing 4: Integral representation of the symmetric stable PDF.

```
function pdf = stab(x, a, b)
    integrand = @(t) exp(-t.^a) .* cos(x.*t);
    pdf = (1/pi) * integral(integrand, 0, Inf,'AbsTol',1e-8,'RelTol'
       ,1e-6);
end
```

Resulting Density

Figure 2 displays the probability density function $f_S(s)$ resulting from the numerical convolution.

2.3 (b) Simulation and kernel density estimation

Direct simulation using `stabgen` function

For the main simulation approach, we generate the stable random variables directly using `stabgen` implementation (Program Listing 75 in the lecture notes). This function implements the Chambers–Mallows–Stuck (CMS) algorithm, which provides efficient exact method for generating α-stable Paretian variates of the form $S_\alpha(\beta, c, \mu)$.

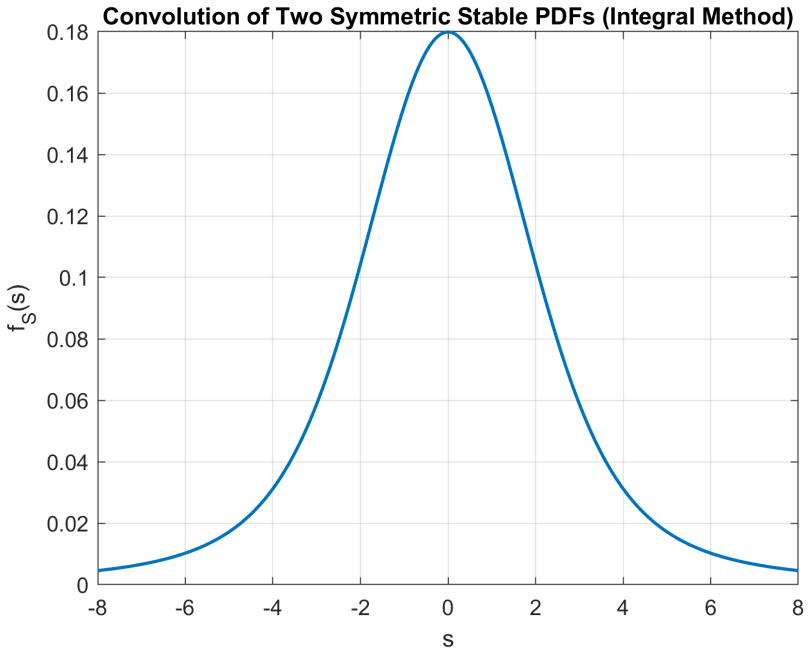


Figure 2: Density of the sum $S = X_1 + X_2$ computed via the integral convolution method.

We require independent samples

$$X_1 \sim S_{\alpha_1}(0, 1, 0), \qquad X_2 \sim S_{\alpha_2}(0, 1, 0),$$

and so we call the generator with parameters $\beta = 0$ (symmetry), $c = 1$ (unit scale), and $\mu = 0$ (zero location). The CMS method represents a stable random variable as a transformation of two independent draws:

$$V \sim \text{Unif}\Big(-\frac{\pi}{2}, \frac{\pi}{2}\Big), \qquad W \sim \text{Exp}(1),$$

so that the entire simulation reduces to transforming these two variables.

Once samples X_1 and X_2 are generated, we form

$$S = X_1 + X_2,$$

and estimate its density f_S using a Gaussian kernel estimator. More precisely, if we do $n_{\text{sim}} = 10^5$ simulations of the sum, we estimate

$$\widehat{f}_S(s) = \frac{1}{n_{\text{sim}}\, h} \sum_{i=1}^{n_{\text{sim}}} K\left(\frac{s - S_i}{h}\right),$$

where K is a Gaussian kernel and its bandwidth is $h = 0.15$.

Alternative approach: inverse-CDF simulation via the Probability Integral Transform

As a second, alternative approach we also approximate the density of

$$S = X_1 + X_2,$$

using the Probability Integral Transform (PIT). PIT states that if $U \sim \text{Unif}(0,1)$ and F_X is the CDF of X, then the inverse transform

$$X = F_X^{-1}(U) \tag{4}$$

generates a random variable with CDF F_X. Thus, by simulating two independent uniforms U_1 and U_2, and applying the respective inverse CDFs, we obtain samples

$$X_1 = F_{X_1}^{-1}(U_1), \qquad X_2 = F_{X_2}^{-1}(U_2),$$

and hence realizations of $S = X_1 + X_2$.

Since the stable CDF has no closed-form expression, we first approximate it numerically on a fine grid using

$$F_X(x_j) \approx \sum_{i \le j} f_X(x_i)\, \Delta x, \tag{5}$$

where f_X is the numerically computed PDF from (a) and Δx is the step. The inverse CDF F_X^{-1} is then implemented via linear interpolation:

$$F_X^{-1}(u) \;\approx\; \texttt{interp1}(F_X(x_j),\, x_j,\, u)\,. \tag{6}$$

Given $n_{\text{sim}} = 10^5$ samples of S, we obtain an approximation to its density as

$$\widehat{f}_S(s) = \frac{1}{n_{\text{sim}}\, h} \sum_{i=1}^{n_{\text{sim}}} K\!\left(\frac{s - S_i}{h}\right), \tag{7}$$

where K is a Gaussian kernel and its bandwidth is $h = 0.15$.

Matlab code

The code below shows how we generate X_1, X_2, and $S = X_1 + X_2$ using these methods, followed by the definition of the `stabgen` function itself.

Listing 5: Direct simulation of S = X1 + X2 using `stabgen`.

```
n_sim=1e5;
X1 = stabgen(n_sim, a1, 0, 1, 0, 10);
X2 = stabgen(n_sim, a2, 0, 1, 0, 20);

```

```
S = X1 + X2;

eval_points = linspace(-8, 8, 500);
[fS_sim2, s_grid2] = ksdensity(S, eval_points, 'Bandwidth',
   bandwidth);
```

Listing 6: CMS-based stable generator `stabgen.m`.

```
function x = stabgen(nobs, a, b, c, d, seed)

if nargin < 3, b = 0; end
if nargin < 4, c = 1; end
if nargin < 5, d = 0; end
if nargin < 6, seed = rand; end

z = nobs;

rng('default');
rng(floor(seed * 1e6), 'twister');

V = unifrnd(-pi/2, pi/2, 1, z);
W = exprnd(1, 1, z);

if a == 1
    x = (2/pi) * (((pi/2) + b*V) .* tan(V) ...
        - b .* log((W .* cos(V)) ./ ((pi/2) + b*V)));
    x = c * x + d - (2/pi) * d * log(c) * b;
else
    Cab = atan(b * tan(pi*a/2)) / a;
    Sab = (1 + b^2 * (tan(pi*a/2))^2)^(1/(2*a));

    A = sin(a*(V + Cab)) ./ (cos(V)).^(1/a);
    B0 = cos(V - a*(V + Cab)) ./ W;
    B = abs(B0).^((1-a)/a);

    x = Sab .* A .* (B .* sign(B0));
    x = c * x + d;
end

x = x(:);
end
```

The following code computes the PDFs and CDFs of X_1 and X_2 on a grid, constructs the inverse

CDF functions, simulates $S = X_1 + X_2$, and finally computes a kernel density estimate of f_S.

Listing 7: Grid-based PDF and CDF for the two symmetric stable distributions.

```
x = linspace(-8, 8, 4000);
dx = x(2) - x(1);
n_sim = 1e5;
bandwidth = 0.15;

pdf_x1 = symstabpdf(x, a1);
pdf_x1 = pdf_x1 / (sum(pdf_x1) * dx);

cdf_x1 = cumsum(pdf_x1) * dx;

pdf_x2 = symstabpdf(x, a2);
pdf_x2 = pdf_x2 / (sum(pdf_x2) * dx);

cdf_x2 = cumsum(pdf_x2) * dx;
```

Listing 8: Inverse CDF approximation using linear interpolation.

```
invF1 = @(u) interp1(cdf_x1, x, u, 'linear', 'extrap');
invF2 = @(u) interp1(cdf_x2, x, u, 'linear', 'extrap');
```

Listing 9: Simulation of X1, X2 via PIT and construction of S = X1 + X2.

```
U1 = rand(n_sim, 1);
U2 = rand(n_sim, 1);

X1 = invF1(U1);
X2 = invF2(U2);

S = X1 + X2;
```

Listing 10: Kernel density estimate of f_S based on the Monte Carlo simulation.

```
[fS_sim, s_grid] = ksdensity(S, 'Bandwidth', bandwidth);
```

Resulting densities

Figure 3 plots the results from both simulation methods. We can see a visual difference between the two lines, especially around the peak ($s = 0$).

To measure this difference, we calculate the L_∞ norm (maximum absolute error):

$$L_\infty = \max_s \left| \widehat{f}_{\mathrm{PIT}}(s) - \widehat{f}_{\mathrm{CMS}}(s) \right| \approx 1.29 \times 10^{-2}. \tag{8}$$

This value confirms what we see in the plot. The difference happens because the Inverse CDF (PIT) method uses linear interpolation on a fixed grid, which is an approximation. The `stabgen` method uses an exact formula. The error in the PIT method causes the difference in the peak.

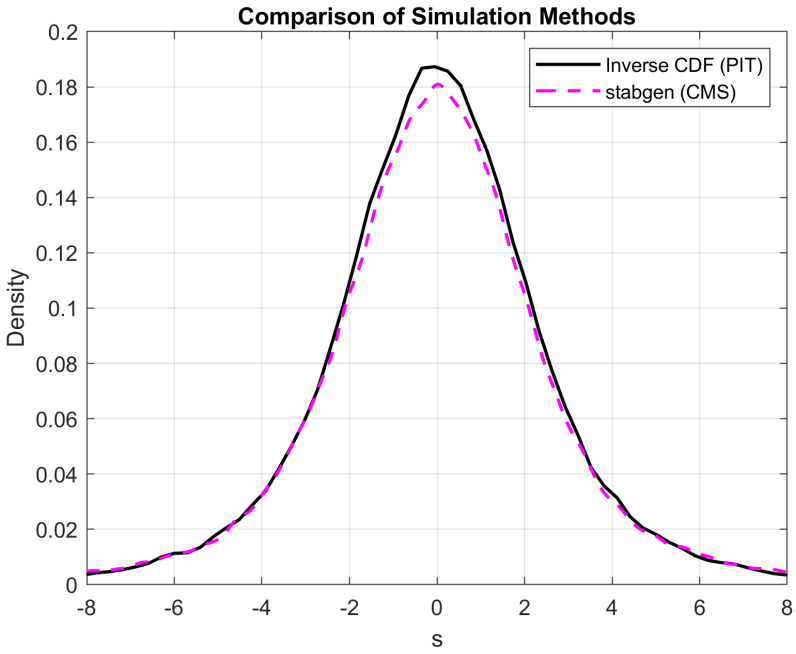


Figure 3: Comparison of simulation results. The Inverse CDF method (black) is higher at the peak than the `stabgen` method (magenta) because of interpolation errors.

