Multivariate Subgaussian Stable Distributions in R

by Bruce J. Swihart, John P. Nolan

Abstract We introduce and showcase **mvpd** (an acronym for *m*ulti*v*ariate *p*roduct *d*istributions), a package that uses a product distribution approach to calculating multivariate subgaussian stable distribution functions. The family of multivariate subgaussian stable distributions are eliptically contoured multivariate stable distributions that contain the multivariate Cauchy and the multivariate normal distribution.

Introduction

Multivariate subgaussian stable distributions are the elliptically contoured subclass of general multivariate stable distributions. To begin a brief introduction to multivariate subgaussian stable distributions we start with univariate stable distributions which may be more readily familiar and accessible. Univariate stable distributions are a flexible family and have four parameters, α, β, γ, and δ, and at least eleven parameterizations (!) which has led to much confusion (Nolan, 2020). Here we focus on the 1-parameterization of the Nolan style. Location is controlled by δ, scale by $\gamma \in (0, \infty)$, while $\alpha \in (0, 2]$ and $\beta \in [-1, 1]$ can be considered shape parameters. Being a location-scale family, a "standard" stable distribution will be when $\gamma = 1$ and $\delta = 0$. A solid introduction to univariate stable distributions can be found in the recent textbook *Univariate Stable Distributions* (Nolan, 2020) and its freely available Chapter 1 online (`https://edspace.american.edu/jpnolan/stable/`).

Univariate symmetric stable distributions are achieved by setting the skew parameter $\beta = 0$, which gives symmetric distributions that are bell-shaped like the normal distribution. A way to remember that these are called subgaussian is to see that as $\alpha \in (0, 2]$ increases from 0 it looks more and more normal until it **is** normal for $\alpha = 2$ (Figure 1). The *sub* in subgaussian refers to the tail behavior in that the rate of decrease in the tails is *less* than that of a gaussian – note how the tails are on top of the gaussian for $\alpha < 2$ in Figure 1. Equivalently, as α decreases, the tails get heavier. A notable value of α for subgaussian distributions is $\alpha = 1$ which is the Cauchy distribution. The Cauchy and Gaussian distribution are most well-known perhaps because they have closed-form densities, which all other univariate symmetric stable distributions lack.

Therefore, numerically computing the densities is especially important for application. For univariate stable distributions, there is open-source software to compute modes, densities, distributions, quantiles and random variates, including a number of R packages (**stabledist**, **stable**, **libstableR**, for example – see CRAN Task View: Probability *Distributions* for more).

As generalizations of the univariate stable, multivariate stable distributions are a very broad family encompassing many complicated distributions. A subclass of this family is the multivariate subgaussian stable distributions. Multivariate subgaussian stable distributions are symmetric and elliptically contoured. Similar to the aforementioned univariate symmetric stable distributions, the value $\alpha = 2$ is the multivariate gaussian and $\alpha = 1$ is the multivariate Cauchy. Being that they are elliptically contoured and symmetric makes them applicable to finance where joint returns have an (approximately) elliptical joint distribution (Nolan, 2020).

For multivariate subgaussian stable distributions, the parameter α is a scalar as in the univariate family, while $\boldsymbol{\delta}$ (location) becomes a d-dimensional vector and the analog for the scale parameter is a $d \times d$ shape matrix Q. The shape matrix Q needs to be semi-positive definite and is neither a covariance matrix nor covariation matrix. An introduction to multivariate subgaussian stable distributions can be found in Nolan (2013).

Aside from **mvpd**, the focus of this paper, if one wanted `R` functions to interact with multivariate subgaussian stable distributions they have two `R` package options:

- **alphastable** provides random univariate and mutlivariate generation, density calculation, and parameter fitting (albeit for modest sample sizes) via an EM alogrithm method (Teimouri et al., 2018, 2019).
- **stable** provides support for all stable univariate distributions and multivariate subgaussian stable distributions. Other cases are handled, to varying degrees, such as isotropic, independent, and spectral measure. For the purposes of this paper, we will note that the **stable** package provides random variate generation, density calculation, parameter fitting, distribution calculations via Monte Carlo methods for multivariate subgaussian stable distributions $2 \leq d \leq 100$. The **stable** package is developed by Robust Analysis and is available for purchase or through a

