University of Zurich[UZH]

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

Home Assignment 1, due October 18th, 2022
Prof. Dr. Marc Paolella

Ilaria Avallone, 18-435-800
Lorena Tassone, 18-700-237
Naina Srivastava, 21-738-117

Question 1

The goal of this exercise is to plot the Stable Paretian Distribution using both the theoretical and simulation-based Probability Density Function (PDF).

For the simulation-based approach we calculate a kernel density estimate based on 1e6 (one million) simulated IID stable variates, using the parameters α = 1.7, β = -0.4, scale (c) = 2 and location (d) = 0.3. This was done by the *stabgen* function, provided in the book "Fundamental Statistical Inference" by Marc S. Paolella. After having simulated those IID stable variates, a probability estimate f is obtained through the MATLAB built-in function *ksdensity*.
The generated PDF is shown in Figure 1 as the red line.

The actual Asymmetric Stable density was computed using numerical integration. We make use of the *asymstab* function which is also provided in the above mentioned book. However, we had to modify the function, such that the scale and location parameters are taken into consideration for the calculation and named it *asymstabplus*.
The generated actual PDF is shown in Figure 1 as the blue line.

Listing 1: Plotting the PDF of the Stable Distribution: True Density vs. Kernel Density Estimate

```
a=1.7;          % alpha
b=-0.4;         % beta
c=2;            % scale
d=0.3;          % location
n=1e6;          % sample size
xvec=-20:0.01:20;

% creating the kernel density estimate by generating 1e6 simulated IID
% variates from the stable distribution and simulating them
eststab = stabgen(n,a,b,c,d,2);
[f,cd] = ksdensity(eststab, xvec);
figure, plot(cd, f, 'r-', 'linewidth', 2)

% creating the true density, based on analytic calculation through the
% asymstabplus function which returns the pdf
truestab1 = asymstabplus(xvec,a,b,c,d);
hold on, plot(xvec, truestab1, 'b--', 'linewidth', 2)

% adjusting the plot
xlim([-20 20])
legend('Simulated PDF','Theoretical PDF', 'Location', 'NorthWest')
title('PDFs of the Stable Distribution')
xlabel("x"); ylabel("S_{1.7, -0.4}(0.3, 2)(x)")
set(gca, 'fontsize', 12)
hold off

saveas(gcf, 'Assignment1_ex1.png')
```

Listing 2: Function that generates a random sample of size 'nobs' from the Stable Distribution

```
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;
rand('twister',seed), V=unifrnd(-pi/2,pi/2,1,z);
rand('twister',seed+42), 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(d)*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
```

Listing 3: Function that generates the PDF of the Asymmetric Stable Distribution based on Numeric Integration, adjusted to use a location and scale parameter

```
function [f,F] = asymstabplus(xvec,a,b,c,d)

if nargin<3, b=0; end
bordertol=1e-8; lo=bordertol ; hi=1-bordertol ; tol =1e-7;
xl=length(xvec); F=zeros(xl,1); f=F;
for loop=1:length(xvec)
    x=xvec(loop);
    f(loop)=(integral(@(uvec) fffplus(uvec,x,a,b,c,d,1), lo, hi)/pi);
    if nargout>1
        F(loop)=0.5-(1/pi)*integral(@(uvec) fffplus(uvec,x,a,b,c,d,0),
            lo, hi)/pi;
    end
end

function I = fffplus(uvec, x, a, b, c, d, dopdf)
subs = 1; I = zeros(size(uvec));
for ii=1:length(uvec)
    u=uvec(ii);
    if subs==1, t=(1-u)/u; else t=u/(1-u); end
    if a==1, cf = exp(-c*abs(t)*(1+1i*b*(2/pi)*sign(t)*log(t)) + 1i*d*t
        );
    else
        cf=exp(-((c^a)*(abs(t))^a)*(1-1i*b*sign(t)*tan(pi*a/2)) + 1i*d*
            t);
    end
    z=exp(-1i*t*x).*cf; if dopdf==1,g=real(z); else g=imag(z)./t; end
    if subs==1, I(ii)=g*u^(-2); else I(ii)=g*(1-u )^(-2); end
end
```

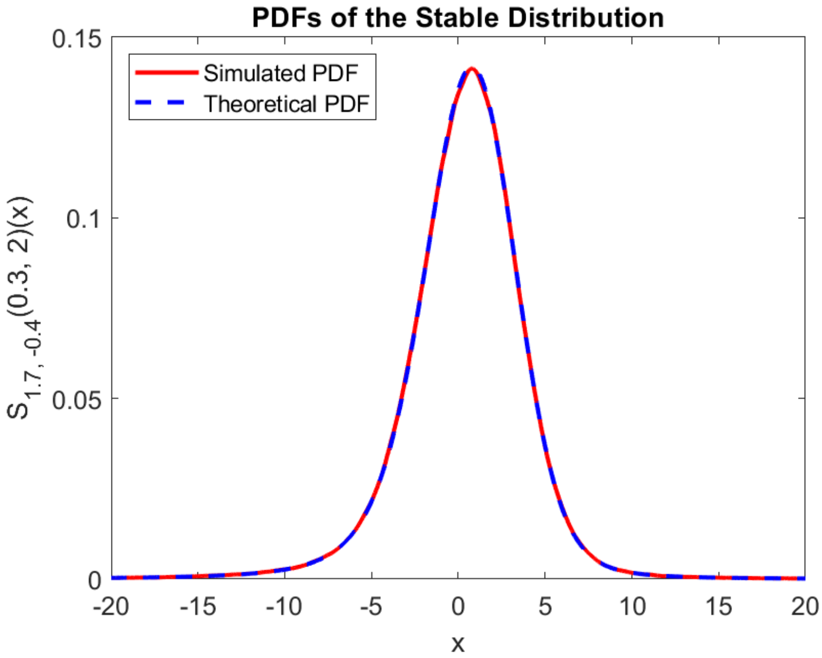

Figure 1: PDFs of the Stable Distribution

The two curves coincide pretty well, since for the simulation we used a sample size of 1e6. The higher the sample size, the better is the approximation. We chose the sample size of 1e6 since for this value the curve stabilizes.

