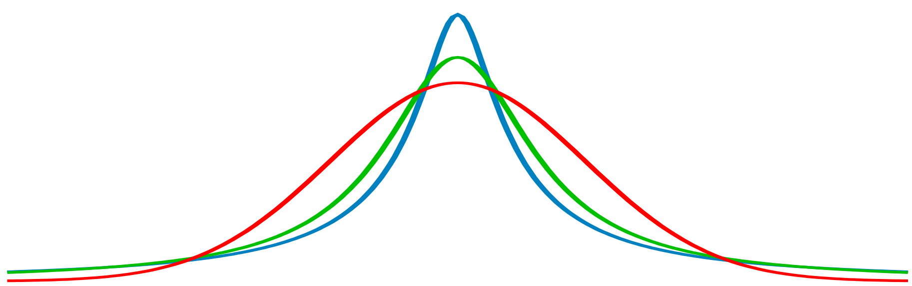
User Manual for STABLE 5.1
Signal Filtering Module 1.1
matlab Version



Abstract

This manual gives information about the STABLE library, which computes basic quantities for univariate stable distributions: densities, cumulative distribution functions, quantiles, and simulation. Statistical routines are given for fitting stable distributions to data and assessing the fit. Utility routines give information about the program and perform related calculations. Quick spline approximations of the basic functions are provided. Densities, cumulative distribution functions and simulation for discrete/quantized stable distributions are described.

The multivariate module gives functions to compute bivariate stable densities, simulate stable random vectors, and fit bivariate stable data. In the radially symmetric case, the amplitude densities, cumulative distribution functions, quantiles are computed for dimension up to 100.

The signal filtering module includes functions to compute non-linear filters for signals with heavy tailed noise. Specifically, a novel stable filter based on stably distributed noise is implemented.

6618 Allegheny Avenue
Takoma Park, MD 20912-4616 USA
phone and fax: 301.891.8484
www.RobustAnalysis.com
Revised 9 July 2009 (processed July 11, 2009)

Contents

1 Univariate Stable Introduction

Stable distributions are a class of probability distributions that generalize the normal distribution. Stable distributions are a four parameter family: α is the tail index, or index of stability, and is in the range $0 < \alpha \le 2$, β is a skewness parameter and is in the range $-1 \le \beta \le 1$, γ is a scale parameter and must be positive, and δ is a location parameter, an arbitrary real number. There are numerous meanings for these parameters. We will focus on two here, which we call the 0-parameterization and the 1-parameterization. The STABLE programs use a variable `param` to specify which of these parameterizations to use. If you are only concerned with symmetric stable distributions, the two parameterizations are identical. For non-symmetric stable distributions, we recommend using the 0-parameterization for most statistical problems, and only using the 1-parameterization in special cases, e. g. the one sided distributions when $\alpha < 1$ and $\beta = \pm 1$.

Since there are no formulas for the density and distribution function of a general stable law, they are described in terms of their characteristic function (Fourier transform). A random variable X is $S(\alpha, \beta, \gamma, \delta; 0)$ if it has characteristic function

$$E \exp(iuX) = \begin{cases} \exp\left(-\gamma^\alpha |u|^\alpha \left[1 + i\beta(\tan\frac{\pi\alpha}{2})(\operatorname{sign} u)(|\gamma u|^{1-\alpha} - 1)\right] + i\delta u\right) & \alpha \neq 1 \\ \exp\left(-\gamma |u| \left[1 + i\beta \frac{2}{\pi}(\operatorname{sign} u)\ln(\gamma|u|)\right] + i\delta u\right) & \alpha = 1. \end{cases} \quad (1)$$

A random variable X is $S(\alpha, \beta, \gamma, \delta; 1)$ if it has characteristic function

$$E \exp(iuX) = \begin{cases} \exp\left(-\gamma^\alpha |u|^\alpha \left[1 - i\beta(\tan\frac{\pi\alpha}{2})(\operatorname{sign} u)\right] + i\delta u\right) & \alpha \neq 1 \\ \exp\left(-\gamma |u| \left[1 + i\beta \frac{2}{\pi}(\operatorname{sign} u)\ln|u|\right] + i\delta u\right) & \alpha = 1. \end{cases} \quad (2)$$

Note that if $\beta = 0$, then these two parameterizations are identical, it is only when $\beta \neq 0$ that the asymmetry term (the imaginary factor involving $\tan \frac{\pi\alpha}{2}$ or $\frac{2}{\pi}$) becomes relevant. More information on parameterizations and about stable distributions in general can be found at `http://academic2.american.edu/~jpnolan`, which has a draft of the first chapter of Nolan (2010).

The next section gives a description of the different functions in STABLE.

2 Univariate Stable Functions

Interfaced STABLE functions require input variables, and return the results of the computations. The interface computes the lengths of all arrays, specifies default values for some of the variables in some case, and handles return codes and results.

The parameters of the stable distribution must be specified. In matlab, the 4 stable parameters are passed in a vector `theta=(alpha,beta,gamma,delta)`: the index of stability `alpha` must be specified; if fewer than four values are supplied, the omited values are replaced with defaults (0 for skewness `beta`, 1 for scale `gamma` and 0 for location `delta`).

The STABLE interface prints an error message when an error occurs. If an error occurs, execution is aborted; if a warning occurs, execution continues.

There is basic help information built into the interfaces. In matlab, type help before the command, e.g. `help stablepdf`, to get the function definition.

The STABLE library is not reentrant; on a single computer, only one user should be using the library at once.

The user should be aware that these routines attempt to calculate quantities related to stable distributions with high accuracy. Nevertheless, there are times when the accuracy is limited. If α is small, the pdf and cdf have very abrupt changes and are hard to calculate. When some quantity is small, e.g. the cdf of the light tail of a totally skewed stable distribution, the routines may only be accurate to approximately ten decimal places.

The remainder of this section is a description of the functions in the STABLE library.

2.1 Basic functions

2.1.1 Stable densities

matlab function: `stablepdf(x,theta,param)`

This function computes stable density functions (pdf): $y_i = f(x_i) = f(x_i|\alpha, \beta, \gamma, \delta; \texttt{param})$, $i = 1, \ldots, n$. The algorithm is described in Nolan (1997).

2.1.2 Stable distribution functions

matlab function: `stablecdf(x,theta,param)`

This function computes stable cumulative distribution functions (cdf): $y_i = F(x_i) = F(x_i|\alpha, \beta, \gamma, \delta; \texttt{param})$, $i = 1, \ldots, n$. The algorithm is described in Nolan (1997).

2.1.3 Stable quantiles

matlab function: `stableinv(p,theta,param)`

This function computes stable quantiles, the inverse of the cdf: $x_i = F^{-1}(p_i)$, $i = 1, \ldots, n$. The quantiles are found by numerically inverting the cdf.

Note that extreme upper tail quantiles may be hard to find because of subtractive cancelation: in double precision arithmetic $1 - p$ and 1 are indistinguishable for small p (less than approximately10^{-15}), STABLE will (correctly) return $F^{-1}(1 - p) = F^{-1}(1) = +\infty$ for most values of α and β. You can get better accuracy on the lower tails, where there is no subtractive cancelation: use the reflection property $F(x|\alpha, \beta) = 1 - F(-x|\alpha, -\beta)$.

Also note that the accuracy of the inversion is determined by two internal tolerances. (See Section 2.3.3.) (1) tolerance 10 is used to limit how low a quantile can be searched for. The default value is $p = 10^{-10}$: quantiles below p will be set to the left endpoint of the support of the distribution, which may be $-\infty$. Likewise, quantiles above $1 - p$ will be set to the right endpoint of the support of the distribution, which may be $+\infty$. (2) tolerance 2 is the relative error used when searching for the quantile. The search tries to get full precision, but if it can't, it will stop when the relative error is less than tolerance 2.

2.1.4 Simulate stable random variates

matlab function: `stablernd(n,theta,param)`

This function simulates n stable random variates: $x_1, x_2, \ldots, x_n$ with parameters $(\alpha, \beta, \gamma, \delta)$ in parameterization `param`. It is based on Chambers et al. (1976).

2.1.5 Stable hazard function

matlab function: `stablehazard(x,theta,param)`

This function computes the hazard function for a stable distribution: $h_i = f(x_i)/(1 - F(x_i))$, $i = 1, \ldots, n$.

2.1.6 Derivative of stable densities

matlab function: `stablepdfderiv(x,theta,param)`

This function computes the derivative of stable density functions: $y_i = f'(x_i) = f'(x_i|\alpha, \beta, \gamma, \delta; \texttt{param})$, $i = 1, \ldots, n$.

2.1.7 Second derivative of stable densities

matlab function: `stablepdfsecondderiv(x,theta,param)`

This function computes the second derivative of stable density functions: $y_i = f''(x_i) = f''(x_i|\alpha, \beta, \gamma, \delta; \texttt{param})$, $i = 1, \ldots, n$.

2.1.8 Stable score/nonlinear function

matlab function: `stablenonlinfn(x,theta,param)`

This function computes the score or nonlinear function for a stable distribution: $g(x) = -f'(x)/f(x) = -(d/dx) \ln f(x)$. The routine uses `stablepdf` to evaluate $f(x)$ and numerically evaluates the derivative $f'(x)$. Warning: this routine will give unpredictable results when $\beta = \pm 1$. The problems occur where $f(x) = 0$ is small; in this region calculations of both $f(x)$ and $f'(x)$ are of limited accuracy and their ratio can be very unreliable.

2.2 Statistical functions

2.2.1 Estimating stable parameters

matlab function: `stablefit(x,method,param)`

Estimate stable parameters from the data in $x_1, \ldots, x_n$, using `method` as described in the following table. This routine calls one of the functions described below to do the actual estimation.

method value	algorithm	notes
1	maximum likelihood	$\alpha \geq 0.4$
2	quantile	$\alpha \geq 0.1$
3	empirical characteristic function	$\alpha \geq 0.1$
4	fractional moment	$\alpha \geq 0.4$, $\beta = 0$, uses power $p = 0.2$
5	log absolute moment	$\beta = \delta = 0$
6	modified quantile	$\alpha \geq 0.4$
7	U statistic method	$\beta = \delta = 0$

Note that the fractional moment and log absolute moment methods do not work when there are zeros in the data set.