2.4 (c) Numerical inversion of the characteristic function

A third approach to compute the density of

$$S = X_1 + X_2$$

is direct numerical inversion of the characteristic function of S. For independent random variables, the characteristic function of their sum is the product of their individual characteristic functions, so for $X_1 \sim S_{\alpha_1}(0,1,0)$ and $X_2 \sim S_{\alpha_2}(0,1,0)$ we have

$$\varphi_S(t) = \varphi_{X_1}(t)\,\varphi_{X_2}(t) = \exp(-|t|^{\alpha_1})\ \exp(-|t|^{\alpha_2}) = \exp(-|t|^{\alpha_1} - |t|^{\alpha_2})\,. \tag{9}$$

The density of S then admits the representation (due to the inversion formula)

$$f_S(s) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-ist}\varphi_S(t)dt = \frac{1}{2\pi}\int_{-\infty}^{\infty}(\cos(st) - i\sin(st))\exp(-|t|^{\alpha_1} - |t|^{\alpha_2})dt \tag{10}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty}\cos(st)\exp(-|t|^{\alpha_1} - |t|^{\alpha_2})dt - i\cdot 0 \tag{11}$$

$$= \frac{1}{\pi}\int_{0}^{\infty}\cos(st)\ \varphi_S(t)\,dt. \tag{12}$$

Equation (12) is convenient here because the characteristic function of a stable law decreases

very quickly, so the integral has little mass for large t. To apply (12) numerically, we discretize the t–axis on a fine equidistant grid $\{t_j\}$ and approximate the integral by the trapezoidal rule:

$$\widehat{f}_S(s) = \frac{1}{\pi} \sum_{j=1}^{N-1} \frac{\cos(st_j)\varphi_S(t_j) + \cos(st_{j+1})\varphi_S(t_{j+1})}{2} \Delta t. \tag{13}$$

The accuracy depends on how far we integrate and how fine the t-grid is. For stable distributions, the characteristic function drops quickly, so the integrand is already very small for large values of t.

Matlab code

The following routine implements (13). We use a dense t-grid to ensure accurate approximation.

Listing 11: Main call for the characteristic-function inversion method.

```
fS_cf_manual = symstab_sum_cf_inversion(range_s, a1, a2);
```

Listing 12: Characteristic-function inversion of S = X1 + X2 (symmetric case).

```
function fS = symstab_sum_cf_inversion(xvec, a1, a2)

phiS = exp( - (abs(t).^a1 + abs(t).^a2) );

fS = zeros(size(xvec));

for k = 1:length(xvec)
    x = xvec(k);

    integrand = cos(t * x) .* phiS;

    I = 0;
    for i = 1:(Nt-1)
        I = I + 0.5 * (integrand(i) + integrand(i+1)) * dt;
    end

    fS(k) = (1/pi) * I;
end
end
```

Resulting Density

We computed the density values using the numerical integration routine described above. Figure 4 shows the resulting probability density function of S.

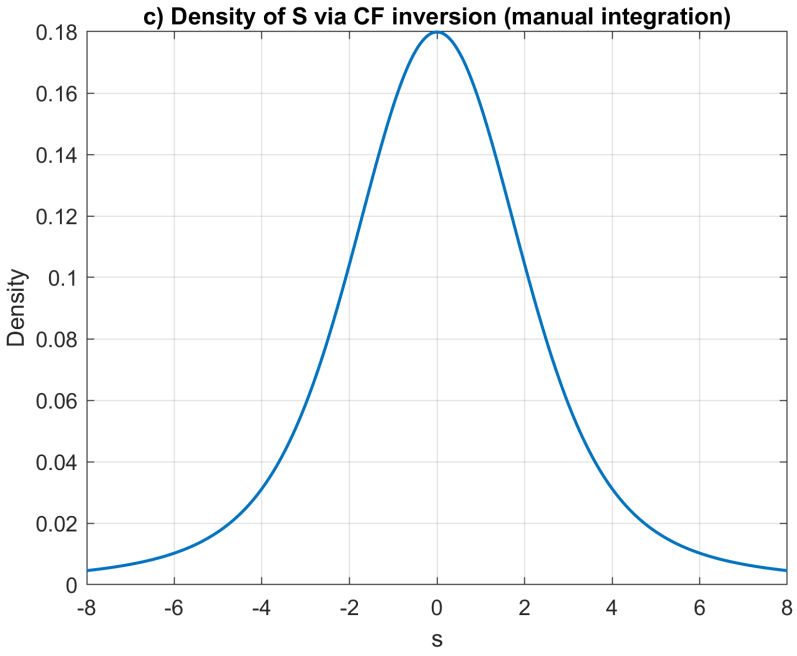


Figure 4: Density of $S = X_1 + X_2$ computed via numerical inversion of the characteristic function of S.

2.5 (d) FFT inversion of the characteristic function

The final method for computing the density of

$$S = X_1 + X_2$$

uses the Discrete Fourier transform (DFT), evaluated efficiently by the Fast Fourier transform (FFT). The idea is the same as in part (c), namely that the density of S can be recovered from the characteristic function of S through a Fourier inversion formula. Instead of approximating the integral directly, we place the inversion on a regular grid and apply the FFT, which yields the entire density vector in a single transform.

For a continuous density f_S and its characteristic function φ_S, a discretized version of the inversion formula can be written as

$$f_S(x_n) \;\approx\; \frac{1}{2\pi} \sum_{k=0}^{N-1} \varphi_S(t_k)\, e^{-i x_n t_k}\, \Delta t, \tag{14}$$

where $\{t_k\}$ is a uniform frequency grid and $\{x_n\}$ is the corresponding spatial grid satisfying the Fourier identity

$$\Delta x\, \Delta t = \frac{2\pi}{N}. \tag{15}$$

This ensures consistency between the two grids and allows the summation in (14) to be computed via an inverse FFT. For the stable distribution, the characteristic function is real and decreasing, so constructing the FFT on a symmetric grid around zero should yield good approximation. Once the transform is computed on the FFT grid, we use linear interpolation to obtain the density at the desired evaluation points $\{z_i\}$.

Matlab code

The following code implements the FFT inversion for $\varphi_S(t) = \exp(-|t|^{\alpha_1} - |t|^{\alpha_2})$ and returns the density evaluated at arbitrary z values. The implementation follows the standard FFT inversion structure: construct the grids, evaluate φ_S, apply the FFT, scale appropriately, and finally interpolate.

Listing 13: Main call for the FFT-based inversion.

```
z = -8:0.001:8;
pdf_fft = fft_stable_sum_density(z, a1, a2);
```

Listing 14: FFT-based c.f. inversion.

```
function pdf = fft_stable_sum_density(z, a1, a2)
pmax = 18;
step0 = 0.01;
p = 14;

maxz = max(abs(z(:)));
maxr = round((maxz/step0)) * 2^(p-1);

while ((maxr/step0 + 1) > 2^(p+1))
    p = p + 1;
end
if p > pmax, p = pmax; end
if maxr/step0 > 2^(p-1)
    step = maxz / (2^(p-1));
else
    step = step0;
end

z_sorted = sort(z(:));
[xgrid, bigpdf] = runthefft_stable(p, step, a1, a2);

pdf = wintp1(xgrid, bigpdf, z_sorted);
end
```

Listing 15: FFT inversion on an evenly spaced grid.

```
function [x, pdf] = runthefft_stable(p, h, a1, a2)

n  = 2^p;
n2 = n/2;

dt = h;
```

```
dx = 2*pi / (n * dt);

t = (-n2 : n2-1) * dt;
x = (-n2 : n2-1) * dx;

phi = exp(-abs(t).^a1 - abs(t).^a2);

f_unscaled = real(fftshift(ifft(ifftshift(phi))));
pdf = f_unscaled / (2*pi) * n * dt;

pdf = pdf / (sum(pdf) * dx);

end
```

Listing 16: Linear interpolation routine.

```
function r = wintp1(x, y, z)

r = zeros(size(z));
for i = 1:length(z)
    if z(i) < x(1)
        r(i) = y(1);
    elseif z(i) > x(end)
        r(i) = y(end);
    else
        j = find(x <= z(i), 1, 'last');
        if j < length(x)
            r(i) = y(j) + (y(j+1) - y(j)) * (z(i) - x(j)) / (x(j+1)
               - x(j));
        else
            r(i) = y(end);
        end
    end
end
end
```

Resulting Density

We calculated the density using the FFT algorithm code shown above. This method computes the entire density vector at once using the Fast Fourier Transform.

Figure 5 shows the resulting probability density function.

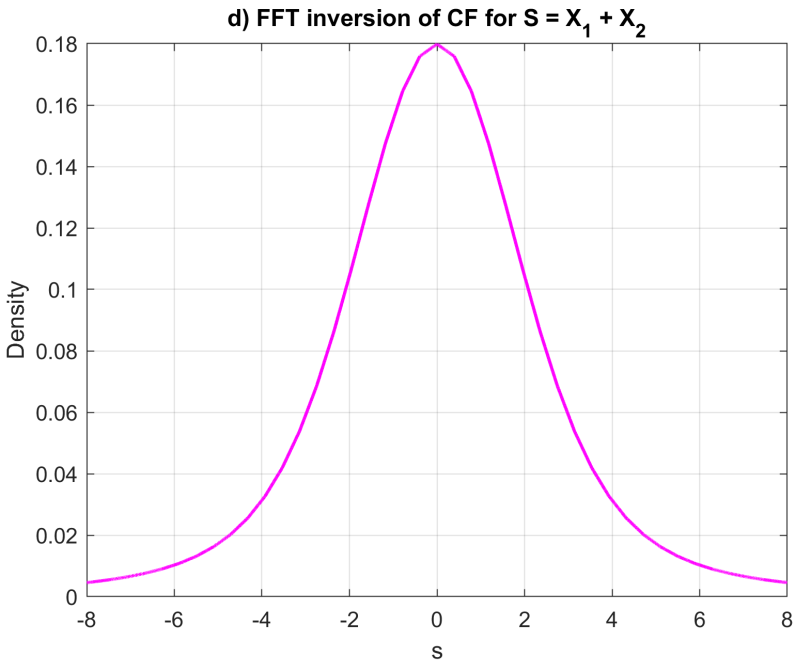


Figure 5: Density of $S = X_1 + X_2$ computed via FFT inversion of the characteristic function.