Question 2

In the second exercise we want to confirm that stable distributions are closed under addition, meaning that the convolution of two independent stable random variables with the same α but possibly different β, location and scale, is itself again stable.

To do so, since the α's are the same, we can calculate the β, scale and location parameter of the convolution by using the following formulas from the lecture slides:

$$\mu = \sum_{t=1}^{n} \mu_i \tag{1}$$

$$c = (c_1^\alpha + \cdots + c_n^\alpha)^{1/\alpha} \tag{2}$$

$$\beta = \frac{\beta_1 c_1^\alpha + \cdots + \beta_n c_n^\alpha}{c_1^\alpha + \cdots + c_n^\alpha} \tag{3}$$

With α = 1.7, β_1 = -0.4, β_2 = -0.6, $scale_1$ = 2, $scale_2$ = 4, $location_1$ = -0.5 and $location_2$ = -0.3 we get the convoluted parameters β = -0.5529, scale = 4.6839 and location = -0.8000.

After having calculated β, scale and location of the convolution, we calculate the theoretical distribution of the convolution using the *asymstabplus* function, as we did in Question 1. In Figure 2 the theoretical distribution is represented by the blue line.
Further, we calculated the kernel density estimate of the simulated values of the convolution. More precisely, as already specified in Question 1, we make use of the *stabgen* function to generate 1e6 simulated IID variates from the stable distribution, and the *ksdensity* function allows us to obtain the probability density estimate. In Figure 2 the simulated distribution is represented by the red line.

As we can see in Figure 2, the two distributions coincide pretty well and we can see that the convolution of two independent stable random variables with the same alpha but different beta, location and scale, is itself again stable.

Listing 4: Convolution of two independent Stable Random Variables

```
a=1.7;        % alpha
b1=-0.4;      % beta 1
c1=2;         % scale 1
d1=-0.5;      % location 1
b2=-0.6;      % beta 2
c2=4;         % scale 2
d2=-0.3;      % location 2
n = 1e6;      % sample size
x=-30:0.01:30;

% since alpha is the same for both r.v., it is easy to calculate the
    beta, scale and location parameter of the convolution
d=d1+d2;
```

```
c=((c1^a)+(c2^a))^(1/a);
b=((b1*c1^a)+(b2*c2^a))/((c1^a)+(c2^a));

% theoretical pdf of the convolution
truestab_conv = asymstabplus(x,a,b,c,d);
figure, plot(x, truestab_conv, 'b-', 'linewidth', 2)

% kernel density estimate of simulated values of stable distribution
eststab = stabgen(n,a,b,c,d,2);
[f,cd] = ksdensity(eststab, x);
hold on, plot(cd, f, 'r--', 'linewidth', 2)

% adjusting the plot
title('Convolution of Stable Random Variables')
legend('True density','Kernel density')
xlim([-30 30])
xlabel("x"); ylabel("S_{a, b}(d, c)(x)");
set(gca, 'fontsize', 12)
hold off

saveas(gcf, 'Assignment1_ex2.png')
```

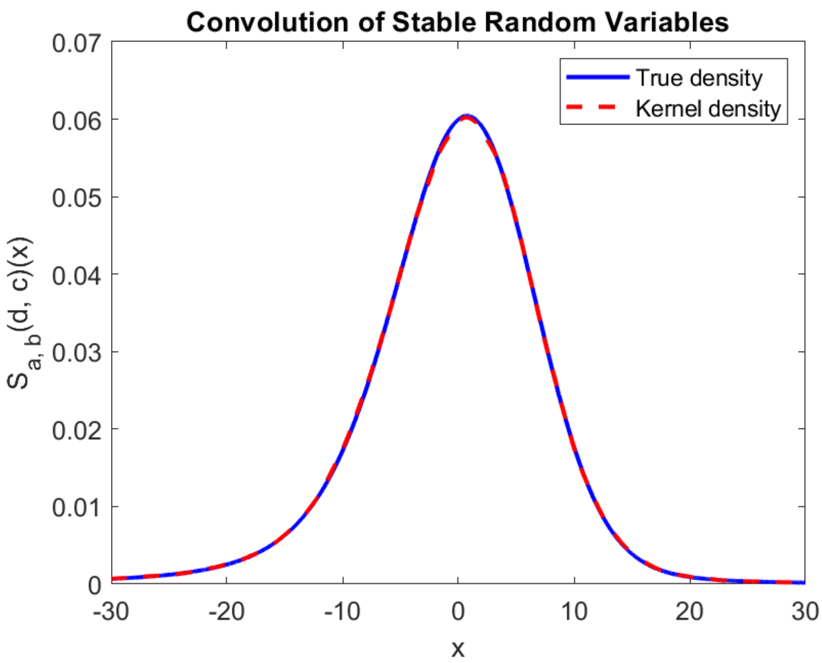

Figure 2: PDFs of the Convolution of two independent Stable Random Variables

Question 3

Question 3 shows four different ways of computing the convolution of two independent stable random variables with different values of tail index alpha.

For all methods, computations were executed twice. Once for alpha values α_1=1.6, α_2=1.8, and second for α_1=1.5, α_2=1.9.

For the first method, we use the simple integration formula (convolution formula) and program the integral convolution through the function *asymstabpdf_conv* shown in Listing 7. It does integration by parts on the product of the two characteristic functions.

Next, with the function *asymstab_invform* we compute the PDF by using the inversion formula applied to the characteristic function of the convolution of the two stable variables. More precisely, we first compute the characteristic functions of both variables using formula (4) for the case $\alpha \neq 1$.

$$\ln \varphi_X(t) = \begin{cases} -c^\alpha |t|^\alpha \left[1 - i\beta \operatorname{sgn}(t) \tan \frac{\pi\alpha}{2}\right] + i\mu t, & \text{if } \alpha \neq 1 \\ -c|t| \left[1 + i\beta \frac{2}{\pi} \operatorname{sgn}(t) \ln |t|\right] + i\mu t, & \text{if } \alpha = 1 \end{cases} \tag{4}$$

By multiplying the two characteristic functions we obtain the characteristic function of the sum of the two variables, the convolution. By then applying the inversion formula on the resulting characteristic function we obtain the desired PDF. This is shown in Listing 8.