2.2.2 Maximum likelihood estimation

matlab function: `stablefitmle(x,param)`

Estimate the stable parameters for the data in $x_1, \ldots, x_n$, in parameterization `param` using maximum likelihood estimation. The likelihood is numerically evaluated and maximized using an optimization routine. This program and the numerical computation of confidence intervals below are described in Nolan (2001). For speed reasons, the quick log likelihood routine is used to approximate the likelihood; this is where the restriction $\alpha \geq 0.4$ comes from.

2.2.3 Maximum likelihood estimation with restricted parameters

matlab function: `stablefitmlerestricted(x,theta,param,restriction)`

This is a modified version of maximum likelihood estimation, where some parameters can be estimated while the others are restricted to a fixed value. The function receives theta = {alpha, beta, gamma, delta} and if `restriction[i]` = 1, then `theta[i]` is fixed.

2.2.4 Maximum likelihood estimation with search control

matlab function: `not implemented in matlab`

This is maximum likelihood estimation with greater control over the search and ranges for the parameters. It is used internally.

2.2.5 Quantile based estimation

matlab function: `stablefitquant(x,param)`

Estimate stable parameters for the data in x, using the quantile based on the method of McCulloch (1986). It sometimes has problems when α is small, say $\alpha < 1/2$, and the data is highly skewed. Try the modified version below in such cases.

2.2.6 Empirical characteristic function estimation

matlab function: `stablefitecf(x,gamma0,delta0,param)`

Estimate stable parameters for the data in x using the empirical characteristic function based method of Koutrovelis-Kogon-Williams, described in Kogon and Williams (1998). An initial estimate of the scale `gamma0` and the location `delta0` are needed to get accurate results. We recommend using the quantile based estimates of these parameters as input to this routine.

2.2.7 Fractional moment estimation

matlab function: `no direct interface, use stablefit with method=4`

Estimate stable parameters for the data in x, using the fractional moment estimator as in Nikias and Shao (1995). This routine only works in the symmetric case, it will always return $\beta = 0$ and $\delta = 0$. In this case the 0-parameterization coincides with the 1-parameterization, so there is no need to specify parameterization. `p` is the fractional moment power used. A reasonable default value is $p = 0.2$; take $p < \alpha/2$ to get reasonable results.

This method does not work if there are zeros in the data set - negative sample moments do not exist. Remove zero values (and possibly values close to 0) from the data set if you want to use this method.

2.2.8 Log absolute moment estimation

matlab function: `no direct interface, use stablefit with method=5`

Estimate stable parameters for the data in x, using the log absolute moment method as in Nikias and Shao (1995). This routine only works in the symmetric case, it will always return $\beta = 0$ and $\delta = 0$. In this case the 0-parameterization coincides with the 1-parameterization, so there is no need to specify parameterization.

The log absolute moment method does not work when there are zeros in the data set, because $\log |x|$ is undefined when x is 0. Remove zero values (and possibly values close to 0) from the data set if you want to use this method.

2.2.9 Quantile based estimation, version 2

matlab function: `no direct interface, use stablefit with method=6`

Estimate stable parameters for the data in x, using a modified quantile method of Nolan (2010). It should work for any values of the parameters, but some extreme values are inaccurate.

2.2.10 U statistic based estimation

matlab function: `no direct interface, use stablefit with method=7`

Estimate stable parameters for the data in x, using the method of Fan (2006). It only works for the symmetric case.

2.2.11 Confidence intervals for ML estimation

matlab function: `stablefitmleci(theta,n,z)`

This routine finds confidence intervals for maximum likelihood estimators of all four stable parameters. The routine returns a vector `sigtheta` of half widths of the confidence interval for each parameter. These values depend on the confidence level you are seeking, specified by z, and the size of the sample n. The z value is the standard critical value from a normal distribution, i.e. use $z = 1.96$ for a 95% confidence interval. For example, the point estimate of α is `theta[1]`, and the confidence interval is `theta[1]`±`sigtheta[1]`. For β, the confidence interval is `theta[2]`±`sigtheta[2]`, for γ, the confidence interval is `theta[3]`±`sigtheta[3]`, For δ, the confidence interval is `theta[4]` ± `sigtheta[4]`. These values do not make sense when a parameter is at the boundary of the parameter space, e.g. $\alpha = 2$ or $\beta = \pm 1$.

These values are numerically approximated using a grid of numerically computed values in Nolan (2001). The values have limited accuracy, especially when $\alpha \leq 1$.

2.2.12 Information matrix for stable parameters

matlab function: `stablemleinfomatrix(theta)`

Returns the 4×4 information matrix for maximum likelihood estimation of the stable parameters for parameter values `theta`. This is done in the continuous 0-parameterization. These are approximate values, interpolated from a grid of numerically computed values in Nolan (2001) for $\alpha \geq 0.5$. The values have limited accuracy, especially when $\alpha \leq 1$.

2.2.13 Log-likelihood computation

matlab function: `stableloglik(x,theta,param)`

Compute the log-likelihood of the data, assuming an underlying stable distribution with the specified parameters.

2.2.14 Chi-squared goodness-of-fit test

matlab function: `stablechisq(x,theta,nclasses,param)`

Compute chi-squared goodness-of-fit statistic for the data in $x_1, \ldots, x_n$ using `nclass` equally probable classes/bins.

2.2.15 Kolmogorov-Smirnov goodness-of-fit test

matlab function: `stableksgof(x,theta,method,param)`

This function computes the Kolmogorov-Smirnov two-sided test statistic:

$$D = \sup_{-\infty < x < \infty} |F(x) - \hat{F}(x)|,$$

where $F(\cdot)$ is the stable cdf with parameters α = `theta[1]`, β = `theta[2]`, γ = `theta[3]`, δ = `theta[4]` and $\hat{F}(\cdot)$ is the sample cdf of the data in x. Use `method`=0 for quick computations (`stableqkcdf` is used to compute cdf), use `method=1` for slower computations (`stablecdf` is used to compute cdf). The routine returns the observed value of D and an estimate of the tail probability $P(D > d)$, i.e. the significance level of the test. This tail probability is calculated using Stephen's approximation to the limiting distribution, e.g. $(n^{1/2} + 0.12 + 0.11n^{-1/2})D$ is close to the limiting Smirnov distribution. This is close to $n^{1/2}D$ for large n, and a better approximation on the tails for small n. Note, this calculation is not very accurate if the tail probability is large, but these cases aren't of much interest in a goodness-of-fit test. (If you don't like this approximation, the function returns D, and you can compute your own tail probability.) WARNING: the computation of the significance level is based on the assumption that the parameter values `theta`=$(\alpha, \beta, \gamma, \delta)$ were chosen independently of the data. If the parameters were estimated from the data, then this tail probability will be an overestimate of the significance level.

2.2.16 Likelihood ratio test

matlab function: `stablelrt(x,alphabnds,betabnds)`

This function computes the likelihood ratio L_0/L_1, where L_0 is the maximum likelihood of the data x under the assumption that x is an i.i.d. sample from a stable distribution with α and β restricted to the range `abnd[1]` $\leq \alpha \leq$ `abnd[2]` and `bbnd[1]` $\leq \beta \leq$ `bbnd[2]`, and L_1 is the maximum likelihood of the data under an unrestricted stable model. The function computes the maximum likelihood using the quick approximation to stable likelihoods, so is limited to α in the range [0.4,2].

The vector `results` will contain the results of the computations:

`results[1]` = ratio of the likelihoods
`results[2]` = -2*log(ratio of likelihoods)
`results[3]` = log likelihood of the data for the restricted H0
`results[4]` = log likelihood of the data for the unrestricted H1
`results[5]` = estimated value of alpha under H0
`results[6]` = estimated value of beta under H0
`results[7]` = estimated value of gamma under H0
`results[8]` = estimated value of delta under H0
`results[9]` = estimated value of alpha without assuming H0
`results[10]` = estimated value of beta without assuming H0
`results[11]` = estimated value of gamma without assuming H0
`results[12]` = estimated value of delta without assuming H0

Note that under the standard assumptions, `results[2]` converges to a chi-squared distribution with d.f. = (# free parameters in H1 parameter space - # free parameters in H0 parameter space) as the sample size tends to ∞.

For example, to compute the likelihood ratio test for the null hypothesis H0: data comes from a normal distribution vs H1: data comes from stable distribution, use `abnd`=(2,2) and `bbnd`=(0,0), in which case `results[2]` will have 2 d.f. To test H0: data comes from a symmetric stable distribution vs H1: data comes from a general stable distribution, use `abnd`=(0.4,2) and `bbnd`=(0,0), in which case `results[2]` will have 3 d.f.

2.2.17 Stable regression

matlab function: `stableregression(x,y,trimprob,symmetric)`

Computes regression coefficients $b_1, b_2, \ldots, b_k$ for the problem

$$y_i = b_1 x_{i,1} + b_2 x_{i,2} + \cdots + b_k x_{i,k} + e_i, \quad i = 1, \ldots, n$$

where the error term e_i has a stable distribution. In matrix form, the equation is $\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e}$. The algorithm is described in Nolan and Ojeda (2006).

y is a vector of length n of observed responses. x is a $n \times k$ matrix, with the columns of x representing the variables and the rows representing the different observations. NOTE: if you want an intercept term, you must include a column of ones in the x matrix. Typically one sets the first column of x to ones, and then b_1 is the intercept.

`trimprob` is a vector of length 2, e.g. (0.1,0.9), which gives the lower and upper quantiles for the trimmed regression. (Trimmed regression trims off extreme values and then performs ordinary least squares regression. The resulting coefficients are used to get an initial estimate of the stable regression coefficients.) `symmetric` can be used to force the fitting program to assume symmetry in the error terms e_i.

The interfaced versions of this function returns a structure with different fields.

- `b` is the vector of coefficients
- `binit` is the initial vector of coefficients from the trimmed regression
- `alpha,beta,gamma,delta` are the stable parameters estimated from the residuals. They can be regarded as nuisance parameters if you only care about the coefficients. Note that all parameters are in the 0-parameterization. You can convert to another representation using the convert parameterization function in Section 2.3.5.