Comparison of resulting densities

Figure 6 plots the density estimates from all five methods.

The results show the difference between deterministic and stochastic approaches. The deterministic methods are the Analytical Convolution, CF Inversion, and FFT Inversion. These methods compute the density using numerical integration on fixed grids. They are exact and produce smooth curves without noise. On the other hand, stochastic methods (direct simulation with `stabgen` and PIT approach) approximate the density using random samples ($N = 10^5$) and Kernel Density Estimation. They show deviations from the density computed by the aforementioned deterministic methods due to variability in the random number generation.

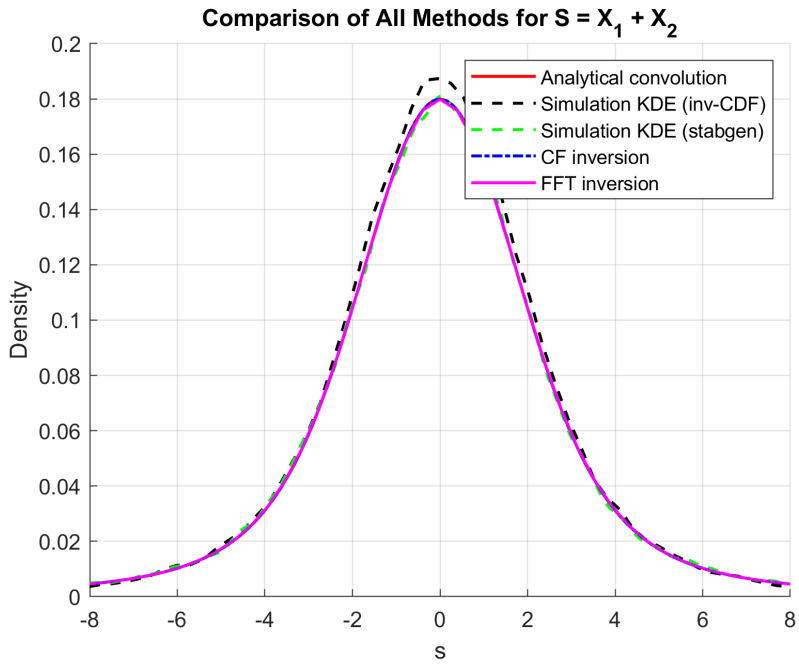


Figure 6: Comparison of density estimation methods. Deterministic methods overlap, while simulations show some noise.

Figure 7 shows the tail region ($4 \leq s \leq 8$).

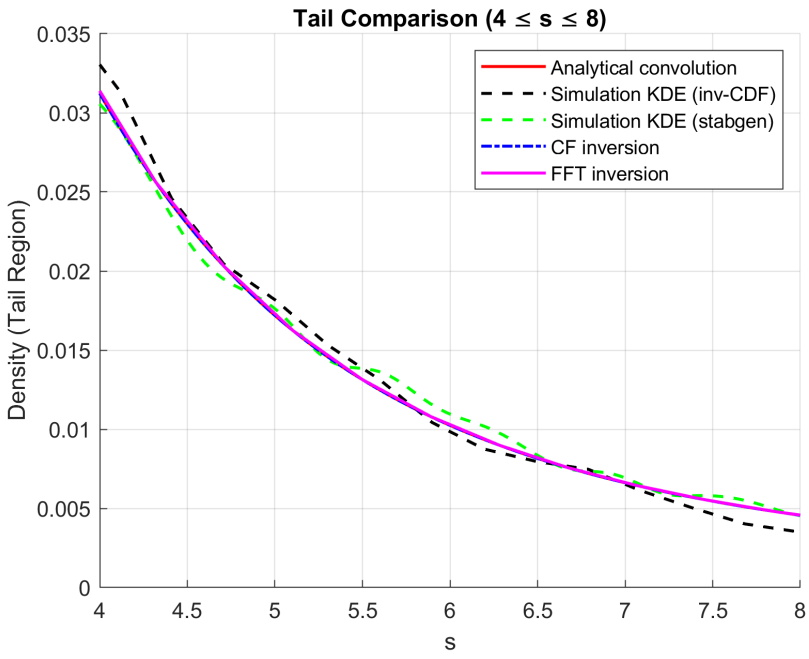


Figure 7: Tail behavior ($4 \leq s \leq 8$). Simulation methods are noisy in the tail because there are fewer data points in this region.

The deterministic methods remain smooth because they compute the exact value at every point. The simulation methods are noisy and rough in the tail region. This happens because the density is low (< 0.03), so very few of the $100,000$ random samples fall into the tail. With fewer points

in the tails, the Kernel Density Estimator has higher variance.

To conclude, we measured the execution time of each method using the `tic` and `toc` commands in MATLAB in order to find out differences in computational efficiency among different approaches. The Analytical convolution method was the slowest (1.1866 seconds). This is because the code must perform numerical integration over the entire range for every single point in the evaluation grid. In contrast, the Simulation using `stabgen` was the fastest method (0.0719 seconds). It uses the Chambers-Mallows-Stuck formula to generate random numbers directly and generate a realization of the symmetric stable with only simple functions (e.g. trigonometric, powers, etc.), which requires very few mathematical operations. The Simulation using the Inverse CDF (PIT) method was significantly slower (0.8136 seconds). This happens because the method first requires calculating the CDF numerically and then performing linear interpolation for every generated random number to find the inverse values. The Characteristic Function inversion took 0.2060 seconds. Finally, the FFT inversion took 0.4290 seconds. This method uses the Fast Fourier Transform algorithm to calculate the density values for the entire grid in a single step.

3 Part II: Estimation of the Tail Index via stableregkw

In this part we study the tail index of the sum of two independent symmetric stable random variables. Let

$$X_1 \sim S_{\alpha_1}(0,1,0), \qquad X_2 \sim S_{\alpha_2}(0,1,0),$$

be independent with $\alpha_1 \in \{1.2, 1.3, \ldots, 1.8\}$ and $\alpha_2 = 1.8$. The sum $S = X_1 + X_2$ is stable only when $\alpha_1 = \alpha_2$, in which case

$$S \sim S_{\alpha_1}(0, c, 0) \quad \text{with} \quad c = 2^{1/\alpha_1}.$$

For all other values of α_1, the distribution of S is not stable. We will still "pretend" it is stable and estimate its four parameters, especially the tail index $\widehat{\alpha}$.

The estimator used is the fast, nonparametric tail-index estimator `stableregkw`, which is designed for the i.i.d. case and avoids the large computational cost of maximum likelihood estimation. Given a sample $\{s_i\}_{i=1}^n$, function returns

$$(\widehat{\alpha}, \widehat{\beta}, \widehat{\sigma}, \widehat{\mu}) = \texttt{stableregkw}(s_1, \ldots, s_n). \tag{16}$$

We will examine how well $\widehat{\alpha}$ approximates the true tail of the sum, both when the sum is a true stable distribution ($\alpha_1 = \alpha_2$) and when it is not.

To evaluate the sampling variability, we repeat the simulation $N_{\text{sim}} = 500$ times, using sample sizes

$$n = 1000 \qquad \text{and} \qquad n = 10{,}000.$$

For each replication and each value of α_1, we record the corresponding estimate $\widehat{\alpha}$. This produces

one boxplot per α_1 value, illustrating how the estimator behaves for different combinations of tail indices. Then for each replication we construct a 90% nonparametric bootstrap confidence interval for α, based only on resampling the simulated sum $\{s_i\}$. If $[L_i, U_i]$ denotes the ith bootstrap CI, we record its length

$$\ell_i = U_i - L_i, \tag{17}$$

and generate histograms of $\{\ell_i\}$ for all settings.

Before turning to the Matlab implementation, we briefly note that there are two different ways in which the stable sums were simulated. The first approach uses the inverse–CDF method based on numerical evaluation of the symmetric stable density on a grid. The second approach, used in some of the Monte Carlo repetitions, relies on Matlab's `makedist('Stable',…)` generator. This method is fast and skips the numerical inversion step, making it practical for large numbers of bootstrap replications. Both approaches give i.i.d. stable draws, and it is informative to compare results obtained under both simulation schemes.

Matlab code

Below is the code used to do the simulations, estimation, and bootstrap calculations for all sample sizes and parameter combionations.