The third method involves simulation and kernel density for computing the convolution. We again take advantage of the *stabgen* function to simulate random samples, as we already explained in the previous questions, together with the *ksdensity* function in order to obtain the simulated PDF.

Finally, we exploit the simple MATLAB built-in *conv* function to compute the convolution for the fourth time. It consists of a simple explicit form of numeric integration, and is a more efficient method in terms of time for the computation. The *conv* function can be also useful for computing the convolution of three independent random variables, as it is shown in Listing 6.

By plotting the density convolutions deriving from all the above-mentioned methods, we observe in Figure 3 (computed with alpha parameters of respectively 1.6 and 1.8) and Figure 4 (computed with alpha parameters of respectively 1.5 and 1.9) that the PDFs perfectly overlap, meaning that all methods are of course equivalent in terms of results.

Figure 5 represents the PDF of the convolution of three random variable, which, as mentioned before, was done by using the *conv* formula. Also in this case, the PDF coincides with the simulated PDF, which was generated by calculating the parameters of the convolution using formulas (1), (2) and (3) as done before in question two with two variables.

Listing 5: Convolution of two independent Stable Random Variables with different tail index alpha

```
a1=1.6; %a1=1.5      %alpha 1
b1=0;                %beta 1
c1=1;                %scale 1
d1=0;                %location 1
a2=1.8; %a2=1.9      %alpha 2
b2=0;                %beta 2
c2=1;                %scale 2
d2=0;                %location 2

xvec1 = -15:.05:15;
xvec2 = -15:.05:15;

%%% Density convolution 1: using integration/convolution formula
conv1 = asymstabpdf_conv(xvec1, a1, b1, a2, b2, 1);

%%% Density convolution 2: using invsersion formula
conv2 = asymstab_invform(xvec1, a1, b1, a2, b2);

%%% Density convolution 3: using simulation and kernel density
n = 1e6;

% simulating random sample with different seeds to avoid correlation
random_conv_sim1 = stabgen(n,a1, b1, c1, d1, 5);
random_conv_sim2 = stabgen(n,a2, b2, c2, d2, 265);
random_conv_3 = random_conv_sim1 + random_conv_sim2;
[f, c] = ksdensity(random_conv_3,xvec2);

%%% Density convolution 4: using the conv() function
% Interval size for density convolution calculation through the conv()
    function
dx = 0.05;
xvec= -15:dx:15;

% pdfs for the different alphas
truestab_a1 = asymstabplus(xvec,a1,b1,c1,d1);
truestab_a2 = asymstabplus(xvec,a2,b2,c2,d2);

% Convolution calculation
truestab_conv = conv(truestab_a1, truestab_a2, 'same')*dx;

% plot
figure
plot(xvec1, conv1, 'b-', 'linewidth', 2)
hold on
plot(xvec1,conv2, 'r-.', 'linewidth', 2)
plot (c, f, 'g--', 'linewidth', 2)
plot (xvec',truestab_conv, 'm:', 'linewidth',2)
title('Comparison Stable R.V. Convolution Methods')
legend('Numeric Integration','Inversion Formula', 'Simulation', 'Matlab
```

```
    Conv Formula')
xlim([-15 15])
xlabel("x"); ylabel("Sconv_{a, b}(d, c)(x)");
hold off

saveas(gcf, 'Assignment1_ex3.png')
```

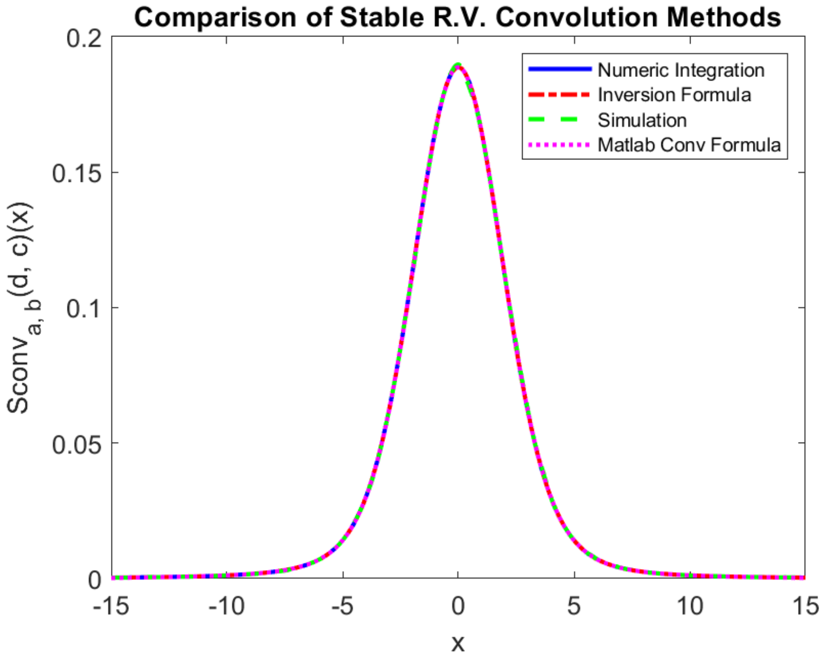

Figure 3: Comparison of Stable Random Variable Convolution PDFs generated using different methods (a1=1.6, a2=1.8)

Listing 6: Convolution of three independent Stable Random Variables with identical tail index alpha (Bonus Question)

```
a = 1.7;
b1 = 0; b2 = 0; b3 = 0;
c1 = 1; c2 = 1; c3 = 1;
d1 = 0; d2 = 0; d3 = 0;
dx = 0.01;
xvec= -15:dx:15;
n = 1e6;

% theoretical stable distribution
x = asymstabplus(xvec,a,b1,c1,d1);
y = asymstabplus(xvec,a,b2,c2,d2);
z = asymstabplus(xvec,a,b3,c3,d3);

conv_bonus_1 = conv(x,y,'same')*dx
truestab_conv_bonus = conv(conv_bonus_1, z, 'same')*dx
figure, plot(xvec',truestab_conv_bonus, 'b-', 'linewidth',2)

% kernel density estimate of simulated values of S
```

```
b_conv = (b1 * c1^a + b2 * c2^a + b3 * c3^a)/(c1^a + c2^a + c3^a);
c_conv = (c1^a + c2^a + c3^a)^(1/a);
d_conv = d1 + d2 + d3;

eststab_conv3 = stabgen(n, a, b_conv, c_conv, d_conv, 8);
[f,cd] = ksdensity(eststab_conv3, xvec);
hold on, plot(cd, f, 'r--', 'linewidth', 2)

% adjusting plot
title('Convolution of three Stable R.V.')
legend('Simulated PDF','Theoretical PDF', 'Location', 'NorthWest')
xlim([-15 15])
xlabel("x"); ylabel("Sconv3_{a, b}(d, c)(x)");
set(gca, 'fontsize', 12)
hold off

saveas(gcf, 'Assignment1_ex3_bonus.png')
```