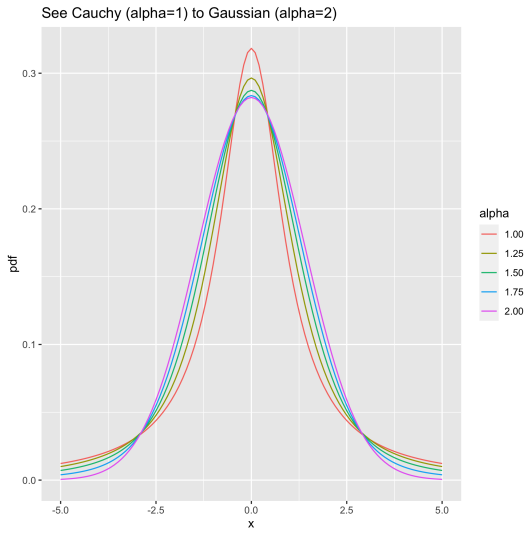

Figure 1: Standard univariate subgaussian distributions for various values of alpha. Cauchy is alpha=1 and normal is alpha=2.

Functionality	`alphastable`	`stable`	`mvpd`
random variates	✓	✓	✓
parameter estimation	✓	✓	✓
density	✓	✓	✓
cumulative distribution (monte carlo)	x	✓	✓
cumulative distribution (integrated)	x	x	✓
multivariate subgaussian stable	✓	✓	✓
multivariate independent stable	x	✓	x
multivariate isotropic stable	x	✓	x
multivariate discrete-spectral-measure stable	x	✓	x

Table 1: Summary of functionality among R packages. Note: The `stable` package referenced is not the one on CRAN.

software grant at http://www.robustanalysis.com/. It is distinct from the univariate **stable** package on CRAN authored by Jim Lindsey.

mvpd provides random variate generation, density calculation, parameter fitting, distribution function calculations via Monte Carlo methods, as well as an integrated method for distribution calculations that allows tolerance specification. A comparison of the packages can be found in Table 1.

While the lack of a tractable density and distribution function impedes directly calculating multivariate subgaussian stable distributions, it is possible to represent them in terms of a product distribution for which each of the two distributions in the product has known numerical methods developed and deployed (in R packages on CRAN). This paper utilizes this approach. The next section covers some product distribution theory.

Product distribution theory

This section reviews the known results of *product distributions* and describes our notation. Allow univariate random variable A with density $f_A(x)$ and d-dimensional random variable G to have density $f_G(\boldsymbol{x})$ and distribution function $F_G(\boldsymbol{v}, \boldsymbol{w}) = P(\boldsymbol{v} < \boldsymbol{x} \leq \boldsymbol{w})$. Consider the $d-$dimensional product $H = A^{1/2}G$. From standard product distribution theory we know the f_H is represented by 1-dimensional integral:

$$f_H(\boldsymbol{x}) \quad = \quad \int_0^\infty f_B(u) f_G(\boldsymbol{x}/u) \frac{1}{|u|^d} du \tag{1}$$

where $B = A^{1/2}$ so that $f_B(x) \coloneqq 2x f_A(x^2)$. Consequently the distribution function (with lower bound $\boldsymbol{v}$ and upper bound $\boldsymbol{w}$) of the r.v. H is represented by

$$\begin{aligned} F_{\mathrm{H}}(\boldsymbol{v},\boldsymbol{w}) &= \int_0^\infty f_B(u) \int_{v_1}^{w_1} \cdots \int_{v_d}^{w_d} f_G(\boldsymbol{t}/u) \frac{1}{|u|^d} dt_1 \ldots dt_d du \qquad (2) \\ &= \int_0^\infty f_B(u) \int_{v_1/u}^{w_1/u} \cdots \int_{v_d/u}^{w_d/u} f_G(\boldsymbol{t}) dt_1 \ldots dt_d du \\ &= \int_0^\infty f_B(u) F_G(\boldsymbol{v}/u, \boldsymbol{w}/u) du \qquad (3) \end{aligned}$$

Take note of the representation in (1) and (3). From a practical standpoint, if we have a (numerical) way of calculating f_A, f_G, and F_G we can calculate f_H and F_H.

Multivariate elliptically contoured stable distributions

From Nolan (2013), H is a d-dimensional subgaussian stable distribution if A is a univariate stable distribution

$$A \sim S\left(\frac{\alpha}{2}, 1, 2\cos\left(\frac{\pi\alpha}{4}\right)^{\left(\frac{2}{\alpha}\right)}, 0; 1\right)$$

and G is a d-dimensional multivariate normal $G \sim MVN(0, Q)$, where the shape matrix Q is positive semi-definite. This corresponds to Example 17 in Hamdan (2000).