Listing 17: Simulation of S = X1 + X2 and estimation of stable parameters.

```
alpha1 = 1.2:0.1:1.8;
alpha2 = 1.8;

n_sim_small = 1000;
n_rep_small = 500;
n_grid = numel(alpha1);
rng(2);

alpha_est_small = zeros(n_rep_small,n_grid);
beta_est_small  = zeros(n_rep_small,n_grid);
sigma_est_small = zeros(n_rep_small,n_grid);
mu_est_small    = zeros(n_rep_small,n_grid);

for j = 1:n_grid
    a1 = alpha1(j);
    for i = 1:n_rep_small
        S = simulate_stable_sum(a1,alpha2,n_sim_small);
        [a_hat,b_hat,s_hat,m_hat] = stableregkw(S);

        alpha_est_small(i,j) = a_hat;
        beta_est_small(i,j)  = b_hat;
        sigma_est_small(i,j) = s_hat;
```

```
        mu_est_small(i,j)    = m_hat;
    end
end
```

Listing 18: Large-sample version (n = 10,000, rep = 50).

```
n_sim = 10000;
n_rep = 500;

alpha_est = zeros(n_rep,n_grid);
beta_est  = zeros(n_rep,n_grid);
sigma_est = zeros(n_rep,n_grid);
mu_est    = zeros(n_rep,n_grid);

rng(1);
for j = 1:n_grid
    a1 = alpha1(j);
    for i = 1:n_rep
        S = simulate_stable_sum(a1,alpha2,n_sim);
        [a_hat,b_hat,s_hat,m_hat] = stableregkw(S);

        alpha_est(i,j) = a_hat;
        beta_est(i,j)  = b_hat;
        sigma_est(i,j) = s_hat;
        mu_est(i,j)    = m_hat;
    end
end
```

Listing 19: Bootstrap CI length for each replication, n = 10,000.

```
B = 500;
conf = 0.90;
ci_len_mat = zeros(n_rep,n_grid);

for j = 1:n_grid
    a1 = alpha1(j);
    for i = 1:n_rep
        S = simulate_stable_sum(a1,alpha2,n_sim);
        ci_len_mat(i,j) = bootstrap_alpha_length(S,B,conf);
    end
end
```

Listing 20: Simulation of the stable sum via inverse CDF on a grid.

```
function S = simulate_stable_sum(a1,a2,n_sim)
```

```

x = linspace(-40,40,1000);
dx = x(2) - x(1);

pdf_x1 = symstabpdf(x,a1); pdf_x1 = pdf_x1/(sum(pdf_x1)*dx);
cdf_x1 = cumsum(pdf_x1)*dx;

pdf_x2 = symstabpdf(x,a2); pdf_x2 = pdf_x2/(sum(pdf_x2)*dx);
cdf_x2 = cumsum(pdf_x2)*dx;

invF1 = @(u) interp1(cdf_x1,x,u,'linear','extrap');
invF2 = @(u) interp1(cdf_x2,x,u,'linear','extrap');

U1 = rand(n_sim,1);
U2 = rand(n_sim,1);
X1 = invF1(U1);
X2 = invF2(U2);

S = X1 + X2;
end
```

Listing 21: Bootstrap CI length for alpha using stableregkw.

```
function ci_len = bootstrap_alpha_length(S,B,conf)

n = numel(S);
alpha_star = zeros(B,1);

for b = 1:B
    idx = randsample(n,n,true);
    S_star = S(idx);

    [a_hat,~,~,~] = stableregkw(S_star);
    alpha_star(b) = a_hat;
end

lower = quantile(alpha_star,(1-conf)/2);
upper = quantile(alpha_star,1-(1-conf)/2);

ci_len = upper - lower;
end
```

Listing 22: Alternative simulation approach using makedist for bootstrap CIs.

```
1
2  B_2     = 500;
3  conf_2  = 0.90;
4  ci_len_mat_2 = zeros(n_rep_2, n_grid);
5
6  for j = 1:n_grid
7      a1 = alpha1(j);
8      for i = 1:n_rep_2
9
10         pd1 = makedist('Stable','alpha',a1,'beta',0,'gam',1,'delta'
               ,0);
11         pd2 = makedist('Stable','alpha',alpha2,'beta',0,'gam',1,'
               delta',0);
12
13         X1 = random(pd1, n_sim_small_2, 1);
14         X2 = random(pd2, n_sim_small_2, 1);
15         S  = X1 + X2;
16
17         ci_len_mat_2(i,j) = bootstrap_alpha_length(S, B_2, conf_2);
18     end
19 end
```

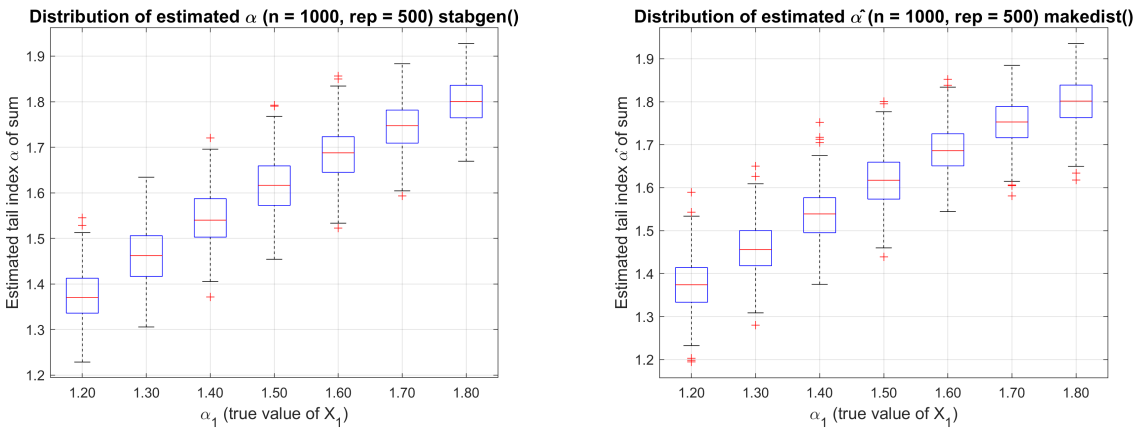
Results for Tail Index Estimation

We simulated the sum of two independent symmetric stable variables $S = X_1 + X_2$ with $\alpha_2 = 1.8$ fixed and α_1 ranging from 1.2 to 1.8 with increments of 0.1. We estimated the tail index $\widehat{\alpha}$ using the `stableregkw` estimator across 500 replications for sample sizes of $n = 1000$ and $n = 10,000$.

We implemented two different simulation approaches to generate the data. The first approach uses `stabgen` function, which implements the Chambers-Mallows-Stuck algorithm. The second approach uses MATLAB's built-in `makedist` function. We compared these two implementations to ensure the robustness of our results.

Small Sample Size ($n = 1000$)

Figure 8 compares the results for $n = 1000$ with 500 replications. The left plot uses `stabgen` and the right plot uses `makedist`.



(a) `stabgen` method (b) `makedist` method

Figure 8: Distribution of $\widehat{\alpha}$ for $n = 1000$. The two simulation methods produce statistically equivalent results.

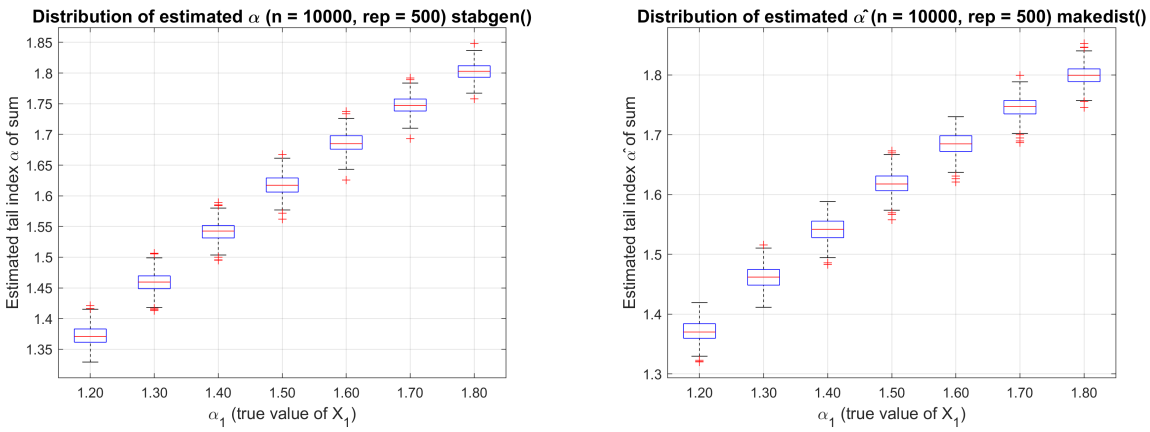
Comparing the two plots side by side, we see that the results are nearly identical. The position of the medians and the size of the boxes match for every value of α_1.

We look specifically at the case where $\alpha_1 = 1.8$. Since we fixed $\alpha_2 = 1.8$, we are summing two independent stable variables with the same index. Theory states that this sum is also a stable distribution with $\alpha = 1.8$. Therefore, our estimated values should be centered very close to 1.8. The numerical results confirm this. For the `stabgen` method, the median estimate is 1.80027, with values ranging from 1.66945 to 1.92791. The `makedist` method gives nearly the same result, with a median of 1.80142. Both medians are practically equal to the theoretical value.

When $\alpha_1 < 1.8$, the sum is not stable. The estimator returns a value between the two input indices. For example, at $\alpha_1 = 1.2$, the result is around 1.46. The estimator tries to fit a single stable distribution to the data and finds a middle ground between the two components.

Large Sample Size ($n = 10,000$)

Figure 9 compares the results for $n = 10,000$. The left plot uses `stabgen` and the right plot uses `makedist`.



(a) `stabgen` method (b) `makedist` method

Figure 9: Distribution of $\widehat{\alpha}$ for $n = 10,000$. The boxes are much narrower than in the small sample case.

Comparing these plots to the small sample size results, the most obvious difference is that the boxes have reduced height. The spread of the data is significantly reduced. This happens because we increased the sample size from 1,000 to 10,000. With ten times more data points, the estimation becomes more precise and the variance decreases.

We look again at the stable case where $\alpha_1 = 1.8$. Because the sample size is larger, we expect the estimates to be even closer to the true value of 1.8. The numerical results confirm this increased precision. For `stabgen`, the median is 1.80287, and the values only range from 1.75821 to 1.84830. For `makedist`, the median is 1.79940, with a range from 1.74537 to 1.85276. Both are very close to the true theoretical one.

For the non-stable cases where $\alpha_1 < 1.8$, the estimator still returns a value between the two indices. However, because the main cluster of estimates is so tight, we see more outliers (red crosses) outside the whiskers compared to the small sample plots.

Bootstrap Confidence Intervals

For the final part of the analysis, we computed 90% confidence intervals (CI) for the tail index α. We simulated the data using the `makedist` function. We chose this method because calculating bootstrap intervals requires repeating the estimation 250,000 times (500 trials $\times$ 500 bootstraps). The `makedist` function is significantly faster than the other method, and as demonstrated in the previous section, it produces statistically equivalent results.

We calculated the length of the confidence interval for 500 replications at each grid point (α_1, α_2) for both $n = 1000$ and $n = 10,000$.

Results

Figure 10 shows the histograms of the CI lengths for the small sample size ($n = 1000$), while Figure 11 shows the results for the large sample size ($n = 10,000$).