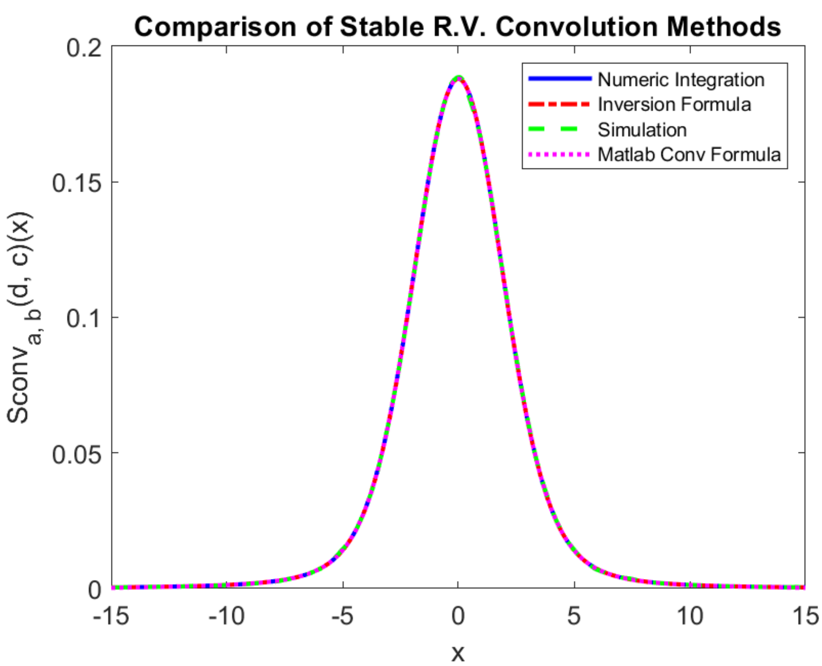

Figure 4: Comparison of Stable Random Variable Convolution PDFs generated using different methods (a1=1.5, a2=1.9)

Listing 7: Function generating the convolution of two Asymmetric Stable Random Variables with different alphas using Numeric Integration / Convolution Formula

```
function f = asymstabpdf_conv(xvec1, a1, b1, a2, b2, plotintegrand)

if nargin<4 , plotintegrand =0; end
xl=length(xvec1); f=zeros(xl, 1); n = length(xvec1);

for loop=1:n
    x=xvec1(loop);
```

```
    f(loop)=integral(@(xvec2) integrand_conv(xvec2,x,a1,b1,a2,b2), -Inf
        , Inf);
end

function f = integrand_conv(xvec, x, a1, b1, a2, b2)
    i1 = asymstab(xvec, a1, b1);
    i2 = asymstab(x-xvec, a2, b2);
    f = (i1.*i2)';
end
```

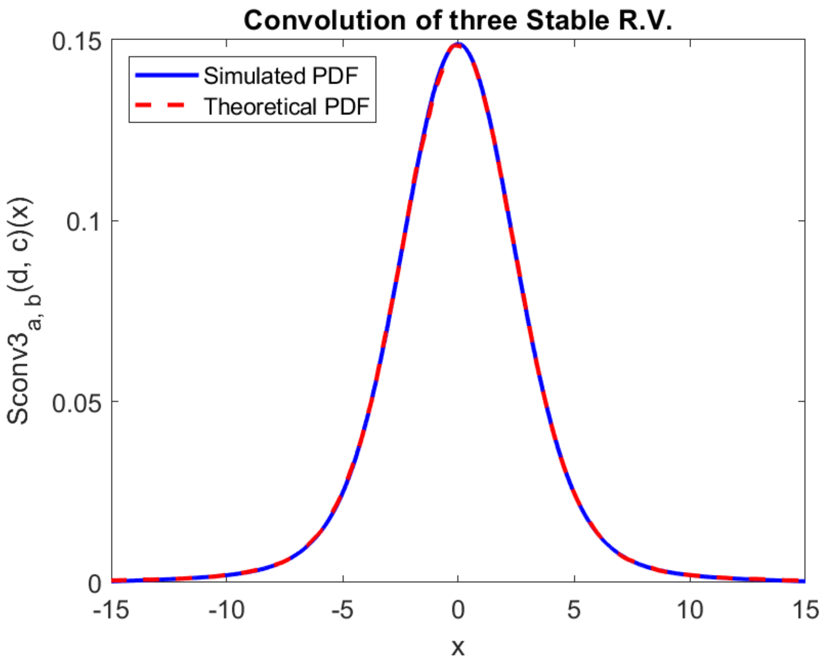

Figure 5: Comparison of Stable Random Variable convolution PDFs generated using different methods including a three R.V. convolution

Listing 8: Function generating the convolution of two Asymmetric Stable Random Variables with different alphas using the Inversion Formula

```
function f = asymstab_invform(xvec, a1, b1, a2, b2)

bordertol=1e-8; lo=bordertol; hi=1-bordertol; tol=1e-7;
xl=length(xvec); F=zeros(xl,1); f=F;

for loop=1:length(xvec)
    x=xvec(loop);
    f(loop)=integral(@(uvec) fff_inv(uvec,x,a1,b1,a2,b2,1), lo, hi) /
        pi;
end

function l = fff_inv(uvec,x,a1,b1,a2,b2,dopdf)
subs=1; l = zeros(size(uvec));
for ii=1:length(uvec)
    u=uvec(ii);
    if subs==1, t =(1-u)/u ; else t=u/(1-u) ; end
    cf1 = exp(-((abs(t))^a1)*(1-1i*b1*sign(t)*tan(pi*a1/2)));
    cf2 = exp(-((abs(t))^a2)*(1-1i*b2*sign(t)*tan(pi*a2/2)));
    cf = cf1.*cf2;
    z=exp(-1i*t*x).*cf;
    if dopdf==1, g=real(z); else g=imag(z)./t;
    end
    l(ii)=g*u^(-2);
end
```

Question 4

Question 4 aims at computing the Expected Shortfall (ES), both theoretically and via simulation.