Note that in the non-Gaussian stable case, some of the traditional assumptions in regression are no longer true. In particular, it is NOT always the case that $Ee_i = 0$. First, if $\alpha \leq 1$, the heavy tails will mean that Ee_i is undefined. Second, in the non-symmetric case, $\beta \neq 0$, even if $\alpha > 1$, we do not require $Ee_i = 0$. Instead, we set `delta` so that the mode of e_i is zero. The reason for this is to make the regression line go through the center of the data points.

2.2.18 Stable regression: profile likelihood parameter confidence intervals

matlab function: `not implemented in matlab`

Compute confidence intervals for regression parameters. This function uses profile likelihood for the specific data set to compute confidence intervals for each parameter, including the stable parameters α, β and γ as well as the regression coefficients $b_1, \ldots, b_k$. It is assumed that the user has already called the regression routine:

```
fit <- stable.regression(x,y,trimprob)
stable.regression.profile.likelihood(fit,x,y)
```

There are two optional arguments: `p.value` to specify the significance level (default `p.value`=0.05 gives 95% confidence intervals), and `show.plots` is a Boolean used to determine if plots of the profile likelihoods are shown for each parameter.

2.3 Informational/utility functions

2.3.1 Version information

matlab function: `stableversion`

This functions returns information of the version of STABLE that is being used. The values are:

`vinfo[1]`= major version number
`vinfo[2]`= minor version number
`vinfo[3]`= modification number
`vinfo[4]`= year of release
`vinfo[5]`= month of release

`vinfo[6]`= day of release
`vinfo[7]`= internal serial number

For example, the values "4 0 2 2005 9 15 123" mean that you are using version 4.0.2 of STABLE, which was released on 2005/9/15, with serial number 123.

`nv` is the length of the integer array `vinfo`. If `nv` is more than 7, other information may be filled into the `vinfo` array.

2.3.2 Modes of stable distributions

matlab function: `stablemode(theta,param)`

Returns the mode of a $S(\alpha = \texttt{theta[1]}, \beta = \texttt{theta[2]}, \gamma = \texttt{theta[3]}, \delta = \texttt{theta[4]}; \texttt{param})$ distribution. If $\beta \neq 0$, the mode is determined by a numerical search of the pdf.

2.3.3 Set internal tolerance

matlab function: `stablesettolerance(inum,value)`

Sets the value of an internal variable that is used during computations. You change these values at your own risk: computation times can become very long and some choices of the parameters can cause infinite loops.

inum	meaning
-1	enable internal warning messages (Warning: this will not work when used from R, Mathematica or matlab.)
0	relative error for pdf numerical integration
1	relative error for cdf numerical integration
2	relative error for quantile search
3	alpha and beta rounding
4	x tolerance near zeta
5	exponential cutoff
6	peak/strim location tolerance
7	stabletrim tolerance
8	minimum alpha
9	minimum xtol
10	threshold for quantile search
11	x tolerance

2.3.4 Get internal tolerance

matlab function: `stablegettolerance(inum)`

Returns the value of the internal settings, see the preceding function for the meanings of each variable.

2.3.5 Convert between parameterizations

matlab function: `stableconvert(iparam,thetai,jparam)`

Convert from the parameters given in `theta` given in the `param`-parameterization to the parameters `thetanew` given in the `newparam`-parameterization. Currently `param` and `newparam` are restricted to the values 0,1,2 and 3.

2.3.6 Omega function

matlab function: `stableomega(u, theta, param)`

Compute the function $\omega(u_i|\alpha,\beta;k)$, $i = 1,\ldots,n$ where

$$\omega(u|\alpha,\beta;0) = \begin{cases} |u|^\alpha \left[1 + i\beta(\tan\frac{\pi\alpha}{2})(\operatorname{sign} u)(|u|^{1-\alpha} - 1)\right] & \alpha \neq 1 \\ |u| \left[1 + i\beta\frac{2}{\pi}(\operatorname{sign} u)\ln|u|\right] & \alpha = 1, \end{cases} \tag{3}$$

$$\omega(u|\alpha,\beta;1) = \begin{cases} |u|^\alpha \left[1 - i\beta(\tan\frac{\pi\alpha}{2})(\operatorname{sign} u)\right] & \alpha \neq 1 \\ |u| \left[1 + i\beta\frac{2}{\pi}(\operatorname{sign} u)\ln|u|\right] & \alpha = 1. \end{cases}$$

These functions are from the characteristic functions of standardized univariate stable distributions: if $Z \sim \mathbf{S}(\alpha,\beta,1,0;k)$, then $E\exp(iuZ) = \exp(-\omega(u|\alpha,\beta;k))$. As before, $k = 0$ or $k = 1$ correspond to two different parameterization. The function returns two vectors containing the real and imaginary parts of $\omega(u|\alpha,\beta;k)$

2.4 Series approximations to basic distribution functions

These functions use the Bergstrom series for stable densities and cdfs, which are only defined for $\alpha \neq 1$.

2.4.1 Series approximation of stable pdf around the origin

matlab function: `stablepdfseriesorigin(x,nterms,theta,param)`

Computes the stable probability distribution function using a series approximation with `nterms` in it. This function is best used to calculate the density near the origin in the 1-parameterization. The series is not defined for $\alpha = 1$. Note that `nterms`=1 corresponds to a constant term, `nterms`=2 corresponds to a linear term, etc.

2.4.2 Series approximation of stable cdf around the origin

matlab function: `stablecdfseriesorigin(x,nterms,theta,param)`

Computes the stable cumulative distribution function using a series expansion with `nterms` in it. This function is best used to calculate the cdf near the origin in the 1-parameterization. The series is not defined for $\alpha = 1$. Note that `nterms`=1 corresponds to a constant term, `nterms`=2 corresponds to a linear term, etc.

2.4.3 Series approximation of stable pdf at the tail

matlab function: `stablepdfseriestail(x,nterms,theta,param)`

Computes the stable probability distribution function using a series approximation with `nterms` in it. This function is best used to calculate points on the tail of a distribution. The series is defined only for $x > 0$. (For $x < 0$, replace x by $-x$ and β by $-\beta$. The series is not defined for $\alpha = 1$.

2.4.4 Series approximation of stable cdf at the tail

matlab function: `stablecdfseriestail(x,nterms,theta,param)`

Computes the stable cumulative distribution function using a series approximation with `nterms` in it. This function is best used to calculate points on the tail of a distribution. The series is defined only for $x > 0$. (For $x < 0$, replace x by $-x$ and β by $-\beta$. The series is not defined for $\alpha = 1$.

2.5 Faster approximations to basic functions

The functions described in preceding sections are accurate, but can take a long time to compute. For evaluating a single pdf or cdf at a single set of parameter values, they are fine. However, when the functions must be evaluated many times, the previous routines are slow. For example, when estimating stable parameters by maximum likelihood estimation, the likelihood is evaluated at each data point for a large number of parameter values during the numerical search for the point where the likelihood is maximized. In these cases, speed is more desirable than great accuracy.

The functions described below are approximations to the functions above, and are based on pre-computed values using those basic functions. They are designed to evaluate the quantity of interest at many x values for fixed values of α and β. Each routine has a setup time, and if you change α or β, that setup code must be rerun. It can be slower to run these routines than the basic routines above if you only want to calculate the quantity at a few x values. These routines work for $0.2 \leq \alpha \leq 2$ and all $-1 \leq \beta \leq 1$.

2.5.1 Quick stable density computation

matlab function: `stableqkpdf(x,theta,param)`

Call is identical to Section 2.1.1, results are approximately the same.

2.5.2 Quick stable cumulative computation

matlab function: `stableqkcdf(x,theta,param)`

Call is identical to Section 2.1.2, results are approximately the same.

2.5.3 Quick stable log pdf computation

matlab function: `stableqkclogpdf(x,theta,param)`

Approximates log(pdf) for stable distributions. Results are approximately the same as $\log(f(x))$.

2.5.4 Quick stable quantile computation

matlab function: `stableqkinv(p,theta,param)`

Call is identical to Section 2.1.3, but much faster. Note the comments in that section about extreme upper quantiles.

2.5.5 Quick stable hazard function computation

matlab function: `stableqkhazard(x,theta,param)`

Call is identical to Section 2.1.5.

2.5.6 Quick stable likelihood computation

matlab function: `stableqkloglik(x,theta,param)`

Call is identical to Section 2.2.13.

2.5.7 Quick stable score/nonlinear function

matlab function: `stableqknonlinfn(x,theta,method,param)`

This function approximates the score or nonlinear function for a stable distribution: $g(x) = -f'(x)/f(x) = -(d/dx)\ln f(x)$. The algorithm used depends on the value of `method`. When `method`=1, `stableqkpdf` is used to compute $f(x)$ and in the numerical evaluation of $f'(x)$. When `method`=2, `stablescorefn` is used to compute $g(x)$ on a grid, then a spline is fit to those values. The resulting spline is used to approximate $g(x)$. If n is large, this is noticeably faster than either `stablescorefn` or `method=1` above. When `method`=3, a rational function approximation is used to approximate $g(x)$. This is the fastest method, but the accuracy depends on the values of `alpha` and `beta`. If `alpha` is between 1 and 1.9 and `beta` is near 0, the approximation is good.

2.6 Discrete stable distributions

Given a stable distribution $X \sim S(\alpha, \beta, \gamma, \delta; \texttt{param})$ and a pair of cutoff values $a < b$, the random variable

$$Y = \text{integer part of } \max(a, \min(X, b))$$

is a discrete stable distribution. These distribution arise in signal processing where a continuous quantity is quantized/digitized and limited accuracy is kept. It is assumed that the cutoff values are integers. The saturation probability is $P(X < a - 1/2) + P(X > b + 1/2)$, and is a measure of how much of the distribution is lost by truncating at the cutoff values. In the routines below, the cutoff is specified by a vector of length 2: `cutoff`=(a, b). In this section X will always refer to the continuous stable distribution, while Y will always refer to a discrete/quantized/integer valued distribution.

In the internal routines, the x values are integers. The matlab/R/Mathematica interfaces use double precision values.