Implementation

Using the aforementioned theory of product distributions, we can arrive at functions for a multivariate subgaussian stable density and distribution function thanks to established functions for univariate stable and multivariate normal distributions. A key package in the implementation of multivariate subgaussians in R is **mvtnorm** (Genz et al., 2020; Genz and Bretz, 2009). In the basic product-distribution approach of **mvpd**, f_G and F_G are `mvtnorm::dmvnorm` and `mvtnorm::pmvnorm` respectively. Allow the density of A, f_A (to be numerically calculated in `R`) using `stable::dstable` or `libstableR::stable_pdf`. Presented as pseudo-code:

- $f_A(x, \alpha) \coloneqq$ `libstableR::stable_pdf`$(x,$ `pars=` $\left(\frac{\alpha}{2}, 1, 2\cos\{\frac{\pi\alpha}{4}\}^{\left(\frac{2}{\alpha}\right)}, 0\right)$ `; pm=1`$)$
- $f_B(x, \alpha) \coloneqq 2x f_A(x^2, \alpha)$
- $f_H(\boldsymbol{x}, \alpha, Q) = \int_0^\infty f_B(u; \alpha) \times$ `mvtnorm::dmvnorm`$(\boldsymbol{x}/u, \text{sigma} = Q) \frac{1}{|u|^d} du$
- $F_H(\boldsymbol{v}, \boldsymbol{w}, \alpha, Q) = \int_0^\infty f_B(u; \alpha) \times$ `mvtnorm::pmvnorm`$(\text{lower} = \boldsymbol{v}/u, \text{upper} = \boldsymbol{w}/u, \text{sigma} = Q) du$

The (outermost) univariate integral is numerically evaluated with `stats::integrate`.

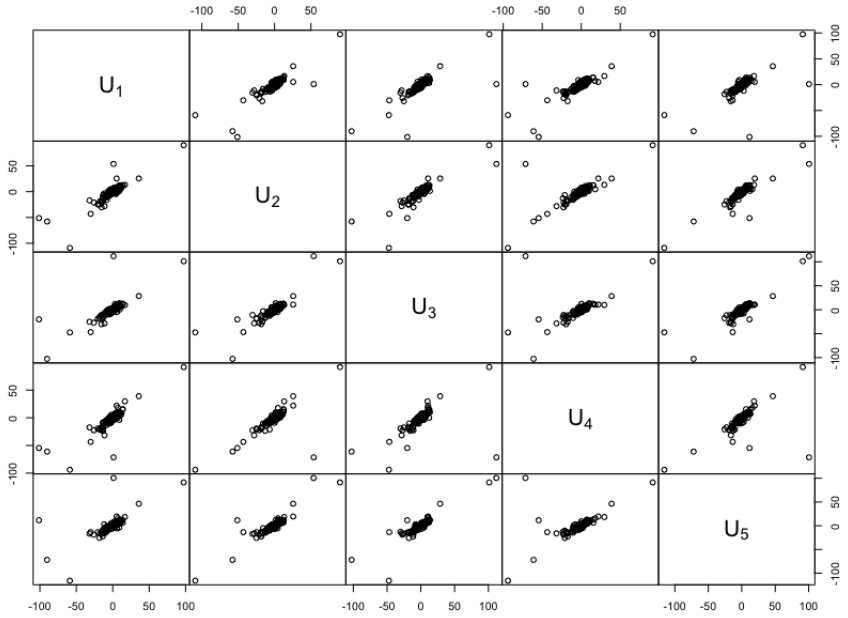

Figure 2: A lattice of scatterplots of the 5-dimensional subgaussian distribution.

Quick start

We like the notion of a quick start to outline the `[dpr]mvss` (*m*ulti*v*ariate *s*ubgaussian *s*table) functions, and walk through the code in the subsequent sections. As an overview, we generate 5000 5-dimensional subgaussian variates with $\alpha = 1.7$ and an "exchangeable" shape matrix using `rmvss`. We then recover the parameters with an illustrative call to `fit_mvss`. We can calculate the density (`dmvss`) at the center of the distribution and get a quick estimate of the distribution between -2 and 2 for each member of the 5-dimensional variate using `pmvss_mc`. We investigate a refinement of that quick distribution estimate using `pmvss`.