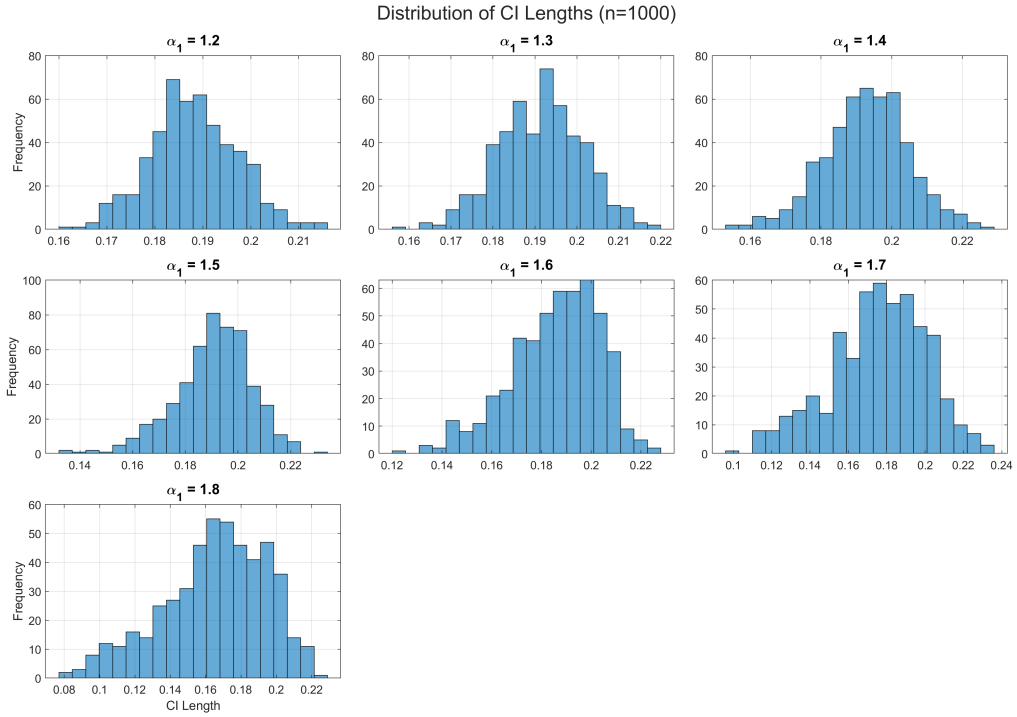


Figure 10: Histograms of the length of 90% bootstrap confidence intervals for the small sample size ($n = 1000$).

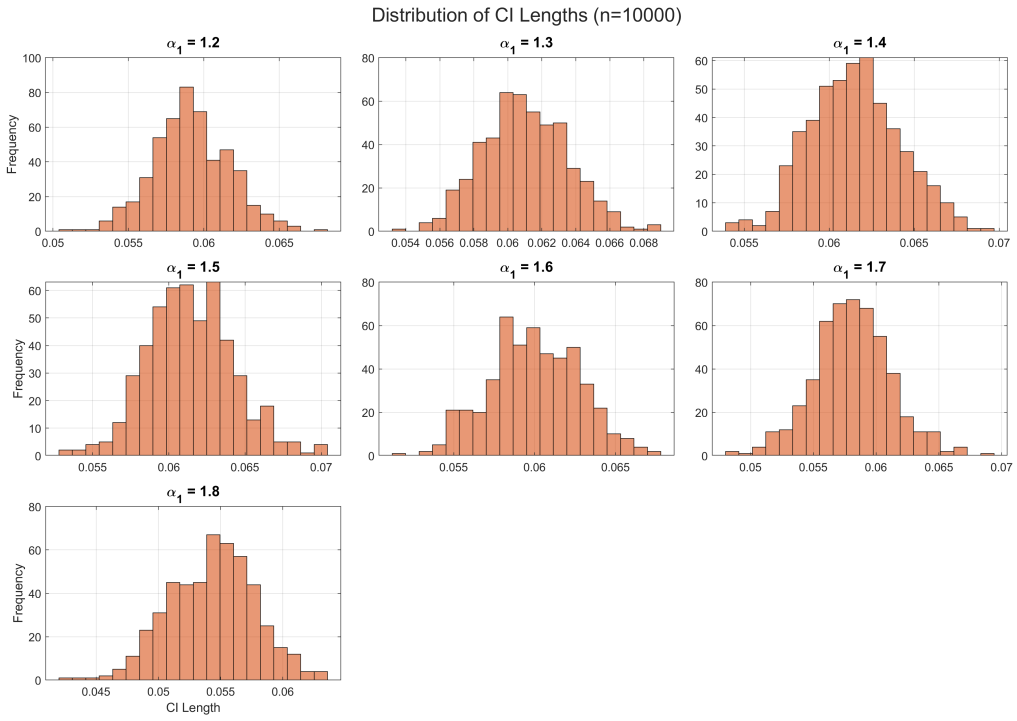


Figure 11: Histograms of the length of 90% bootstrap confidence intervals for the large sample size ($n = 10,000$). Note the x-axis scale is much smaller compared to Figure 10.

We observe from the histograms that as the sample size n increases, the average length of the confidence intervals decreases significantly. For each simulation, we calculated the length L of the 90% confidence interval as the difference between the upper and lower bootstrap quantiles.

In our simulation, we increased the sample size by a factor of 10 (from 1000 to 10,000). We found that the ratio of the average lengths across all α_1 values was approximately 3.10. This means that increasing the sample size by 10 times causes the interval length to decrease by a factor of around 3.

4 Part III: Maximum Likelihood Estimation of a Symmetric Stable and Mixture of Symmetric Stable Models

We first implement maximum likelihood estimation (MLE) for an IID symmetric stable model. The goal is to estimate the parameters $(\alpha, \beta = 0, c, \mu)$ of the symmetric stable distribution. Second, we implement maximum likelihood estimation for a two-component mixture model of symmetric stable distributions with parameters $(\alpha_1, \beta_1 = 0, c_1, \mu_1)$ and $(\alpha_2, \beta_2 = 0, c_2, \mu_2)$ respectively, as well as a mixture coefficient $p \in (0, 1)$. Finally, we fit the two-component mixture model to the time series of log-returns of 25 different assets from the Dow Jones Industrial Average index for a period of approximately 32 years.

4.1 MLE for the symmetric stable distribution

A symmetric stable random variable $X \sim S_\alpha(\beta = 0, c, \mu)$ has density f_X, which satisfies

$$f_X(x; \alpha, \mu, c) = \frac{1}{c} f_Z\left(\frac{x - \mu}{c}; \alpha\right),$$

where $Z \sim S_\alpha(0, 1, 0)$ is a symmetric stable random variable with location 0 and scale 1. The characteristic function of Z satisfies

$$\varphi_Z(t; \alpha) = \exp(-|t|^\alpha),$$

and the PDF f_Z of Z can be obtained from the inversion formula as

$$f_Z(z; \alpha, 0, 1) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-itz} \varphi_Z(t; \alpha) dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-itz - |t|^\alpha} dt.$$

In our project, we use the built-in Matlab implementation for the PDF of the symmetric stable with

```
makedist('Stable','alpha',alpha,'beta',0,'gam',c,'delta',mu);
```

and

```
pdf('Stable',xvec,alpha,0,c,mu);
```

where the parameter $'\text{gam}'$ is c and $'\text{delta}'$ is μ.

For IID observations $\{x_i\}_{i=1}^N$ from the symmetric stable distribution, we can write the likelihood

as

$$\mathcal{L}(\alpha, \mu, c; \mathbf{x}) = f_{\mathbf{X}}(\mathbf{x}; \alpha, \mu, c) = \prod_{i=1}^{N} f_X(x_i; \alpha, \mu, c).$$

The log-likelihood is then given by

$$\ell(\alpha, \mu, c; \mathbf{x}) = \log \mathcal{L}(\alpha, \mu, c; \mathbf{x}) = \sum_{i=1}^{N} \log f_X(x_i; \alpha, \mu, c).$$

In our implementation we define a custom function, which returns the negative of the log-likelihood:

```
custnloglf = @(params,data,cens,freq) ...
    - sum(log(pdf('Stable',data,params(1),0,params(2),params(3))));
```

which we then pass to the mle() function in Matlab for the actual MLE fitting:

```
estimated_parameters = mle(xvec,'nloglf',custnloglf,'Start',[1.5,
   1.5, 2.5],'options',opts);
opts = statset('MaxIter',1000,'MaxFunEvals',2000,'TolX',1e-9, ...
        'TolFun',1e-9,'Display','iter');
```

Implementation

To verify our implementation, we generated a sample of $N = 10,000$ IID realizations from a symmetric stable distribution. We set the true parameters to $\alpha = 1.7$, scale $c = 3$, and location $\mu = 2$.

We performed the Maximum Likelihood Estimation using the custom log-likelihood function defined above. We initialized the optimization with the initial guess vector $[1.5, 1.5, 2.5]$.

Figure 12 displays the result. The red line represents the true probability density function used to generate the data. The blue dashed line represents the density function plotted using the estimated parameters returned by the MLE. The estimated values are $\widehat{\alpha} = 1.707$, $\widehat{c} = 3.015$, and $\widehat{\mu} = 2.004$. These are very close to the true values.

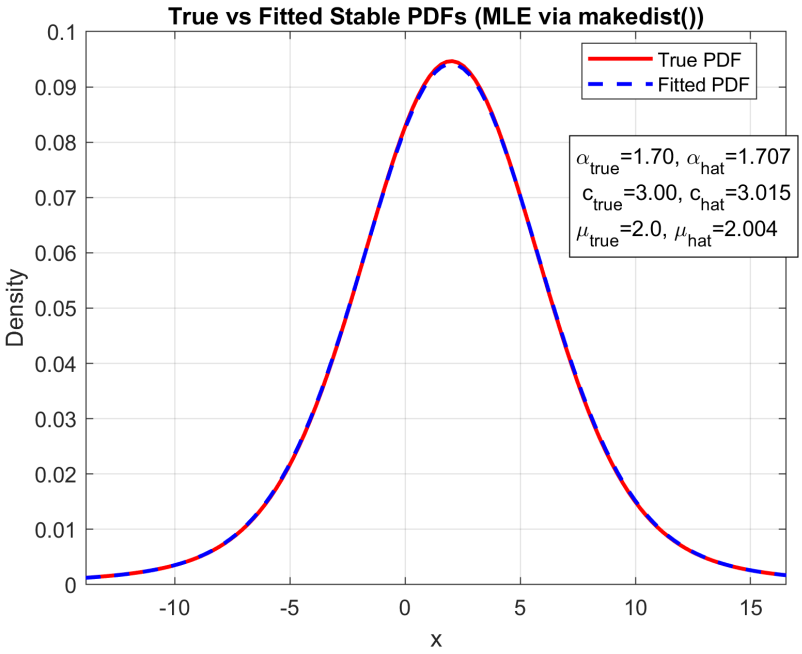


Figure 12: Comparison of the true stable PDF (red) and the fitted PDF (blue dashed) obtained via MLE. The estimated parameters match the true parameters closely.