2.6.1 Discrete stable density

matlab function: `stablepdfdiscrete(x,theta,cutoff,param)`

Calculates $f_i = P(Y = x_i)$, $i = 1, \ldots, n$.

2.6.2 Quick discrete stable density

matlab function: `stableqkpdfdiscrete(x,theta,cutoff,param)`

Calculates $f_i = P(Y = x_i)$, $i = 1, \ldots, n$. Faster than above, less accurate.

2.6.3 Discrete stable cumulative distribution function

matlab function: `stablecdfdiscrete(x,theta,cutoff,param)`

Calculates $F_i = P(Y \le x_i)$, $i = 1, \ldots, n$.

2.6.4 Quick discrete stable cumulative distribution function

matlab function: `stableqkcdfdiscrete(x,theta,cutoff,param)`

Calculates $F_i = P(Y \le x_i)$, $i = 1, \ldots, n$. Faster than above, less accurate.

2.6.5 Simulate discrete stable random variates

matlab function: `stablernddiscrete(n,theta,cutoff,param)`

Simulates discrete stable random variates with the specified parameters and cutoffs.

2.6.6 Simulate discrete stable random variates with specified saturation probability

matlab function: `stablernddiscrete2(n,theta,cutoff,psaturation,param)`

Simulates discrete stable random variates, where the scale is computed internally to make the saturation probability=`psaturation`. Note that in cases where the stable parameters are passed individually, `gamma` is NOT used. In the cases where the vector `theta` is used, the value of γ =`theta[3]` is ignored. The following function is used to compute γ, then the previous function is called to generate the values.

2.6.7 Find scale γ to have a specified saturation probability for a discrete stable distribution

matlab function: `stablediscretefindgamma(theta,cutoff,psaturation,param)`

Given α, β, δ and `cutoff`= (a, b), the scale γ is computed to get the requested saturation probability, e.g. `psaturation`=$P(X < a - 1/2) + P(X > b + 1/2)$.

2.6.8 Discrete maximum likelihood estimation

matlab function: `stablefitdmle(x,cutoff,method,param)`

Estimate the stable parameters for the discrete stable data in $x_1, \ldots, x_n$, in parameterization `param` using maximum likelihood estimation. The likelihood is numerically evaluated and maximized using an optimization routine. When `method`=1, `stablepdfdiscrete` is used to calculate likelihood, when `method`=2, symmetry is assumed ($\beta = 0$) and a faster method is used to compute the likelihood.

3 Multivariate Stable Introduction

To specify a multivariate stable distribution $\mathbf{X} = (X_1, X_2, \ldots, X_d)^T$ in d dimensions requires an index of stability $\alpha \in (0, 2]$, a finite Borel measure Λ on the unit sphere $\mathbb{S} = \{\mathbf{s} \in \mathbb{R}^d : |\mathbf{s}| = 1\}$ and a shift vector $\boldsymbol{\delta} \in \mathbb{R}^d$. The measure Λ is called the spectral measure of the distribution. The joint characteristic function of $\mathbf{X} \sim \mathbf{S}(\alpha, \Lambda, \boldsymbol{\delta}; k)$ is given by:

$$E \exp(i < \mathbf{u}, \mathbf{X} >) = \exp\left(- \int_{\mathbb{S}} \omega_k(< \mathbf{u}, \mathbf{s} > |\alpha, 1; k) \Lambda(d\mathbf{s}) + i < \mathbf{u}, \boldsymbol{\delta} > \right),$$

where $\omega(u|\alpha, \beta; k)$ is defined in (3). As in one dimension, the 1-parameterization is more common in theoretical research, while the 0-parameterization is better suited to computation and statistical problems. Here and below, $< \mathbf{u}, \mathbf{X} >= \mathbf{u}\mathbf{X}^T = u_1 X_1 + \cdots + u_d X_d$ is the inner product. Symmetric stable distributions are defined by the condition $\mathbf{X} \stackrel{d}{=} -\mathbf{X}$, which is equivalent to Λ being a symmetric measure on $\mathbb{S}$, i.e. $\Lambda(A) = \Lambda(-A)$ for any Borel subset $A \subset \mathbb{S}$. As in the univariate case, in the symmetric case the 0=parameterization and the 1-parameterization coincide.

The general case is beyond current computational capabilities, but several special cases: isotropic (radially symmetric), elliptical, independent components and discrete spectral measure are computationally accessible.

isotropic The spectral measure is continuous and uniform, leading to isotropic/radial symmetry for the distribution. The characteristic function is

$$E \exp(i < \mathbf{u}, \mathbf{X} >) = \exp\left(-\gamma_0^\alpha |\mathbf{u}|^\alpha + i < \mathbf{u}, \boldsymbol{\delta} >\right). \tag{4}$$

elliptical The characteristic function is

$$E \exp(i < \mathbf{u}, \mathbf{X} >) = \exp\left(-(\mathbf{u}^T R \mathbf{u})^{(\alpha/2)} + i < \mathbf{u}, \boldsymbol{\delta} >\right) \tag{5}$$

where R is a positive definite matrix. ($R = \gamma_0^2 I$ is equivalent to the isotropic case above.)

independent components If components are independent with $X_j \sim \mathbf{S}(\alpha, \beta_j, \gamma_j, \delta_j; k)$, then the characteristic function is

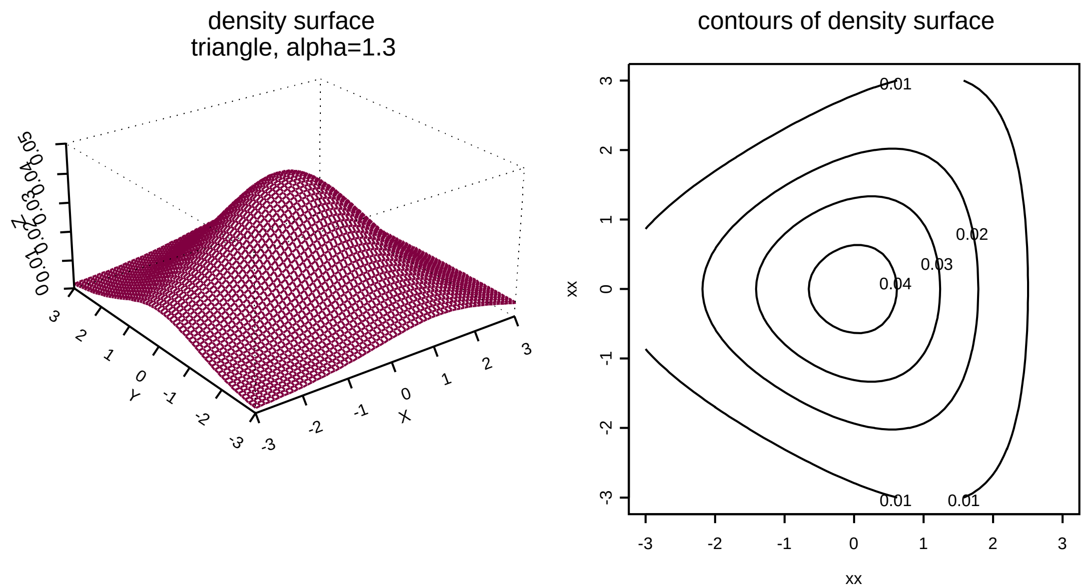
$$E \exp(i < \mathbf{u}, \mathbf{X} >) = \exp\left(- \sum_{j=1}^{d} \omega(u_j|\alpha, \beta_j; k)\gamma_j^\alpha + i < \mathbf{u}, \boldsymbol{\delta} > \right) \tag{6}$$

This is a special case of the discrete spectral measure below: the spectral mass is concentrated on the points where the coordinates axes intersect the unit sphere.

discrete When the spectral measure is discrete with mass λ_j at $\mathbf{s}_j \in \mathbb{S}$, $j = 1, \ldots, m$ the characteristic function is

$$E \exp(i < \mathbf{u}, \mathbf{X} >) = \exp\left(- \sum_{j=1}^{m} \omega(< \mathbf{u}, \mathbf{s}_j > |\alpha, 1; k)\lambda_j + i < \mathbf{u}, \boldsymbol{\delta} > \right) \tag{7}$$

This discrete class is dense in the class of all stable distributions: any finite spectral measure Λ can be approximated by a discrete measure, see Byczkowski et al. (1993). Below is a plot of the density surface of a bivariate stable density with three point masses, each of weight 1 at locations $(\cos(\pi/3), \sin(\pi/3))$, (-1,0), and $(\cos(5\pi/3), \sin(5\pi/3))$



4 Multivariate Stable Functions

Since the specification of a multivariate stable distribution is somewhat cumbersome, a different approach from the univariate case is taken in these routines. Two steps are needed to work with a multivariate stable distribution. First, the distribution is specified by calling a function to define the distribution. Second, call a separate functions to compute densities, cumulatives, simulate, etc.

The programs for working with multivariate stable distributions are less well developed and generally limited to 2 dimensions. At the current time, when dimension is greater than 2, you can: (a) simulate using mvstablernd, (b) calculate the pdf using mvstablepdf if the components are independent OR the spectral measure has exactly d point masses, (c) calculate the cdf using mvstablecdf if the components are independent, or (d) calculate the cdf using mvstablecdfMC by Monte Carlo estimation for any type of distribution.

The accuracy of the pdf and cdf calculations are limited. In all cases, X is a column vector, this is important to remember when you specify x for calculating, say, a pdf.

4.1 Define multivariate stable distribution

The interfaced versions of STABLE have the ability to work with multiple distributions. When a multivariate stable distribution is defined, a 'distribution descriptor' is returned. That descriptor must be used when computing quantities for that distribution. Note: The descriptor should not be changed by a user. The descriptor may change between calls, and contents may vary in future versions of STABLE.

There are different functions used to define each of the different types of distributions that STABLE can work with. They are described below.