Random variates generation with `rmvss`

We'll generate 5000 5-dimensional subgaussian random variates with a specified α and shape matrix. They are pictured in Figure 2. In the next section we will fit a distribution to these.

```
R> library(mvpd)
R> set.seed(10)
R> shape_matrix <- structure(c(1, 0.9, 0.9, 0.9, 0.9,
                               0.9, 1, 0.9, 0.9, 0.9,
                               0.9, 0.9, 1, 0.9, 0.9,
                               0.9, 0.9, 0.9, 1, 0.9,
                               0.9, 0.9, 0.9, 0.9, 1),
                             .Dim = c(5L, 5L))
R> X <- mvpd::rmvss(n=5000, alpha=1.7, Q=shape_matrix)
R> copula::pairs2(X)
```

Fitting a multivariate subgaussian distribution with `fit_mvss`

If you have data in a $n \times d$ matrix $\mathbf{X}$ and want to fit a d-dimensional multivariate subgaussian distribution to those data, then `fit_mvss` will return estimates of the parameters using the method outlined in Nolan (2013). The method involves fitting univariate stable distributions for each column and assessing the resulting α, β and δ parameters. The column-wise α estimates should be similar and the column-wise β estimates close to 0. This column-wise univariate fitting is carried out by `libstableR::stable_fit_mle2d(W,parametrization = 1L)` and the off diagonal elements can be found due to the properties of univariate stable distributions (see Nolan (2013)). For your convenience,

the univariate column-wise estimates of α, β, γ and δ are returned in addition to the raw estimate of shape matrix and the result of `Matrix::nearPD` applied to the raw estimate of the shape matrix.

```
R> fitmv <- mvpd::fit_mvss(X)
R> fitmv
$univ_alphas
[1] 1.698617 1.708810 1.701662 1.696447 1.699372

$univ_betas
[1] -0.02864287 -0.04217262 -0.08444540 -0.06569907 -0.03228573

$univ_gammas
[1] 1.016724 1.000151 1.008055 1.012017 1.002993

$univ_deltas
[1] -0.03150732 -0.06525291 -0.06528644 -0.07730645 -0.04539796

$mult_alpha
[1] 1.700981

$mult_Q_raw
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.0337276 0.9034599 0.8909654 0.8937814 0.8647089
[2,] 0.9034599 1.0003026 0.9394846 0.9072368 0.8535091
[3,] 0.8909654 0.9394846 1.0161748 0.8929937 0.9037467
[4,] 0.8937814 0.9072368 0.8929937 1.0241777 0.9281714
[5,] 0.8647089 0.8535091 0.9037467 0.9281714 1.0059955

$mult_Q_posdef
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.0337276 0.9034599 0.8909654 0.8937814 0.8647089
[2,] 0.9034599 1.0003026 0.9394846 0.9072368 0.8535091
[3,] 0.8909654 0.9394846 1.0161748 0.8929937 0.9037467
[4,] 0.8937814 0.9072368 0.8929937 1.0241777 0.9281714
[5,] 0.8647089 0.8535091 0.9037467 0.9281714 1.0059955
```

Once the distribution has been fitted, `fitmv$mult_alpha`, `fitmv$mult_Q_posdef`, and `fitmv$univ_deltas`, can be used as the `alpha`, `Q`, and `delta` arguments, respectively, in calls to `dmvss` to calculate densities and `pmvss_mc` or `pmvss` to calculate probabilities. They could also be passed to `rmvss` to generate random variates for simulations.

Density calculations with `dmvss`

We can calculate the density at the center of the distribution.

```
R> mvpd::dmvss(x=fitmv$univ_deltas,
+              alpha=fitmv$mult_alpha,
+              Q=fitmv$mult_Q_posdef,
+              delta=fitmv$univ_deltas)[1]
$value
[1] 0.1278952
```

Distribution calculation by Monte Carlo with `pmvss_mc`

The method of calculating the distribution by Monte Carlo relies on the ability to produce random variates quickly and then calculate what proportion of them them fall within the specified bounds. To generate multivariate subgaussian stable variates, a scalar A is drawn from

$$\texttt{libstableR::stable\_rnd}(n, \texttt{pars=}\left(\frac{\alpha}{2}, 1, 2\cos\{\frac{\pi\alpha}{4}\}^{\left(\frac{2}{\alpha}\right)}, 0\right)\texttt{;pm=1})$$

and then the square-root of A multiplied by a draw G from

$$\texttt{mvtnorm::rmvnorm}(n, \text{sigma} = Q).$$

This allows for quick calculations but to increase precision requires generating larger number of random variates. For instance, if we wanted the distribution between -2 and 2 for each dimension, we could generate 10,000 random variates and then see how many of them fall between the bounds. It looks like 6,820 variates were within the bounds:

```
R> mvpd::pmvss_mc(lower=rep(-2,5),
+                 upper=rep( 2,5),
+                 alpha=fitmv$mult_alpha,
+                 Q=fitmv$mult_Q_posdef,
+                 delta=fitmv$univ_deltas,
+                 n=10000)
[1] 0.6820
```