4.2 MLE for a two-component mixture of symmetric stable random variables

Provided we know the PDFs of two symmetric stable random variables X_1, X_2, namely f_{X_1} and f_{X_2}, with parameters (α_1, μ_1, c_1) and (α_2, μ_2, c_2) respectively, we can define the mixture of X_1 and X_2 with coefficient $p \in (0, 1)$ as the random variable X with PDF

$$f_X(x) = p f_{X_1}(x; \alpha_1, \mu_1, c_1) + (1-p) f_{X_2}(x; \alpha_2, \mu_2, c_2).$$

Note that in general ($p \in (0, 1)$) the mixture of two symmetric stable random variables is not a symmetric stable random variable.

Once again, for IID observations $\{x_i\}_{i=1}^N$ of the random variable X, we can write the likelihood as

$$\mathcal{L}(\alpha_{1,2}, \mu_{1,2}, c_{1,2}; \mathbf{x}) = f_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^{N} f_X(x_i) = \prod_{i=1}^{N} \left(p f_{X_1}(x_i; \alpha_1, \mu_1, c_1) + (1-p) f_{X_2}(x_i; \alpha_2, \mu_2, c_2)\right)$$

and the log-likelihood as

$$\ell(\alpha, \mu, c; \mathbf{x}) = \log \mathcal{L}(\alpha, \mu, c; \mathbf{x}) = \sum_{i=1}^{N} \log \left(p f_{X_1}(x_i; \alpha_1, \mu_1, c_1) + (1-p) f_{X_2}(x_i; \alpha_2, \mu_2, c_2)\right).$$

We again use a custom function for the negative log-likelihood, namely

```
custnloglf = @(params,data,cens,freq) ...
- sum(log(params(1) * pdf('Stable',data,params(2),0,params(3),...
```

```
    params(4)) + (1 - params(1)) * pdf('Stable',data,params(5),0,
       params(6),params(7))),'omitnan');
```

The correctness of our code is again verified via simulating 10000 pairs of IID symmetric stable r.v.s with parameters $\alpha_1 = 1.3$, $\mu_1 = -2$, $c_1 = 1$, $\alpha_2 = 1.7$, $\mu_2 = 1.5$, $c_2 = 2.5$ respectively, forming their mixture with coefficient $p = 0.4$, and finally fitting our MLE model with the mle() function from Matlab:

```
opts = statset('MaxIter', 1000, 'MaxFunEvals', 2000, 'TolX'...
,1e-9, 'TolFun',1e-9,'Display','iter');
estimated_parameters = mle(xvec,'nloglf',custnloglf,'Start',[0.5,
   1.5, 1, -1.5, 1.5, 2, 2],'Options',opts);
```

The true and fitted PDFs are plotted in Figure 13.

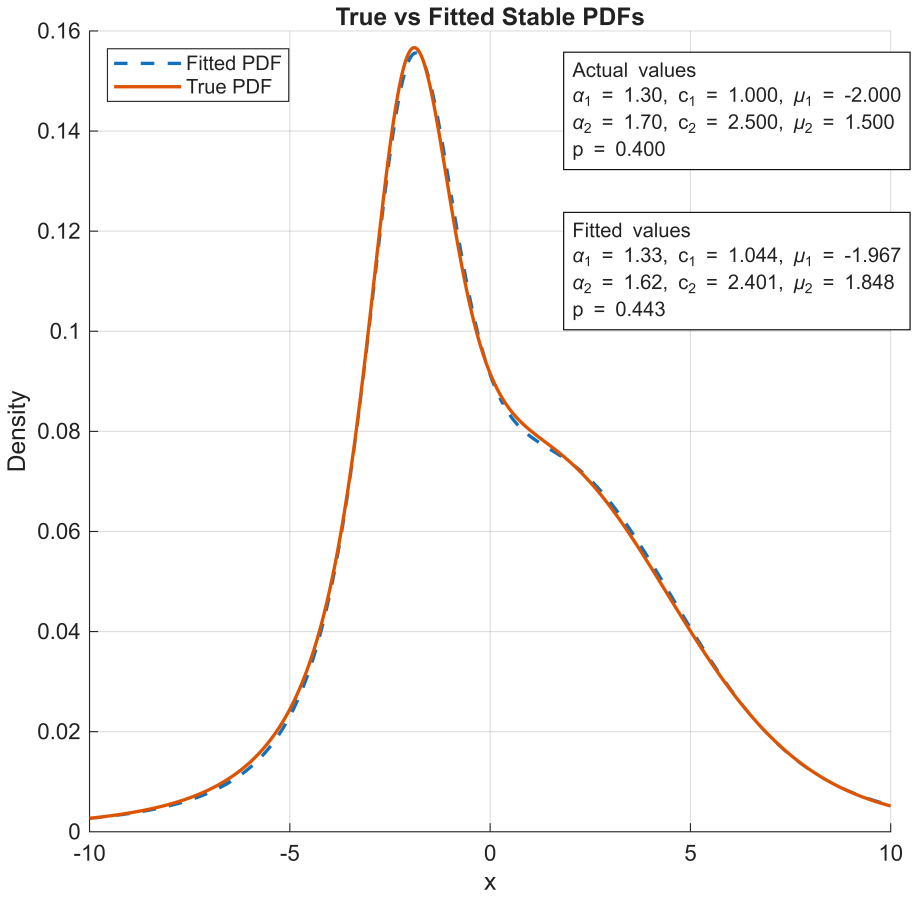


Figure 13: Comparison of the fitted PDF (via MLE) of a two-component mixture (blue) and true PDF of the mixture (orange). The estimated parameters match the true parameters closely.

4.3 Fitting the two-component mixture model to stock returns

The data provided for the project contains the time series of log-returns for 25 assets from the Dow Jones Industrial Average index for a period of 32 years. They were first plotted to see if modeling them with the symmetric stable would be feasible. The data showed to be highly

centered around zero and exhibiting heavy-tailed behavior. We fitted our mixture model by passing to the mle() function our custom negative log-likelihood function from Subsection 4.2:

```
for j=1:n_assets
    pars = mle(ret(:,j),'nloglf',custnloglf,'Start',[0.7, 1.8, std(
       ret(:,j)), mean(ret(:,j)), 1.8, std(ret(:,j)), mean(ret(:,j))
       ],'options',opts);
    save(sprintf("Estimated_parameters_%d.mat", j), 'pars');
end
```

The options for the optimization were specified as

```
opts = statset('MaxIter',1000,'MaxFunEvals',2000,'TolX',1e-9,...
        'TolFun',1e-9,'Display','iter');
```

The optimization process took a significant amount of time. Initially there were some numerical problems, arising mainly due to wrong selection of initial parameters, but they were fixed.

Below we present the fitted PDFs, as well the actual returns series for all 25 DJIA assets.