A simple matlab example that defines two distributions and works with them is:

```
% Define two bivariate stable distributions: one isotropic and
```

```
% one with indep. components
dist1 = mvstableisotropic( 1.3, 2, 1, [ 0 0] );
dist2 = mvstableindep( 1.3, [0 0], [1 1], [0 0], 0 );

% compute the pdf for both distributions:
x = [0 0 1 1; 0 1 0 1];
y1 = mvstablepdf(dist1,x);
y2 = mvstablepdf(dist2,x);

% simulate from both distributions
z1 = mvstablernd(dist1,10000);
z2 = mvstablernd(dist2,10000);
```

4.1.1 Independent components

matlab function: `mvstableindep(alpha,beta,gamma,delta,param)`

Define a multivariate stable distribution with independent components with characteristic function (6). `beta`, `gamma` and `delta` should be vectors of length `d`= the dimension of the distribution.

4.1.2 Isotropic stable

matlab function: `mvstableisotropic(alpha,d,gamma0,delta, iparam)`

Define a multivariate isotropic stable distribution with characteristic function (4). `d` is the dimension of the distribution, `gamma0` is the scale parameter, `delta` is the location vector.

4.1.3 Elliptical stable

matlab function: `mvstableelliptical(alpha, R, delta, iparam)`

Define a multivariate elliptically contoured/sub-Gaussian stable distribution with characteristic function (5). The dimension of the distribution is determined from the size of `R`, a positive definite $d \times d$ shape matrix, and `delta` is the location vector.

4.1.4 Discrete spectral measure

matlab function: `mvstablediscspecmeas(alpha,s,lambda,beta,delta,param)`

Define a multivariate stable distribution with discrete spectral measure having characteristic function (7). `s` should be a `d` × `nlambda` matrix specifying the location of the point masses as columns, `lambda` should be a row vector of length `nlambda` containing the weights. `beta` should be a row vector of length `nlambda` specifying the skewness at each point mass. `delta` is the shift as a column vector. `param` is the parameterization, must be 0 or 1.

The spectral measure is defined by putting mass `lambda`[j]*(1+`beta`[j])/2 at $\mathbf{s}_j$ and mass `lambda`[j]*(1-`beta`[j])/2 at $-\mathbf{s}_j$. Setting all `beta` equal to 1 gives the standard definition of a spectral measure, with mass `lambda[j]` at `s[j]`. Setting all `beta` equal to 0 guarantees that the distribution is symmetric, putting weight `lambda`[j]/2 at $\pm\mathbf{s}_j$. If any element of `beta` is not 0, the distribution is assumed to be nonsymmetric. (It is possible to manually make a spectral measure symmetric with nonzero `beta` by defining antipodal points and weights and values of `beta` that balance correctly. However, STABLE does not detect this.) Some parts of STABLE are significantly faster and more accurate in the symmetric case, e.g. density calculations and simulations.

4.1.5 Discrete spectral measure in 2 dimensions

matlab function: `mvstablediscspecmeas2d(alpha,angle,lambda,beta,delta,param)`

Define a bivariate stable distribution with discrete spectral measure. This is a special case of the previous function. In two dimensions the locations of the point masses can be specified by angles: `angle`[j] gives the angle (in radians) of the location of $\mathbf{s}_j = (\cos(\texttt{angle}[j]), \sin(\texttt{angle}[j]))$.

There are several special cases that are handled differently internally:

- When `angle` and `lambda` are of length 2, densities can be calculated in terms of univariate densities.
- The special case of the previous one is when `angle`=(0, $\pi/2$). This corresponds to a distribution with independent components. Both density and cdf are calculated in terms of products of univariate density and cdf respectively.
- If all elements of `beta` are 0, the distribution is symmetric. Cumulative distribution function calculations only work in the symmetric case (though Monte Carlo based cdf estimation works for any case you can simulate, including skewed.)

4.1.6 Undefine a stable distribution

matlab function: `mvstableundefine(dist)`

Clears the definition of the stable distribution `dist`.

4.2 Basic functions

4.2.1 Density function

matlab function: `mvstablepdf(dist,x)`

Computes the density $f(\mathbf{x})$ for stable distribution `dist` at each value in `x`. Note: this routine assumes that the density exists. The density will not exist in the discrete spectral measure case if the mass is concentrated on a proper subspace of the domain.

In the independent case, the program computes the pdf as a product of univariate stable pdfs. There is one other case that can be evaluated in terms of univariate pdfs: if the spectral measure is discrete AND the number of point masses is equal the dimension of the problem.

Otherwise, only 2-dimensional computations can be done. The symmetric case uses the method in Abdul-Hamid and Nolan (1998), the nonsymmetric case uses the method in Nolan and Rajput (1995). The symmetric case is faster and more accurate than the nonsymmetric case. Both routines are accurate near the center of the distribution, and have limited accuracy near the tails.

4.2.2 Cumulative function

matlab function: `mvstablecdf(dist,a,b)`

This function approximates $P(\mathbf{a} \leq \mathbf{X} \leq \mathbf{b})$. If the components are independent, it computes this by taking the product of the corresponding univariate probabilities.

In the symmetric two-dimensional case, the probability is evaluated by numerically integrating the (numerically computed) 2-dimensional density $f(\mathbf{x})$. Due to the limited precision in the numerical calculation of the density, and the approximate nature of the integration of this density, this routine gives only a few digits of accuracy. To find the probability of an unbounded regions, it is best to truncate the region using the routine in Section 4.2.5 to find a bounded rectangle containing most of the probability.

Use the function in Section 4.2.3 to approximate in 2-dimensional nonsymmetric case or in higher dimensions.

4.2.3 Cumulative function (Monte Carlo)

matlab function: `mvstablecdfMC(dist,a,b,n)`

This function approximates $P(\mathbf{a} \le \mathbf{X} \le \mathbf{b})$ by simulating n indepedent random vectors with the same distribution as $\mathbf{X}$ and counting how many are in the interval $[\mathbf{a}, \mathbf{b}]$. It works for any distribution and dimension that can be simulated.

4.2.4 Multivariate simulation

matlab function: `mvstablernd(dist,n)`

Simulate `n` stable random vectors from the stable distribution `dist`. This works for any distribution that can be defined in dimensions $d \ge 2$.

4.2.5 Find a rectangle with probability at least p

matlab function: `mvstablefindrectangle(dist,p)`

Find a number r so that the rectangle $A = A(r) = [-r, r] \times [-r, r]$ has $P(\mathbf{X} \in A) \ge p$, where $\mathbf{X}$ is a bivariate stable distribution defined by `dist`. This is used for technical calculations, e.g. in approximating the probability of unbounded regions. The method uses univariate projections and will generally give an overestimate of r. The method is less accurate for small p or if the distribution is not centered or highly skewed, it gets more accurate if p is close to 1 and the distribution is centered and symmetric.

If p is not too close to 1, one can get a better value of r by making repeated calls to the multivariate cdf function with rectangles of the form $A(r)$ and search for a value of r that makes $P(\mathbf{X} \in A(r))$ close to p. That procedure involves bivariate numerical integration will take much longer than this function.

4.3 Statistical functions

4.3.1 Estimate a discrete spectral measure - fit a stable distribution to bivariate data

matlab function: `mvstablefit(x,nspectral,method1d,method2d,param)`

`x` contains the data values, `nspectral` is the number of points in the estimated spectral measure (must be divisible by 4), `method1d` is the method to use for estimating univariate stable parameters internally (see Section 2.2.1 for codes; only used if method2d=1), `method2d` is the method to use in estimating bivariate distribution. Use `method2d`=1 for Rachev-Xin-Cheng method, `method2d`=2 for projection method, `method2d`=3 for empirical characteristic function method. The methods are described in Nolan et al. (2001), see Nolan and Panorska (1997) for some discussion of suggested values and diagnostics. Suggest using `nspectral`=40, `method1d`=3, `method2d`=2, `param`=1.

The function returns a list/structure that contains information about the fit, which is always done as a discrete spectral measure. The fields in the fit are: the estimated value of α, the estimated shift/location vector $\boldsymbol{\delta}$, `angle` which is a uniform grid from 0 to 2π of length `nspectral`, and `lambda` for the estimated weights at each position.

4.3.2 Estimate parameter functions

matlab function: `mvstablefitparfn2d(x,angle,method1d,param)`

Estimate the parameter functions for the bivariate data in `x`. The data is projected in each direction given by `angle` and the parameters are estimated in the `param` parameterization. `method1d` is the univariate method used to estimate the parameters (see Section 2.2.1 for codes).

The result is a matrix of dimension length(`x`)$\times$5. The columns of the result are (1) for the angle, (2) for the estimate of α, (3) for the estimate of β, (4) for the estimate of γ, (5) for the estimate of δ at that angle.

4.3.3 Fit an elliptical stable distribution to multivariate data

matlab function: `mvstablefitelliptical(x,method1d)`

`x` contains the data values, `method1d` is the method to use for estimating univariate stable parameters internally (see Section 2.2.1 for codes).

The function returns a list/structure that contains information about the fit. The fields in the fit are: the estimated value of α, the estimated shift/location vector $\boldsymbol{\delta}$, and `R` for the estimated shape matrix.

4.4 Amplitude distribution

For d-dimensional random vector $\mathbf{X}$, the univariate quantity $R = |\mathbf{X}|$ is called the amplitude of $\mathbf{X}$. When $\mathbf{X}$ is isotropic, the radial symmetry allows one to reduce the dimension of the problem to a univariate problem. The following routines compute the cdf, pdf, quantiles, simulate and estimate for amplitudes of isotropic stable random vectors. Since these are univariate quantities, and it is required that the distribution is isotropic, one does NOT have to define the isotropic distribution separately. Because of computational delicacy, these routines are limited to dimension $d \leq 100$.

4.4.1 Amplitude cumulative distribution function

matlab function: `mvstableamplitudecdf(r,alpha,gamma0,dim)`

Compute the cdf of the amplitude distribution: $F_R(r) = P(R \leq r)$ for $R = |\mathbf{X}|$, where $\mathbf{X}$ is an d dimensional isotropic stable random vector with characteristic function $E\exp(i < \mathbf{u}, \mathbf{X} >) = \exp(-\gamma_0^\alpha |u|^\alpha)$. Current implementation works for $\alpha \in [0.8, 2]$. There seems to be a relative error of approximately 3% for large r.