We make use of the Stoyanov et al.(2006) result to first compute the theoretical ES for a particular set of stable parameters by using the *asymstableES* function in Listing 10. Using a tail probability ξ of 0.01, we obtain a value of -11.4713 for the theoretical Expected Shortfall.

We then compute it via simulation. More precisely, we simulate 1e6 stable variates via the *stabgen* function, and we then compute the empirical ES, by first calculating the Value-at-Risk (VaR) using the *quantile* function. We obtain a simulated ES value of -11.2002.

We observe that the two methods return quite similar values. The slight difference comes from the simulation error.

Listing 9: Theoretical and empirical Expected Shortfall

```
a = 1.7;
b = 0;
c = 0;
d = 1;
xi = 0.01;
method = 1;

% Theoretical ES based on a tail probability xi=0.01 using the Stoyanov
    et al. result.
[Theo_ES, Theo_VaR] = asymstableES(xi, a, b, c, d, method);
X = ['ES via Stoyanov et al: ', num2str(Theo_ES)];
disp(X);

% Simulation of ES
nobs = 10^6;
P = stabgen(nobs, a, b, d, c, 0);
VaR = quantile(P, xi);
Plo = P(P <= VaR);
ES_simulated = mean(Plo);
X = ['Simulated ES: ', num2str(ES_simulated)];
disp(X);

% Results
>> ES via Stoyanov et al: -11.4713
>> Simulated ES: -11.2002
```

Listing 10: Function calculating the Expected Shortfall of a Asymmetric Stable Distribution

```
function [ES, VaR] = asymstableES (xi, a, b, mu, scale, method)
if nargin < 3 , b=0; end, if nargin < 4 , mu=0; end
if nargin < 5 , scale=1; end, if nargin < 6 , method = 1; end

% Get q , the quantile from the S(0 ,1) distribution
opt=optimset('Display', 'off', 'TolX', 1e-6);
q=fzero(@stabcdfroot, -6, opt, xi, a, b); VaR=mu+scale*q;
if (q == 0)
     t0 = (1/a)*atan(b*tan(pi*a/2));
     ES = ((2*gamma((a-1)/a))/(pi-2*t0))*(cos(t0)/cos(a*t0)^(1/a));
return;
end
if (method==1), ES=(scale*Stoy(q, a, b)/xi)+mu;
else ES=(scale*stabletailcomp(q, a, b)/xi)+mu;
end

function diff = stabcdfroot(x, xi, a, b)
if exist('stableqkcdf.m', 'file'), F = stableqkcdf (x, [a, b], 1); % Nolan routine
else [~, F] = asymstab(x, a, b); %get the cdf of the asymmetric stable
end
diff = F - xi;

function tailcomp = stabletailcomp(q, a, b)
% direct integration of x*f(x) and use of asymptotic tail behavior
K = (a/pi)*sin(pi*a/2)*gamma(a)*(1-b); % formula for K_
ell = -120; M = ell; display = 0; term1 = K*(-M)^(1-a)/(1-a);
%term3 = quadl(@stableCVARint, ell, q, 1e-5, display, a, b);
term3 = integral(@(x) stableCVARint(x, a, b), ell, q);
tailcomp = term1 + term3;

function [g] = stableCVARint(x, a, b)
if exist('stableqkpdf.m', 'file'), den = stableqkpdf(x,[a, b], 1);
else den = asymstab(x, a, b)';
end
g = x.*den;

function S = Stoy(cut, alpha, beta)
if nargin<3, beta=0; end
cut= -cut; %weuseadifferentsignconvention
bbar = -sign(cut) * beta;
t0bar = (1 / alpha) * atan(bbar * tan(pi * alpha / 2));
% 'beta==0' => 'bbar==0' => 't0bar==0'
small = 1e-8; tol = 1e-8; abscut = abs(cut); display = 0;
integ = quadl(@stoyint , -t0bar+small , pi/2-small , tol , ...
     display , abscut , alpha , t0bar );
S = alpha / (alpha-1) / pi * abscut * integ;
```

```
function I = stoyint(t, cut, a, t0bar)
s=t0bar + t;
g = sin(a*s - 2*t) ./ sin(a*s) - a * cos(t).^2 ./ sin(a*s).^2;
v = (cos(a*t0bar)).^(1 / (a - 1)) .* (cos(t) ./ sin(a*s)).^(a / (a-1))
    ...
.* cos(a*s-t) ./ cos(t); term = -(abs(cut)^(a / (a-1))); I=g.*exp(term
    .*v);
```

To find an optimal sample size, such that the absolute difference between the theoretical ES and the simulated one is lower than a certain value (we chose 0.001), we calculated this difference for different sample sizes and increased nobs as long as the difference was above this threshold. This resulted in a sample size of 29’000’000 which yields a difference between the two methods of 0.00060281. One could lower the increment in the loop to be more precise. Additionally, this might depend on the choice of the parameters.

Listing 11: Optimal sample size to minimize difference between theoretical and simualted ES

```
Theo_ES_bonus = asymstableES(xi, a,b,c,d, method);
ES_simulated_bonus = ES_simulated;
nobs = 10e6;

while abs(Theo_ES_bonus - ES_simulated_bonus) > 0.001
    nobs=nobs + 1000000;
    P_bonus = stabgen(nobs,a,b,d,c,0);
    Var_bonus = quantile(P_bonus,xi);
    Plo_bonus = P_bonus(P_bonus<=Var_bonus);
    ES_simulated_bonus=mean(Plo_bonus);
end

disp(['Difference: ',num2str(abs(Theo_ES_bonus - ES_simulated_bonus))])
disp(['Sample Size: ',num2str(nobs)])

% Results
>> Difference: 0.00060281
>> Sample Size: 29000000
```

Question 5

In Question 5 we compute the Expected Shortfall for a convolution of two independent stables with different alpha parameters. For the simulated ES, we have used the mathematical formula:

$$ES_\alpha = -1/\alpha \int_0^\alpha VaR_\gamma(X)\, d\gamma \tag{5}$$

This shows that the Expected Shortfall is the negative of the average of all the returns worse than the Value-at-Risk.