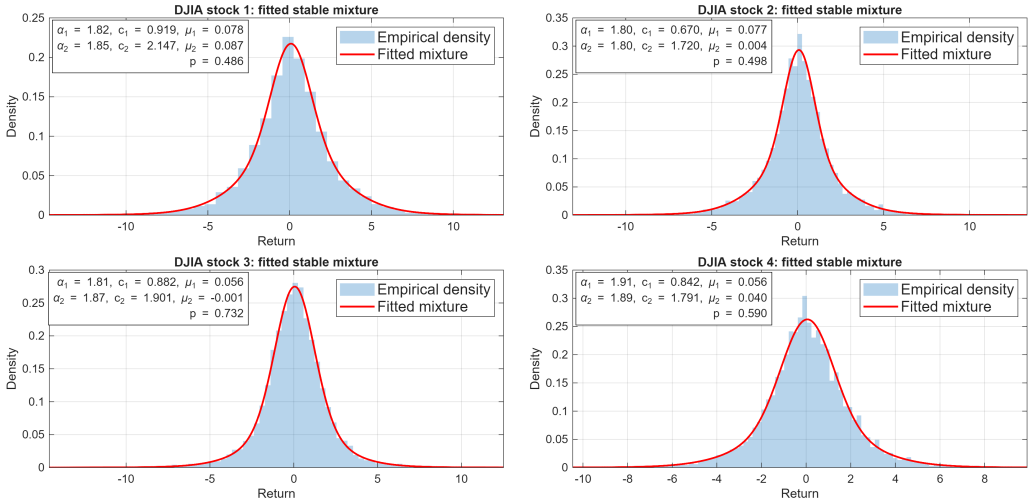


Figure 14: Estimated parameters, fitted PDFs and histograms of assets 1-4

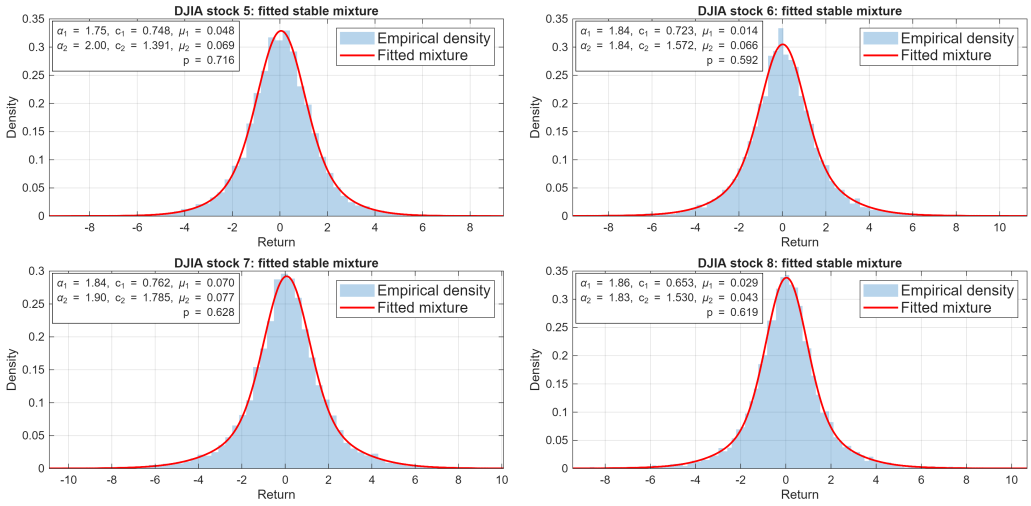


Figure 15: Estimated parameters, fitted PDFs and histograms of assets 5-8

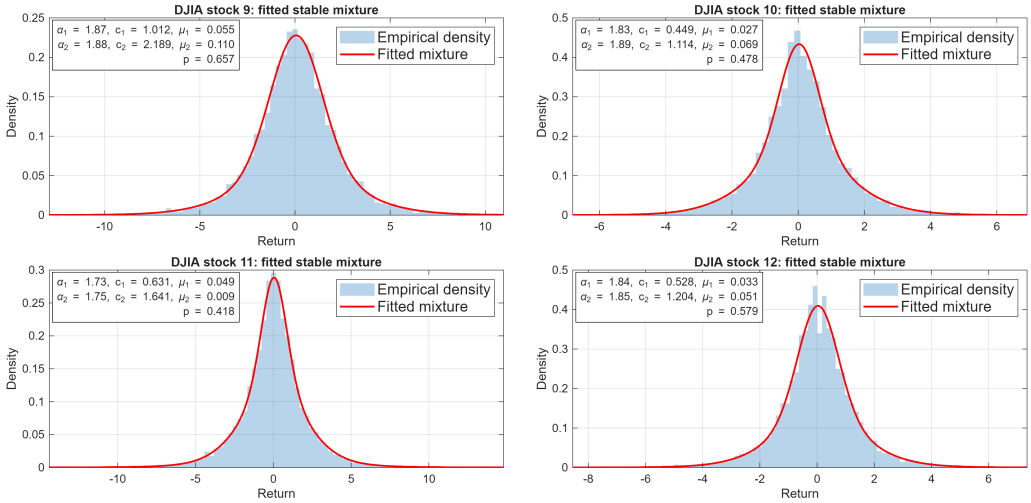


Figure 16: Estimated parameters, fitted PDFs and histograms of assets 9-12

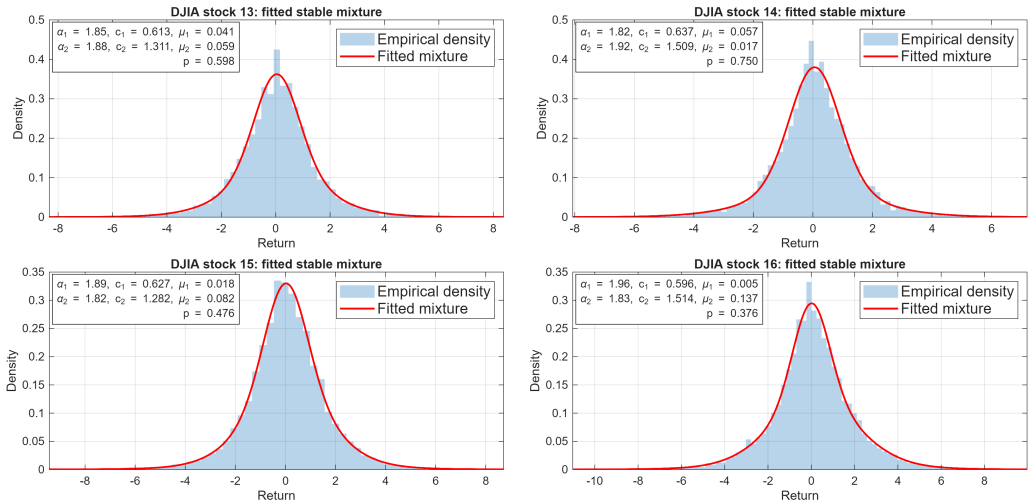


Figure 17: Estimated parameters, fitted PDFs and histograms of assets 13-16

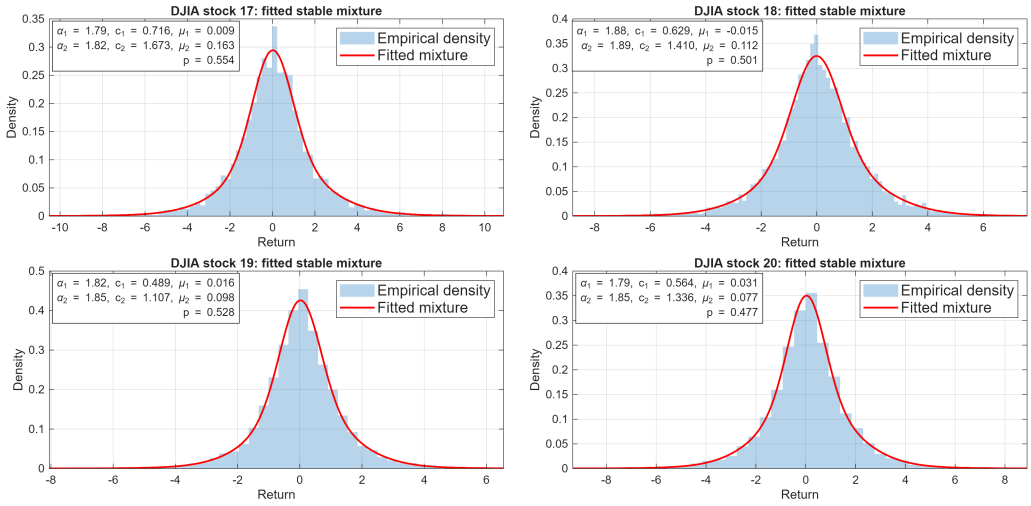


Figure 18: Estimated parameters, fitted PDFs and histograms of assets 17-20

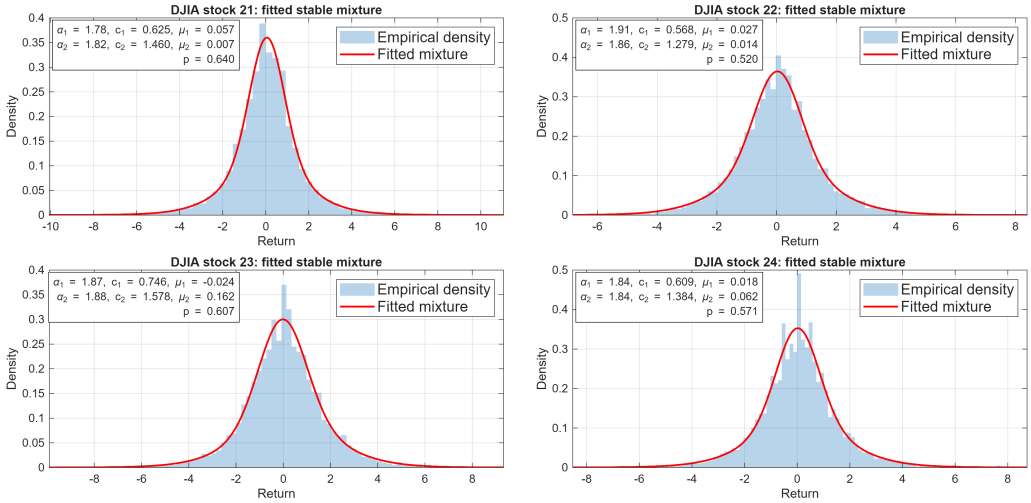


Figure 19: Estimated parameters, fitted PDFs and histograms of assets 21-24

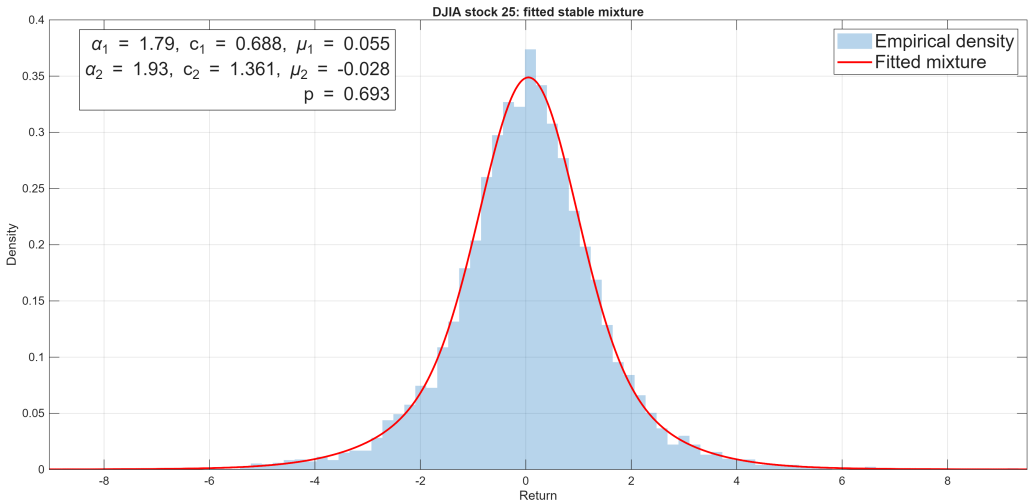


Figure 20: Estimated parameters, fitted PDF and histogram of asset 25

We observe that for most of the assets, a two-component mixture of symmetric stable distributions fits the returns series well. The estimated parameter values seem reasonable for most of the assets, with the means being slightly positive (as expected) and the α_1 and α_2 values being close to each other. There were no numerical problems when running the final MLE fitting of the mixture models. Finally, we see that the fits could be improved (by a more sophisticated model) in cases where returns are very concentrated around 0.