4.4.2 Amplitude density

matlab function: `mvstableamplitudepdf(r,alpha,gamma0,dim)`

Compute the density $f_R(r)$ where R is described above. Current implementation works for $\alpha \in [0.8, 2]$. There seems to be a relative error of approximately 3% for large r.

4.4.3 Amplitude quantiles

matlab function: `mvstableamplitudeinv(p,alpha,gamma0,dim)`

Compute the quantiles of the amplitude R described above. Current implementation works for $\alpha \in [0.8, 2]$.

4.4.4 Simulate amplitude distribution

matlab function: `mvstableamplitudernd(nr,alpha,gamma0,dim)`

Simulate n i.i.d. values of the amplitude distribution R as described above. Current implementation works for $\alpha \in (0.2, 2]$.

4.4.5 Fit amplitude data

matlab function: `mvstablefitamplitude(r,dim,method)`

Estimate the parameters α and γ_0 for amplitude data. `r` contains the (univariate) amplitude data values, `d` is the dimension of the underlying distribution that the amplitude data comes from, `method` is the method to use for estimating. This initial implementation allows only `method=5`, which uses method of moments on the $\log$ of the amplitude data. Other methods are planned for the future. The function returns the estimated value of α and γ_0.

4.4.6 Amplitude score function

matlab function: `mvstableamplitudescore(r,alpha,gamma0,d)`

Compute the score function of the amplitude: $g(r) = -f'(r)/f(r)$, where $f(r) = f_R(r|\alpha, \gamma_0, d)$ is the amplitude pdf defined above. Current implementation works for $\alpha \in [0.8, 2]$, $d \le 98$. There seems to be a relative error of approximately 3% for large r.

4.5 Faster approximations to multivariate routines

There are a limited number of functions for quickly calculating multivariate functions in the 2-dimensional isotropic case. Such a distribution is specified by the index of stability α, the scale γ_0, and the location $\boldsymbol{\delta} = (\delta_1, \delta_2)$. Because the description is simple, these functions use those arguments directly and do not use a distribution descriptor.

4.5.1 Quick log-likelihood for bivariate isotropic case

matlab function: `mvstableqkloglikisotropic2d(x,alpha,gamma0,delta)`

Compute the log likelihood of the bivariate isotropic stable data in `x` with stable index `alpha`, scale `gamma0`, and location vector `delta`. An internal approximation is used to compute the single value

$$\ell(\alpha, \gamma_0, \boldsymbol{\delta}|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \log \prod_{i=1}^{n} f_{\mathbf{X}}(\mathbf{x}_i|\alpha, \gamma_0, \boldsymbol{\delta}).$$

This function is designed to compute the log likelihood for a fixed α many times. In this case, it is much faster than trying to compute the right hand side above using the bivariate pdf routine in Section 4.2.1. It is also more accurate than that routine, especially on the tails. The program initially computes an approximation that depends on α; if α changes, the approximation must be recomputed and it will be slower.

4.5.2 Quick amplitude density in bivariate case

matlab function: `mvstableqkamplitudepdf2d(r,alpha,gamma0)`

Compute the amplitude function $f_R(r|\alpha, \gamma_0, d = 2)$ for a 2-dimensional isotropic stable vector . For large `n`, it is much faster than the function in Section 4.4.2.

4.6 Bivariate discrete stable distribution

A bivariate discrete stable distribution is defined by digitizing and truncating a continuous bivariate stable distribution $\mathbf{X} = (X_1, X_2)^T$: discrete $\mathbf{Y} = (Y_1, Y_2)^T$ has components

$$Y_i = \text{integer part of } \max(a, \min(X_i, b)),$$

where `cutoff=`(a, b) are the upper and lower cutoff values. Note that the same cutoff is used for both components of $\mathbf{X}$. These distributions arise in signal processing where a bivariate continuous quantity is quantized/digitized and limited accuracy is kept. It is assumed that the cutoff values are integers. The saturation probability is $p_{sat} = P(X_1 < a - 1/2) + P(X_1 > b + 1/2) + P(X_2 < a - 1/2) + P(X_2 > b + 1/2)$, and is a measure of how much of the distribution is lost by truncating at the cutoff values.

In the internal routines, the x values are integers. The R, Mathematica and matlab interfaces store these integer values in double precision numbers.

4.6.1 Discrete bivariate density

matlab function: `mvstablepdfdiscrete2d(dist,x,cutoff,eps,method)`

Compute the pdf of a discrete bivariate stable distribution. `x` should be a $2 \times n$ matrix of integer values, `cutoff` is a vector of length 2 with upper and lower cutoff values for the truncation. The typical value for `cutoff` is (-128,127); both components of (X_1, X_2) are truncated at the same value. The function returns a vector `p` of length n with

$$p_i = P(\mathbf{Y} = \mathbf{x}_i) = P(Y_1 = x_{1i}, Y_2 = x_{2i}).$$

Note that `eps` is the attempted accuracy for each probability p_i, not for the total error. The probabilities are computed using the bivariate cdf function above and thus only works for symmetric stable two dimensional distributions. It's accuracy is limited: it is likely that when all possible values of `x` are used, $\sum_i p_i$ will be slightly different from 1. The current implementation is slow. The `method` variable is unused at the current time; it will be used for faster approximations in future implementations.

4.7 Multivariate informational/utility functions

4.7.1 Information about a distribution

matlab function: `mvstableinfodist`

Returns information about distribution `dist`. Useful for checking that definition.

4.7.2 Compute projection parameter functions

matlab function: `mvstableparfn2d(dist,angle)`

Compute the exact parameter functions for a bivariate stable distribution. For direction $\mathbf{t} \in \mathbb{R}^2$, $(\mathbf{X}, \mathbf{t}_j)$ is univariate stable with parameters $(\alpha, \beta(\mathbf{t}), \gamma(\mathbf{t}), \delta(\mathbf{t}))$. This function computes the parameter functions $\beta(\cdot)$, $\gamma(\cdot)$ and $\delta(\cdot)$ at the values $\mathbf{t} = (\cos \texttt{angle}[j], \sin \texttt{angle}[j])$. Angles in `angle` are given in radians.

4.7.3 Multivariate convert parameterization

matlab function: `mvstableconvert(dist,newparam)`

Converts between multivariate stable parameterizations. `newparam` must be 0 or 1. In the interfaced versions of STABLE, the input distribution `dist` is converted to the new parameterization, and a new distribution descriptor is returned.

5 Signal Filtering Introduction

The standard additive noise model for a signal is

$$s_t = x_t + n_t, \quad t = 1, 2, 3, \ldots \tag{8}$$

Here the x_t are some signal we are interested in, and the n_t are noise terms that corrupt the signal. This noise can be caused by natural events like lightening, sea clutter in naval radar systems, animal sounds (snapping shrimp) underwater, or man made sources like electro-mechanical noise in urban environments. The goal is to recover the unknown signal x_t as well as possible. This is done by computing an estimate $\widehat{s}_t$ using a sliding window of the data, centered at t.

The traditional linear filter uses a window of width m to compute an estimate for each window: $\widehat{s}_t = \widehat{\theta}_{\mathrm{LINEAR}}(s_{t-k_1}, s_{t-k_1+1}, \ldots, s_{t+k_2})$, where $k_1 = \lfloor m/2 \rfloor$, $k_2 = m - k_1$ and

$$\widehat{\theta}_{\mathrm{LINEAR}}(s_1, \ldots, s_m) := (s_1 + s_2 + \cdots + s_m)/m. \tag{9}$$

The well established theory of linear filter shows this is optimal when $\alpha = 2$, but experience shows that the linear filter can be severely degraded when $\alpha < 2$. As is well known in the statistics literature, extreme values of the noise terms n_t can have a large effect on the sample mean. This is illustrated in Figure 1

Models built on a stable distribution yield a non-linear filtering technique that is optimal for the case when the noise terms are stable. These filters are also robust, working well with other heavy tailed distributions. Let $\rho(x) = -\log f(x)$ be the negative of the log density, and define a cost function

$$C(\theta; s_1, \ldots, s_m) = C_{\mathrm{unweighted}}(\theta; s_1, \ldots, s_m) = \sum_{i=1}^{m} \rho(s_i - \theta) \tag{10}$$

We define the stable filter to be the value of θ that minimizes the cost: $\widehat{s}_t = \widehat{\theta}_{\mathrm{STABLE}}(s_{t-k_1}, s_{t-k_1+1}, \ldots, s_{t+k_2})$, where

$$\widehat{\theta}_{\mathrm{STABLE}}(s_1, \ldots, s_m) = \arg\min_{\theta} C(\theta; s_1, s_2, \ldots, s_m). \tag{11}$$

Since $\rho(x) = -\log f(x)$, the minimum of the cost function is exactly the maximum likelihood estimate of the location parameter θ. Note that in the Gaussian $\alpha = 2$ case, $\rho(x) = -x^2/2$, and the minimum can be found explicitly; it is simply $\widehat{\theta}_{\mathrm{LINEAR}}$. For $0 < \alpha < 2$, the filters are nonlinear with no closed formula, and the minimum in (11) must be found numerically.