For the simualted ES, we simulated two independent stable random variables by using the *stabgen* function and built their sum. Note, we used two different seed values to be able to generate two independent random variables. As in Question 4, we go through the VaR to calculate the empirical ES.
Also for the theoretical ES, we simulated the sum of two stable distributions as before, using the same seeds in the *stabgen* function as in the simulated method, to be able to compare the results.
We then estimated the parameters using Stable Regression which is done with the funtion *stablereg*. We get the following parameters: $\alpha = 1.6921$, $\beta = 0.0024$, $\sigma = 1.5066$ and $\mu = 0.0006$. Next, we applied the Stoyanov et al. (2006) method to the estimated parameters to calculate the ES.

The two above mentioned methods were executed for different levels of confidence ξ. The results for $\alpha_1 = 1.6$ and $\alpha_2 = 1.8$ are the following:

ξ	0.01	0.025	0.05
Simulated ES	-18.6088	-11.196	-7.8586
ES via Stoyanov et al.	-17.6999	-10.8336	-7.6822

For $\alpha_1 = 1.5$ and $\alpha_2 = 1.9$ we get the following parameters: $\alpha = 1.6565$, $\beta = 0.0032$, $\sigma = 1.5079$ and $\mu = 0.0009$. The results are the following:

ξ	0.01	0.025	0.05
Simulated ES	-23.2487	-13.2523	-8.9363
ES via Stoyanov et al.	-19.8521	-11.9039	-8.287

We can see that the higher ξ gets, the absolute ES becomes smaller. Further, increasing ξ results in more accurate values for the ES, since the results of the two methods become more similar. This can be seen for both α choices.

The difference between the simulated method and the Stoyanov et al. (2006) method might come from the fact, that the Stoyanov et al. (2006) method does not involve the stable density to compute the ES as opposed to the prior method. Further, working with simulated values, as for the simulated ES, always involves simulation error.

Listing 12: Theoretical and empirical Expected Shortfall of a convolution of two Stable Random Variables

```
a1=1.6;  %alpha 1     ->1.5
b1=0;    %beta 1
c1=1;    %scale 1
d1=0;    %location 1
a2=1.8;  %alpha 2     -> 1.9
b2=0;    %beta 2
c2=1;    %scale 2
d2=0;    %location 2

xi = [0.01 0.025 0.05];
xvec1 = -15:.05:15;

% simulated ES
n = 1e6;
X1 = stabgen(n, a1, b1, c1, d1, 1); X2 = stabgen(n, a2, b2, c2, d2, 2);
Sum = X1 + X2;
for xi=[0.01 0.025 0.05]
    VaR = quantile(Sum, xi);
    Plo = Sum(Sum <= VaR);
    ES_simulated = mean(Plo);
    X = ['Simulated ES for xi=', num2str(xi), ': ', num2str(
        ES_simulated)];
    disp(X);
end

% Theoretical Stoyanov ES with estimated parameters
n = 1e6; method = 1;
X1 = stabgen(n, a1, b1, c1, d1, 1); X2 = stabgen(n, a2, b2, c2, d2, 2);
Sum = X1 + X2;
[alpha,beta,sigma,mu] = stablereg(Sum);
s1=['Alpha: ',num2str(alpha),' Beta: ',num2str(beta),' Sigma: ',num2str
    (sigma),' mu: ',num2str(mu)];
disp(s1)
for xi=[0.01 0.025 0.05]
    [Theo_ES, Theo_VaR] = asymstableES(xi, alpha, beta, mu, sigma,
        method);
    X2 = ['ES via Stoyanov et al. for xi=',num2str(xi),': ', num2str(
        Theo_ES)];
    disp(X2);
end


% Results for a1=1.6, a2=1.8
>> Simulated ES for xi=0.01: -18.6088
>> Simulated ES for xi=0.025: -11.196
>> Simulated ES for xi=0.05: -7.8586

>> Alpha: 1.6921 Beta: 0.00238 Sigma: 1.5066 mu: 0.00062009

>> ES via Stoyanov et al. for xi=0.01: -17.6999
>> ES via Stoyanov et al. for xi=0.025: -10.8336
>> ES via Stoyanov et al. for xi=0.05: -7.6822
```

```
% Results for a1=1.5, a2=1.9
>> Simulated ES for xi=0.01: -23.2487
>> Simulated ES for xi=0.025: -13.2523
>> Simulated ES for xi=0.05: -8.9363

>> Alpha: 1.6565 Beta: 0.0032434 Sigma: 1.5079 mu: 0.00088684

>> ES via Stoyanov et al. for xi=0.01: -19.8521
>> ES via Stoyanov et al. for xi=0.025: -11.9039
>> ES via Stoyanov et al. for xi=0.05: -8.287
```