5 Appendix: Matlab codes not listed above

5.1 Part I

Listing 23: Calculation of L_∞ norm between the two simulation methods.

```
fS_sim1_interp = interp1(s_grid1, fS_sim1, s_grid2, 'linear', 0);

L_inf_sim = max(abs(fS_sim1_interp - fS_sim2));
```

5.2 Part II

Listing 24: Helper function for bootstrap confidence interval length

```
function ci_len = bootstrap_alpha_length(S, B, conf)
    n = length(S);
    alphas = zeros(B, 1);

    for k = 1:B
        indices = randi(n, n, 1);
        S_boot = S(indices);
        [a, ~, ~, ~] = stableregkw(S_boot);
        alphas(k) = a;
    end

    lower = quantile(alphas, (1-conf)/2);
    upper = quantile(alphas, 1 - (1-conf)/2);
    ci_len = upper - lower;
end
```

5.3 Part III

Listing 25: 10000 simulations of a symmetric stable random variable; true and fitted PDFs via maximum likelihood

```
load carsmall;
```

```
load carbig;
alpha = 1.7;
c = 2;
mu = 3;
n = 10000;
pd = makedist('Stable','alpha',alpha,'beta',0,'gam',c,'delta',mu);
xvec = random(pd,n,1);

custnloglf = @(params,data,cens,freq) ...
    - sum(log(pdf('Stable',data,params(1),0,params(2),params(3))));

opts = statset('MaxIter', 1000, 'MaxFunEvals', 2000, 'TolX'...
,1e-9, 'TolFun',1e-9, 'Display','iter');

estimated_parameters = mle(xvec,'nloglf',custnloglf,'Start',[1.5,
   1.5, 2.5],'options',opts);

alpha_fitted = estimated_parameters(1);
c_fitted = estimated_parameters(2);
mu_fitted = estimated_parameters(3);

pd_fitted = makedist('Stable','alpha',alpha_fitted,'beta',0,'gam',
   c_fitted,'delta',mu_fitted);

fitted_pdf = pdf(pd_fitted, xvec);

xgrid = linspace(-15, 15, 500);
plot(xgrid, pdf(pd_fitted, xgrid), 'LineWidth', 2,'LineStyle', '--')
   ;
hold on;
plot(xgrid, pdf(pd, xgrid), 'LineWidth', 1);
grid on;
xlabel('x');
ylabel('Density');
title('True vs Fitted Stable PDFs (MLE via makedist())');
legend('Fitted PDF', 'True PDF','Location', 'northeast');

text(0.8, 0.8, ...
    sprintf(['Actual values\n', '\\alpha = %.2f, c = %.3f, \\mu =
       %.3f\n'], ...
            alpha, c, mu), ...
    'FontSize', 9, 'BackgroundColor', 'w','Units', 'normalized', ...
```

```
    'EdgeColor', 'k');
text(0.8, 0.6, ...
    sprintf(['Fitted values\n', '\\alpha = %.2f, c = %.3f, \\mu =
       %.3f\n'], ...
            alpha_fitted, c_fitted, mu_fitted), ...
    'FontSize', 9, 'BackgroundColor', 'w','Units', 'normalized', ...
    'EdgeColor', 'k');
exportgraphics(gcf, 'PlotMLEFitTask3.png', 'Resolution', 1200);
```

Listing 26: 10000 simulations of a two-component mixture of symmetric stables and true and fitted PDFs via maximum likelihood

```
load carsmall;
load carbig;
alpha1 = 1.3;
alpha2 = 1.7;
mu1 = -2;
c1 = 1;
mu2 = 1.5;
c2 = 2.5;
n = 10000;
pd1 = makedist('Stable','alpha',alpha1,'beta',0,'gam',c1,'delta',mu1
   );
pd2 = makedist('Stable','alpha',alpha2,'beta',0,'gam',c2,'delta',mu2
   );
p = 0.4;
U = rand(n,1) <= p;
n1 = sum(U);
n2 = n - n1;
xvec = zeros(n,1);
xvec(U) = random(pd1,n1,1);
xvec(~U) = random(pd2,n2,1);

custnloglf = @(params,data,cens,freq) ...
    - sum(log(params(1) * pdf('Stable',data,params(2),0,params(3),
       params(4)) ...
    + (1 - params(1)) * pdf('Stable',data,params(5),0,params(6),
       params(7))),'omitnan');

opts = statset('MaxIter', 1000, 'MaxFunEvals', 2000, 'TolX',1e-9, '
   TolFun',1e-9,'Display','iter');
estimated_parameters = mle(xvec,'nloglf',custnloglf,'Start',[0.5,
   1.5, 1, -1.5, 1.5, 2, 2],'Options',opts)
```

```
p_fitted = estimated_parameters(1);
alpha1_fitted = estimated_parameters(2);
c1_fitted = estimated_parameters(3);
mu1_fitted = estimated_parameters(4);
alpha2_fitted = estimated_parameters(5);
c2_fitted = estimated_parameters(6);
mu2_fitted = estimated_parameters(7);

pdf_mix = @(x, p, params1, params2) ...
      p * pdf(makedist('Stable','alpha',params1(1),'beta',0,'gam',
         params1(2),'delta',params1(3)), x) + ...
    (1-p) * pdf(makedist('Stable','alpha',params2(1),'beta',0,'gam',
       params2(2),'delta',params2(3)), x);

xgrid = linspace(-10, 10, 4000);
true_pdf = pdf_mix(xgrid, p, [alpha1, c1, mu1], [alpha2, c2, mu2]);
fitted_pdf = pdf_mix(xgrid, p_fitted, [alpha1_fitted, c1_fitted,
   mu1_fitted], [alpha2_fitted, c2_fitted, mu2_fitted]);

figure;
hold on
axis square

xlim([-10, 10]);
plot(xgrid, fitted_pdf, 'LineWidth', 1.5,'LineStyle', '--');
hold on
plot(xgrid, true_pdf, 'LineWidth', 1.5);
legend('Fitted PDF', 'True PDF', 'Location', 'northeast');
title('True vs Fitted values');
text(0.02, 0.9, ...
    sprintf(['Actual values\n', '\\alpha_1 = %.2f, c_1 = %.3f, \\
       mu_1 = %.3f\n', ...
             '\\alpha_2 = %.2f, c_2 = %.3f, \\mu_2 = %.3f\n', ...
             'p = %.3f'], ...
             alpha1, c1, mu1, alpha2, c2, mu2, p), ...
    'FontSize', 9, 'BackgroundColor', 'w','Units', 'normalized', ...
    'EdgeColor', 'k');
text(0.02, 0.7, ...
    sprintf(['Fitted values\n', '\\alpha_1 = %.2f, c_1 = %.3f, \\
       mu_1 = %.3f\n', ...
             '\\alpha_2 = %.2f, c_2 = %.3f, \\mu_2 = %.3f\n', ...
             'p = %.3f'], ...
```

```
            alpha1_fitted, c1_fitted, mu1_fitted, alpha2_fitted,
               c2_fitted, mu2_fitted, p_fitted), ...
    'FontSize', 9, 'BackgroundColor', 'w','Units', 'normalized', ...
    'EdgeColor', 'k');
exportgraphics(gcf, 'PlotMixtureTask3.png', 'Resolution', 1200);
```

Listing 27: Code to load the DJIA stock returns, fit the IID mixture model, and save the estimated parameters.

```
load carsmall;
load carbig;

R = load('DJIA30stockreturns.mat');
ret = R.DJIARet;
n_assets = size(ret);
n_assets = n_assets(2);

custnloglf = @(params,data,cens,freq) ...
    - sum(log(params(1) * pdf('Stable',data,params(2),0,params(3),
       ...
    params(4)) ...
    + (1 - params(1)) * pdf('Stable',data,params(5),0,params(6),
       params(7))),'omitnan');
opts = statset('MaxIter', 1000, 'MaxFunEvals', 2000, 'TolX',1e-9, '
   TolFun',1e-9, 'Display','iter');
for j=1:n_assets
    pars = mle(ret(:,j),'nloglf',custnloglf,'Start',[0.7, 1.8, std(
       ret(:,j)), mean(ret(:,j)), 1.8, std(ret(:,j)), mean(ret(:,j))
       ],'options',opts);
    save(sprintf("estimated_parameters_%d.mat", j), 'pars');
end
```

Listing 28: Code to load the estimated parameters, plot the fitted PDFs and the actual values, and export the plots for all 25 stocks.

```
figure;
t = tiledlayout(2,2,'TileSpacing','compact','Padding','compact');
% the tiledlayout parameters were changed to 1,1 for the final plot
   of the 25th asset
for j = 1:4
% the loop parameters were changed in batches of 4, and finally in a
    batch of 1
nexttile;
params = load(sprintf("estimated_parameters_%d.mat", j)).pars;
```

```
pdf_mix = @(x, params1) ...
      params1(1) * pdf(makedist('Stable','alpha',params1(2),'beta'
         ,0,'gam',params1(3),'delta',params1(4)), x) + ...
    (1-params1(1)) * pdf(makedist('Stable','alpha',params1(5),'beta'
       ,0,'gam',params1(6),'delta',params1(7)), x);

xvec = linspace(quantile(ret(:,j), 0.001)...
, quantile(ret(:,j), 0.999), 10000);

y = pdf_mix(xvec, params);
histogram(ret(:,j), 150, 'Normalization', 'pdf', ...
          'EdgeColor', 'none', 'FaceAlpha', 0.3); hold on;
plot(xvec, y, 'r-', 'LineWidth', 1.6);
xlabel('Return');
ylabel('Density');
title(sprintf('DJIA stock %d: fitted stable mixture', j));
legend('Empirical density', 'Fitted mixture','FontSize', 16...
, 'Location', 'northeast');
grid on;

xlim([quantile(ret(:,j),0.001) quantile(ret(:,j),0.999)]);
yl = ylim; xl = xlim;
text(0.35, 0.97, ...
    sprintf(['\\alpha_1 = %.2f, c_1 = %.3f, \\mu_1 = %.3f\n', ...
             '\\alpha_2 = %.2f, c_2 = %.3f, \\mu_2 = %.3f\n', ...
             'p = %.3f'], ...
             params(2), params(3), params(4), params(5), params(6),
                params(7), params(1)), ...
    'FontSize', 16, 'BackgroundColor', 'w','Units', 'normalized', ...
    'EdgeColor', 'k', 'HorizontalAlignment', 'right'...
    ,'VerticalAlignment', 'top');

% save figure for the report
print(gcf, sprintf('PlotTask3_%d.png', 1), ...
      '-dpng', '-r300');

end
```