There are several generalizations of the filter that are given by modifying the cost function (10). The simplest extension is to allow non-negative weights w_i:

$$C_{\mathrm{weighted}}(\theta; s_1, \ldots, s_m) = \sum_{i=1}^{m} \rho(w_i(s_i - \theta)) \tag{12}$$

In detection problems, one looks for a known pattern $x_1, \ldots, x_n$ in the signal, which is contaminated by stable noise:

$$s_t = \theta x_t + n_t, \quad t = 1, 2, 3, \ldots$$

The null case corresponds to $\theta = 0$ (no signal), the case where a signal is present corresponds to $\theta \neq 0$. There are two ways to implement this. The first is to allow signed weights as in Arce (2005). This is an approximation to a matched filter, using cost function

$$C_{\mathrm{signed}}(\theta; s_1, \ldots, s_m) = \sum_{i=1}^{m} \rho(|x_i|((\operatorname{sign} x_i)s_i - \theta)) = \sum_{i=1}^{m} \rho(x_i s_i - |x_i|\theta). \tag{13}$$

The second way is a true stable matched filter, with cost function

$$C_{\mathrm{matched}}(\theta; s_1, \ldots, s_m) = \sum_{i=1}^{m} \rho(s_i - \theta x_i). \tag{14}$$

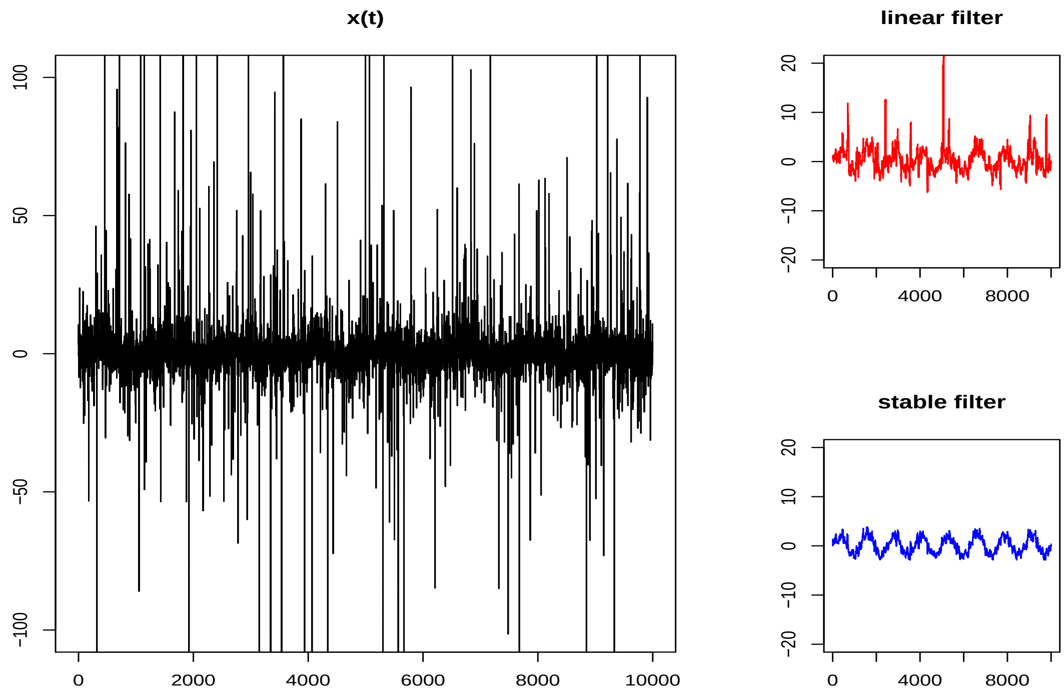


Figure 1: Signal filtering with symmetric stable noise, $\alpha = 1.3$, $\gamma = 2$, $n = 10000$ samples, and window width $m = 50$. The large graph shows a sinusoidal signal with simulated noise, the smaller plots show the output of a linear filter and a (unweighted) stable filter. Note the difference in the scales for the input and the output.

The $\widehat{\theta}$ obtained by minimizing (14) gives an estimate of the strength of the original pattern in the signal.

The STABLE Signal Filtering module implements these filters, making it possible to optimally deal with heavy tailed noise. The implementation of these filters involves numerous computational difficulties. The difficulty of computing the relevant cost function is resolved by the functions in the STABLE univariate module. A second problem is nonconvexity: if $\alpha \neq 2$, the $\rho(x)$ function above is not convex, so the cost function is not convex. In particular, there can be multiple local minima of the cost function, and a reliable filter should find the global min. Finally, filters should be fast. While we cannot match the speed of a simple linear filter, stable filters provided by the STABLE program makes it possible to use non-linear filters in practical problems for the first time.

The STABLE Signal Filtering routines also implement the linear filter, myriad filter, selection filter, and the median filter. More information on non-linear filters can be found in Arce (2005), Nunez et al. (2008), Nolan (2008), Pitas and Venetsanopoulos (1990), and Astola and Kuosmanen (1997).

5.1 The filter information structure

When a stable filter is used, numerous pieces on information must be supplied. The STABLE routine uses a structure called `filterinfo` to define all the parameters of the filter. Not all of these are required for every filter. Most users will use BASIC mode with a minimum number of parameters. Expert users can use ADVANCED mode, but must be careful about specifying all the needed parameters. What values are needed are specified in the individual function definitions below.

The following are the fields of `filterinfo`:

- `type` : The type of filter chosen. It can be 'MEAN_FILTER', 'MEDIAN_FILTER', 'MYRIAD_FILTER', 'SELECTION_FILTER' or 'STABLE_FILTER'.

- `mode` : 'BASIC' filter mode or 'ADVANCED' filter mode. In the BASIC mode, only required parameters are needed and defaults are used for the others. In ADVANCED mode, the caller must specify all parameters needed by the filter requested.
- `rhofn` : Rho function to use in cost function calculations. It can be 'RHO_STABLELOGLIK' (slowest), 'RHO_STABLEQKLOGLIK' (medium speed) or 'RHO_STABLEQKLOGPDFGRID_LINEAR' (fastest).
- `costfn` : Form of the cost function. It can be 'COST_UNWEIGHTED', 'COST_NONNEG_WEIGHTED', 'COST_SIGNED_WEIGHTED' or 'COST_MATCHED'. These corresponds to equations (10), (12), (13), and (14), respectively.
- `searchmethod` : Search/minimization method to use. It can be 'FIXEDPOINTSEARCH_I', 'FIXEDPOINTSEARCH_II', 'BRANCHANDBOUND_LIP' or 'BRANCHANDBOUND_LOC_BND'.
- `queuesize` : Length of queue in Branch and Bound search.
- `iternum` : Number of iterations.
- `param` : Parameterization of stable distribution. If `beta` is not zero, `param` must be 2. Recall that the 2 parameterization is centered at the mode of the stable density.
- `alpha` : Alpha in the stable distribution.
- `beta` : Beta in the stable distribution.
- `gamma` : Gamma in the stable distribution.
- `xtol` : Tolerance as stopping criteria of searchmethod.
- `lipconst` : Lipschitz constant for Branch and Bound Lipschitz.
- `init` : Theta initialization criteria. It can be 'USER_DEFINED' or 'FIXED'.
- `stop` : Stopping criteria in the adaptive filter. It can be 'VARY' or 'FIXED'.
- `x0` : initial point for search

6 Signal Filtering Functions

6.1 Filter functions

6.1.1 Stable Signal filtering

matlab function: `stablesigfilter(data, padding, weights, filterinfo)`

Implement a sliding window operation over a 1-dimensional signal. STABLESIGFILTER passes a window over the input data selecting successive windows. On each window, the filtering operation defined in `filterinfo` is performed.

The vectors `data` contains the observed data points $s_1, \ldots, s_n$ and `weights` contains the weights. The parameter `padding` defines how the end extremes of the input data are treated. Several extension modes are possible and represent different ways of handling the problem of border distortion in the analysis. In any case, floor($(n-1)/2$) samples are appended to the signal at beginning and ceil($(n-1)/2$) at the end. These modes are:

- 'ZERO_PADDING' : Adds zeros at the signal extremes.

- 'CONSTANT_PADDING' : The first sample value and the last value are repeated at the beginning and at the end, respectively.
- 'SYMMETRIC_PADDING' : Symmetric replication at both ends (padding is a reversed image of data at that endpoint).
- 'PERIOD_PADDING' : Periodic extension at both ends (padding comes from other end of data).
- 'UNFILTERED_PADDING' : Leave unfiltered the first floor$((n-1)/2)$ samples and the last ceil$((n-1)/2)$ samples. Thus, the filter output for the unfiltered samples are the corresponding input samples.

The type of filtering done and it's associated parameters are specified in the structure `filterinfo`. All filters require two fields:

The `type` field must be filled in with one of the following: 'MEAN_FILTER', 'MEDIAN_FILTER', 'MYRIAD_FILTER', 'SELECTION_FILTER' or 'STABLE_FILTER'.

For the myriad, selection or stable filter, other fields must be specified as described in the respective section below. The functions below each take a single window and compute a single value as the output of the filter for that window.

6.1.2 Mean filter

matlab function: `meanfilter(s,w)`

Calculates the weighted mean of a set of n samples S_i where each sample is weighted by S_i. The mean of the set of samples is $\sum_{i=1}^{n} w_i\, s_i$.

6.1.3 Median filter

matlab function: `medianfilter(s,w)`

Calculates the weighted median of a vector of samples `s`. The output of the weighted median for nonnegative integer weights w_i is given by:

$$\mathrm{MEDIAN}(s_1,\, s_2, \cdots, s_n; w_1,\, w_2, \cdots, w_n) \;=\; \mathrm{MEDIAN}\{s_1 \diamond w_1,\, s_2 \diamond w_2, \cdots, s_n \diamond w_n\},$$

where $\diamond$ represents the repetition operator, i.e. repeat value s_i w_i times.

A generalization is too allow non-integer weights, and sign-coupling is used in order to allow negative weights. This process can be described as a multiplication between the sign of the weight $\mathrm{sign}(w_i)$ and the corresponding sample s_i. The output of this function is then represented as:

$$\mathrm{MEDIAN}(s_1,\, s_2, \cdots, s_n; w_1,\, w_2, \cdots, w_n) \;=\; \arg\min_{\beta} \sum_{i=1}^{n} |w_i| \cdot |\mathrm{sign}(w_i) \cdot s_i - \beta|$$

6.1.4 Myriad filter

matlab function: `myriadfilter(s,w,tuningparam)`

Computes the myriad of a set of N samples $S = s_i$ with weights $W = w_i$. The myriad filter is a stable filter for the Cauchy case: $\alpha = 1$, $\beta = 0$, $\gamma =$`tuningparam`, `param`=0. See Arce (2005) for more information.

6.1.5 Selection filter

matlab function: `selectionfilter(s,w,filterinfo)`

Computes the so-called selection M-filter. The selection filter outputs the input sample with the lowest cost function. The following parameters of the structure `filterinfo` are expected: `alpha`, `beta`, `gamma`, `param`, `rhofn`, and `costfn`. If `costfn` is not 'COST_UNWEIGHTED', then the weights must be specified.