Listing 13: Function generating regression parameter estimates of a Stable Distribution

```
function [alpha,beta,sigma,mu]=stablereg(x,epsilon,maxit,deltac)
%STABLEREG Regression parameter estimates of a stable distribution.

% Initialize input parameters with default values
if nargin<4, delc=0:99; else delc=deltac; end;
if nargin<3, maxit=5; end;
if nargin<2, epsilon=0.00001; end;

% Define optimal parameter vectors (see [1])
Kopt   = [9   11  16  18  22  24  68  124];
Lopt   = [10  14  16  18  14  16  38  68];
indexA = [1.9 1.5 1.3 1.1 0.9 0.7 0.5 0.3];

[xrow,xcol]=size(x);
if xrow==1,
    x = x';
    xrow = xcol;
    xcol = 1;
end;

% Compute initial parameter estimates using McCulloch's method
[alpha,beta,sigma,mu] = stablecull(x);

% Run regression
x = (x-ones(xrow,1)*mu)./(ones(xrow,1)*sigma);
for n = 1:xcol,
    X = x(:,n);
    K = 11;
    t = [1:K]*pi/25;
    w = log(abs(t));
    w1 = w-mean(w);
    y = [];
    for tt = t,
        y = [y mean(exp(i*tt*X))];
    end;
    y = log(-2*log(abs(y)));
    alpha1 = (sum(w1.*(y-mean(y))))/sum(w1.*w1);
    if alpha1<=0.9,
        K = 30;
        t = [1:K]*pi/25;
```

```
        w = log(abs(t));
        w1 = w-mean(w);
        y = [];
        for tt = t,
            y = [y mean(exp(i*tt*X))];
        end;
        y = log(-2*log(abs(y)));
        alpha1 = (sum(w1.*(y-mean(y))))/sum(w1.*w1);
    end;
    it = 1;
    c1 = 0;
    beta2 = beta(n);
    delta2 = 0;
    while (it<=maxit) & ((abs(c1-1))>epsilon),
        K = Kopt(min([(find(indexA<=alpha1)) 8]));
        t = [1:K]*pi/25;
        w = log(abs(t));
        w1 = w-mean(w);
        y = [];
        for tt = t,
            y = [y mean(exp(i*tt*X))];
        end;
        y = log(-2*log(abs(y)));
        alpha1 = (sum(w1.*(y-mean(y))))/sum(w1.*w1);
        c1 = (0.5*exp(mean(y-alpha1*w)))^(1/alpha1);
        L = Lopt(min([(find(indexA<=alpha1)) 8]));
        u = [1:L]*pi/50;
        X = X/c1;
        sinXu = [];
        cosXu = [];
        for uu = u,
            uuX = uu*X;
            sinXu = [sinXu sum(sin(uuX))];
            cosXu = [cosXu sum(cos(uuX))];
        end;

        testcos = ((1+0*(delc'))*cosXu).*cos(delc'*u)+((1+0*(delc'))*sinXu).*sin(delc'*u);
        testcos = sum(abs(diff(sign(testcos'))));
        deltac = delc(min(find(testcos==0)));
        if length(deltac)==0,
            warning('Unable to find DELTAc');
            it = maxit + 1;
        else
            X = X-deltac;
            z = atan((sinXu*cos(deltac)-cosXu*sin(deltac))./(cosXu*cos(deltac)+sinXu*sin(deltac)));
            y = (c1^alpha1)*tan(pi*alpha1/2)*sign(u).*(u.^alpha1);
            delta2 = (sum(y.*y)*sum(u.*z)-sum(u.*y)*sum(z.*y))/(sum(u.*u)*sum(y.*y)-(sum(u.*y))^2);
            beta2 = (sum(u.*u)*sum(y.*z)-sum(u.*y)*sum(u.*z))/(sum(u.*u)*sum(y.*y)-(sum(u.*y))^2);
            sigma(n) = sigma(n)*c1;
            mu(n) = mu(n)+deltac*sigma(n);
            it = it+1;
```

```
            end;
        end;
        alpha(n) = alpha1;
        beta(n) = beta2;
        mu(n) = mu(n)+sigma(n)*delta2;
end;

% Correct estimates for out of range values
alpha(alpha<=0) = 10^(-10)+0*alpha(alpha<=0);
alpha(alpha>2) = 2+0*alpha(alpha>2);
sigma(sigma<=0) = 10^(-10)+0*sigma(sigma<=0);
beta(beta<-1) = -1+0*beta(beta<-1);
beta(beta>1) = 1+0*beta(beta>1);

function [alpha,beta,sigma,mu]=stablecull(x)
% STABLECULL Quantile parameter estimates of a stable distribution.

% Compute quantiles
x = sort(x);
x05 = prctile(x,5);
x25 = prctile(x,25);
x50 = prctile(x,50);
x75 = prctile(x,75);
x95 = prctile(x,95);

% Compute quantile statistics
va = (x95-x05)./(x75-x25);
vb = (x95+x05-2*x50)./(x95-x05);
vs = x75-x25;

% Define interpolation matrices (see [1])
tva = [2.439 2.5 2.6 2.7 2.8 3.0 3.2 3.5 4.0 5.0 6.0 8.0 10.0 15.0
    25.0];
tvb = [0.0, 0.1, 0.2, 0.3, 0.5, 0.7, 1.0];
ta = [2.0 1.9 1.8 1.7 1.6 1.5 1.4 1.3 1.2 1.1 1.0 0.9 0.8 0.7 0.6 0.5];
tb = [0.0, 0.25, 0.5, 0.75, 1.0];

psi1 = [2.000, 2.000, 2.000, 2.000, 2.000, 2.000, 2.000;
        1.916, 1.924, 1.924, 1.924, 1.924, 1.924, 1.924;
        1.808, 1.813, 1.829, 1.829, 1.829, 1.829, 1.829;
        1.729, 1.730, 1.737, 1.745, 1.745, 1.745, 1.745;
        1.664, 1.663, 1.663, 1.668, 1.676, 1.676, 1.676;
        1.563, 1.560, 1.553, 1.548, 1.547, 1.547, 1.547;
        1.484, 1.480, 1.471, 1.460, 1.448, 1.438, 1.438;
        1.391, 1.386, 1.378, 1.364, 1.337, 1.318, 1.318;
        1.279, 1.273, 1.266, 1.250, 1.210, 1.184, 1.150;
        1.128, 1.121, 1.114, 1.101, 1.067, 1.027, 0.973;
        1.029, 1.021, 1.014, 1.004, 0.974, 0.935, 0.874;