6.1.6 Stable filter

matlab function: `stablefilter(s,w,filterinfo)`

Computes the stable filter of a set of `nwin` samples s_i with weights w_i. If mode is BASIC, the following parameters of the structure `filterinfo` are expected: `costfn`, `alpha`, `beta`, `gamma`, and `param`. If `costfn` is not 'COST_UNWEIGHTED', then the weights must be specified. `searchmethod` is set to 'BRANCHANDBOUND_LOC_BND', and other values are set to default values.

If mode is ADVANCED, the following parameters of the structure `filterinfo` are expected: `costfn`, `rhofn`, `alpha`, `beta`, `gamma`, `param`, and `searchmethod`. If `costfn` is not 'COST_UNWEIGHTED', then the weights must be specified. Depending on the value of `searchmethod`, the following fields are required in `filterinfo`:

- If `searchmethod`='FIXEDPOINTSEARCH_I', then set `iternum`. This method uses a fixed point search routine initialized at the output of the selection filter (the most likely data point).
- If `searchmethod`='FIXEDPOINTSEARCH_II', then set `iternum`. This method uses a fixed point algorithm with multiple initialization points. Precisely, the fixed point search is run starting at each sample value.
- If `searchmethod`='BRANCHANDBOUND_LIP', then set `xtol`, `queuesize`, `lipconst`. This method uses a branch and bound minimization routine based on Lipschitz bound for the function $\rho(x)$.
- If `searchmethod`='BRANCHANDBOUND_LOC_BND', then set `xtol`, `x0` and `queuesize`. This method uses a branch and bound minimization routine based on a local bound.

6.2 Cost functions

6.2.1 Cost function vectorized

matlab function: `stablecostfn(x,s,w,rhofn,costfn,theta,param)`

Computes the general cost function for a vector of `x` values. `s` and `w` are as in the cost function formula. `costfn` determines which one of equations (10), (12), (13), and (14) is used, and `param` is the parameterization. `rhofn` determines how $\rho(x)$ is evaluated, `theta` = $[\alpha, \beta, \gamma, \delta]$ are the stable parameters.

6.2.2 Cost function at a single point

matlab function: `not implemented in matlab`

Computes the general cost function at a single x value. Used internally.

7 Error/return codes

An error is unrecoverable and stops execution. For example, if you ask to compute the density of a stable parameter with $\alpha = 3$, you will get a return code of 1 and your function will stop. In contrast, a warning is informational and is usually not serious. It alerts you to the fact that the results of a calculation may have some inaccuracy. For example, stable densities have radical changes of the tail behavior when $\alpha = 2$ or $\beta = \pm 1$, and the computations have small inaccuracies in them. In practical terms this usually means little, as the difference between an $\alpha = 1.99$ stable distribution and an $\alpha = 2$ stable distribution in an statistical problem is likely to be unobservable in practice.

In matlab, you can turn error and warning messages off with: `warning off all`. The warning messages can be enable again with the command: `warning on all`. See the section on the `warning` command in matlab help for more information.

Return codes for STABLE program are given in the tables below. Univariate routines return error codes in the range 1-99, multivariate routines return error codes in the range 100-199.

code	type	meaning
0		No error
1	error	Invalid input parameter
2	error	alpha parameter outside of tabulated values in QKSTABLE
3	error	Too many data points for internal array
4	error	Error computing the likelihood, e.g. pdf=0
5	warning	Possible approx. error while using QKSTABLE for alpha or beta near boundary
6	warning	Possible error in confidence intervals because parameter is near boundary
7	warning	alpha and/or beta rounded to a special value, adjust tol(4)
8	warning	alpha is at lower bound for search, may not have found best value for alpha
9	error	Too many bins (distinct possible values) in sdiscretemle
10	error	beta must be 0 to use this function
11	error	beta near +1 or -1 does not work in this function
12	error	sinc error in sfitfracmoment
13	error	Internal error in sfitlogabs
14	error	Data value near zero in sfitfracmoment or sfitlogabs
15	error	Error in subroutine
16	error	Internal error while computing derivatives
17	error	f(a) and f(b) have the same signs
18	error	Too many function evaluations
19	error	Not enough memory
20	error	X zero value
21	error	Internal error in quickstable
30	error	Two parameterization is required in skewed case

Table 1: Univariate error codes

Warning code 7 can arise in several ways. The purpose of this warning is to avoid numerical problems in internal calculations that can occur near the boundary in the parameter space or to use special cases to increase speed, but to let the user know that something nonstandard is being done. In the following discussion, let $\epsilon =$ the value of tolerance(4). The default value is $\epsilon = 0.01$. (You can change the value of tolerance(4) by using the function stablesettolerance above, and query it's value by using function stablegettolerance. The default value was picked in an ad hoc way; you can make it smaller, even 0, if you wish to calculate certain quantities in one of the cases below. But be aware that numerical errors may arise.) Special cases where warning code 7 occur are:

1. α near 2: if $\alpha \in (2 - \epsilon, 2)$, then α is set to 2 and β is set to 0.
2. Near $\alpha = 1$ but not Cauchy: if $|\alpha - 1| < \epsilon$ and $|\beta| \geq \epsilon$, then α is set to 1 and β is left unchanged. This is to avoid computations involving $\beta \tan(\pi\alpha/2)$, which blows up as $\alpha \to 1$ if $\beta \neq 0$.
3. Near Cauchy case: if $|\alpha - 1| < \epsilon$ and $|\beta| < \epsilon$, then α is set to 1 and β is set to 0.
4. Near Lévy case: if $|\alpha - 1/2| < \epsilon$ and $|\beta - 1| < \epsilon$, then α is set to 1/2 and β is set to 1; if $|\alpha - 1/2| < \epsilon$ and $|\beta + 1| < \epsilon$, then α is set to 1/2 and β is set to -1.

code	type	meaning
101	error	Invalid input parameter
102	warning	Accuracy warning, alpha < 1
103	warning	vmax exceeded in mvstablepdf
104	error	Too many points in spectral measure
105	error	nspectral must be divisible by 4
106	error	This parameterization is not allowed in this function
107	error	Too few uniform(0,1) input values for simulation
108	error	Distribution not defined
109	error	mvstablecdf not implemented for nonsymmetric case
110	error	Matrix is not positive definite
111	error	alpha must be at least 0.8
112	error	Definition error
113	error	Dimension is greater than the max allowed
115	error	Spline error
150	error	Not enough memory
151	error	Error in a subroutine

Table 2: Multivariate error codes

The signal filtering module returns error codes in the range 1000-2000.

code	type	meaning
1100	error	Too few input parameters
1101	error	Too many input parameters
1110	error	Undefined padding
1111	error	Undefined filter
1112	error	Undefined rho function
1113	error	Undefined minimization method
1114	error	Full queue
1116	error	Skewed data needs parameterization 2
1117	error	Undefined cost function
1118	error	Negative weight
1119	error	Buffer too small
1120	error	Dimension error
1121	error	Undefined initialization method
1130	error	Invalid input argument
1140	error	Method parameter undefined
1150	error	Insufficient memory
1160	error	Memory violation
1170	error	Error in subroutine
1180	error	SAR model not defined
1999	error	Other kind of error

Table 3: Signal filtering error codes

References

Abdul-Hamid, H. and J. P. Nolan (1998). Multivariate stable densities as functions of one dimensional projections. *J. Multivar. Anal. 67*, 80–89.

Arce, G. R. (2005). *Nonlinear Signal Processing*. Wiley, NY.

Astola, J. and P. Kuosmanen (1997). *Fundamentals of Nonlinear Digital Filtering*. ARC Press.

Byczkowski, T., J. P. Nolan, and B. Rajput (1993). Approximation of multidimensional stable densities. *J. Multivar. Anal. 46*, 13–31.

Chambers, J., C. Mallows, and B. Stuck (1976). A method for simulating stable random variables. *Journal of the American Statistical Association 71(354)*, 340–344.

Fan, Z. (2006). Parameter estimation of stable distributions. *Communications in Statistics. Theory and Methods 35*, 245–256.

Kogon, S. and D. Williams (1998). *Characteristic function based estimation of stable parameters*. In R. Adler, R. Feldman, and M. Taqqu (Eds.), A Practical Guide to Heavy Tailed Data pp. 311-338. Boston, MA: Birkhäuser.

McCulloch, J. H. (1986). Simple consistent estimators of stable distribution parameters. *Communications in Statistics. Simulation and Computation 15*, 1109–1136.

Nikias, C. L. and M. Shao (1995). *Signal Processing with Alpha-Stable Distributions and Applications*. New York: Wiley.

Nolan, J. P. (1997). Numerical calculation of stable densities and distribution functions. *Commun. Statist. -Stochastic Models 13*, 759–774.

Nolan, J. P. (2001). Maximum likelihood estimation of stable parameters. In O. E. Barndorff-Nielsen, T. Mikosch, and S. I. Resnick (Eds.), *Lévy Processes: Theory and Applications*. Boston: Birkhäuser.

Nolan, J. P. (2008, July). Advances in nonlinear signal processing for heavy tailed noise. Intl. Workshop in Applied Probability 2008.

Nolan, J. P. (2010). *Stable Distributions - Models for Heavy Tailed Data*. Boston: Birkhäuser. In progress, Chapter 1 online at academic2.american.edu/∼jpnolan.

Nolan, J. P. and D. Ojeda (2006). Linear regression with general stable errors. Preprint.

Nolan, J. P., A. Panorska, and J. H. McCulloch (2001). Estimation of stable spectral measures. *Mathematical and Computer Modelling 34*, 1113–1122.

Nolan, J. P. and A. K. Panorska (1997). Data analysis for heavy tailed multivariate samples. *Comm. in Stat. - Stochastic Models 13*, 687–702.

Nolan, J. P. and B. Rajput (1995). Calculation of multidimensional stable densities. *Commun. Statist. - Simula. 24*, 551–556.

Nunez, R. C., J. G. Gonzalez, G. R. Arce, and J. P. Nolan (2008). Fast and accurate computation of the myriad filter via branch-and-bound search. *IEEE Transactions on Signal Processing 56*, 3340–3346.

Pitas, I. and A. Venetsanopoulos (1990). *Nonlinear Digital Filters. Principles and Applications*. Kluwer Academic Publishers.

Index