        0.896, 0.892, 0.887, 0.883, 0.855, 0.823, 0.769;
        0.818, 0.812, 0.806, 0.801, 0.780, 0.756, 0.691;
        0.698, 0.695, 0.692, 0.689, 0.676, 0.656, 0.595;
        0.593, 0.590, 0.588, 0.586, 0.579, 0.563, 0.513];
psi2 = [0.000, 2.160, 1.000, 1.000, 1.000, 1.000, 1.000;
        0.000, 1.592, 3.390, 1.000, 1.000, 1.000, 1.000;
        0.000, 0.759, 1.800, 1.000, 1.000, 1.000, 1.000;
```

```
        0.000, 0.482, 1.048, 1.694, 1.000, 1.000, 1.000;
        0.000, 0.360, 0.760, 1.232, 2.229, 1.000, 1.000;
        0.000, 0.253, 0.518, 0.823, 1.575, 1.000, 1.000;
        0.000, 0.203, 0.410, 0.632, 1.244, 1.906, 1.000;
        0.000, 0.165, 0.332, 0.499, 0.943, 1.560, 1.000;
        0.000, 0.136, 0.271, 0.404, 0.689, 1.230, 2.195;
        0.000, 0.109, 0.216, 0.323, 0.539, 0.827, 1.917;
        0.000, 0.096, 0.190, 0.284, 0.472, 0.693, 1.759;
        0.000, 0.082, 0.163, 0.243, 0.412, 0.601, 1.596;
        0.000, 0.074, 0.147, 0.220, 0.377, 0.546, 1.482;
        0.000, 0.064, 0.128, 0.191, 0.330, 0.478, 1.362;
        0.000, 0.056, 0.112, 0.167, 0.285, 0.428, 1.274];

psi3 = [1.908, 1.908, 1.908, 1.908, 1.908;
        1.914, 1.915, 1.916, 1.918, 1.921;
        1.921, 1.922, 1.927, 1.936, 1.947;
        1.927, 1.930, 1.943, 1.961, 1.987;
        1.933, 1.940, 1.962, 1.997, 2.043;
        1.939, 1.952, 1.988, 2.045, 2.116;
        1.946, 1.967, 2.022, 2.106, 2.211;
        1.955, 1.984, 2.067, 2.188, 2.333;
        1.965, 2.007, 2.125, 2.294, 2.491;
        1.980, 2.040, 2.205, 2.435, 2.696;
        2.000, 2.085, 2.311, 2.624, 2.973;
        2.040, 2.149, 2.461, 2.886, 3.356;
        2.098, 2.244, 2.676, 3.265, 3.912;
        2.189, 2.392, 3.004, 3.844, 4.775;
        2.337, 2.635, 3.542, 4.808, 6.247;
        2.588, 3.073, 4.534, 6.636, 9.144];


psi4 = [0.0,    0.0,    0.0,    0.0,    0.0;
        0.0, -0.017, -0.032, -0.049, -0.064;
        0.0, -0.030, -0.061, -0.092, -0.123;
        0.0, -0.043, -0.088, -0.132, -0.179;
        0.0, -0.056, -0.111, -0.170, -0.232;
        0.0, -0.066, -0.134, -0.206, -0.283;
        0.0, -0.075, -0.154, -0.241, -0.335;
        0.0, -0.084, -0.173, -0.276, -0.390;
        0.0, -0.090, -0.192, -0.310, -0.447;
        0.0, -0.095, -0.208, -0.346, -0.508;
        0.0, -0.098, -0.223, -0.383, -0.576;
        0.0, -0.099, -0.237, -0.424, -0.652;
        0.0, -0.096, -0.250, -0.469, -0.742;
        0.0, -0.089, -0.262, -0.520, -0.853;
        0.0, -0.078, -0.272, -0.581, -0.997;
        0.0, -0.061, -0.279, -0.659, -1.198];
% Compute estimates by interpolationg through the tables
[xrow,xcol] = size(x);
if (xrow == 1), xcol = 1; end;
for n = 1:xcol,
    tvai1 = max([1 find(tva <= va(n))]);
    tvai2 = min([15 find(tva >= va(n))]);
    tvbi1 = max([1 find(tvb <= abs(vb(n)))]);
    tvbi2 = min([7 find(tvb >= abs(vb(n)))]);
```

```
    dista = (tva(tvai2)-tva(tvai1));
    if dista ~= 0,
        dista = (va(n)-tva(tvai1))/dista;
    end;
    distb = (tvb(tvbi2)-tvb(tvbi1));
    if distb ~= 0,
        distb = (abs(vb(n))-tvb(tvbi1))/distb;
    end;
    psi1b1 = dista*psi1(tvai2,tvbi1)+(1-dista)*psi1(tvai1,tvbi1);
    psi1b2 = dista*psi1(tvai2,tvbi2)+(1-dista)*psi1(tvai1,tvbi2);
    alpha(n) = distb*psi1b2+(1-distb)*psi1b1;
    psi2b1 = dista*psi2(tvai2,tvbi1)+(1-dista)*psi2(tvai1,tvbi1);
    psi2b2 = dista*psi2(tvai2,tvbi2)+(1-dista)*psi2(tvai1,tvbi2);
    beta(n) = sign(vb(n))*(distb*psi2b2+(1-distb)*psi2b1);
    tai1 = max([1 find(ta >= alpha(n))]);
    tai2 = min([16 find(ta <= alpha(n))]);
    tbi1 = max([1 find(tb <= abs(beta(n)))]);
    tbi2 = min([5 find(tb >= abs(beta(n)))]);
    dista = (ta(tai2)-ta(tai1));
    if dista ~= 0,
        dista = (alpha(n)-ta(tai1))/dista;
    end;
    distb = (tb(tbi2)-tb(tbi1));
    if distb ~= 0,
        distb = (abs(beta(n))-tb(tbi1))/distb;
    end;
    psi3b1 = dista*psi3(tai2,tbi1)+(1-dista)*psi3(tai1,tbi1);
    psi3b2 = dista*psi3(tai2,tbi2)+(1-dista)*psi3(tai1,tbi2);
    sigma(n) = vs(n)/(distb*psi3b2+(1-distb)*psi3b1);
    psi4b1 = dista*psi4(tai2,tbi1)+(1-dista)*psi4(tai1,tbi1);
    psi4b2 = dista*psi4(tai2,tbi2)+(1-dista)*psi4(tai1,tbi2);
    zeta = sign(beta(n))*sigma(n)*(distb*psi4b2+(1-distb)*psi4b1) + x50
        ;
    if (abs(alpha(n)-1) < 0.05 )
       mu(n) = zeta;
    else
       mu(n) = zeta - beta(n)* sigma(n) * tan(0.5 * pi *alpha(n));
    end;
end;

% Correct estimates for out of range values
alpha(alpha <= 0) = 10^(-10)+0*alpha(alpha <= 0);
alpha(alpha > 2) = 2+0*alpha(alpha > 2);
sigma(sigma <= 0) = 10^(-10)+0*sigma(sigma <= 0);
beta(beta < -1) = -1+0*beta(beta < -1);
beta(beta > 1) = 1+0*beta(beta > 1);
```