We run it again and the answer changes:

```
R> mvpd::pmvss_mc(lower=rep(-2,5),
+                 upper=rep( 2,5),
+                 alpha=fitmv$mult_alpha,
+                 Q=fitmv$mult_Q_posdef,
+                 delta=fitmv$univ_deltas,
+                 n=10000)
[1] 0.6742
```

With the Monte Carlo method, no error is calculated. One can calculate a standard confidence interval for binomial proportions to get a sense of the uncertainty, but this is not the same as specifying precision. The next section introduces how to use the integrated distribution function F_H from product theory and specify precision.

Distribution function calculation via integration with `pmvss`

There are three inexact entities involved in the distribution calculation F_H as found in `pmvss`: the numerically calculated F_G, the numerically calculated f_A, and the outer numerical integration.

The outer integral by `integrate` assumes the integrand is calculated without error, but this is not the case. See the supplementary materials section "Thoughts on error propagation in `pmvss`" for justification and guidance for specifying the values of `abs.tol.si`, `abseps.pmvnorm`, and `maxpts.pmvnorm`. The first of these three arguments is passed to the `abs.tol` argument of `stats::integrate` and controls the absolute tolerance of the numerically evaluated outer 1-dimensional integral. The remaining two are passed to `maxpts` and `abseps` of `mvtnorm::GenzBretz` and control the accuracy of `mvtnorm::pmvnorm`.

Briefly, our experience suggests that to be able to treat `abs.tol.si` as the error of the result, `abseps.pmvnorm` should be 1e-2 times smaller than the specified `abs.tol.si` which may require a multiple of the default 25000 default of `maxpts.pmvnorm` – which will lead to more computational intensity and longer computation times as demonstrated below:

```
## abs.tol.si    abseps.pmnvorm   maxpts         Time
## 1e-01         1e-03            25000
## 1e-02         1e-04            25000*10        3 sec
## 1e-03         1e-05            25000*100      22 sec
## 1e-04         1e-06            25000*1000      4 min
## 1e-05         1e-07            25000*10000    26 min
## 1e-06         1e-08            25000*85000   258 min
```

With this in mind, the output from the Quick Start code is:

```
R> mvpd::pmvss(lower=rep(-2,5),
+              upper=rep( 2,5),
+              alpha=fitmv$mult_alpha,
+              Q=fitmv$mult_Q_posdef,
+              delta=fitmv$univ_deltas,
+              abseps.pmvnorm = 1e-4,
+              maxpts.pmvnorm = 25000*10,
+              abs.tol.si = 1e-2)[1]
$value
[1] 0.6768467
```

but with `abs.tol.si=1e-2` we view everything past the hundredths digit as discardable. So the answer is 0.67 and we trust that the answer is between 0.66 and 0.68.

Both `pmvss` and `pmvss_mc` take infinite limits. Since `pmvss_mc` calculates how many random variates H_i, $i \in \{1, \ldots, n\}$ are within the bounds, `pmvss` might be preferred to `pmvss_mc` when calculating the tails of the distribution, unless n is made massively large.

Speed and Accuracy Trials

We provide a sense of accuracy and computational time tradeoffs with a modest simulation experiment (Figure 3, see supplementary materials for code). Estimating these distributions is inherently difficult – difficult in the sense that expecting accuracy farther out than the 5th decimal place for distribution functions is unreasonable. Therefore, we will define our "gold standard" targets for accuracy evaluation as the numerical density produced by Robust Analysis' `dstable` integrated by `cubature::hcubature()` with tolerance `tol=1e-5`.

We will time three functions using `bench::mark()` in different scenarios. The first function is Robust Analysis' `pmvstable.MC()` (RAMC) and the other two are `mvpd::pmvss_mc()` (PDMC) and `mvpd::pmvss()` (PD). Fixing $\alpha = 1.7$ and dimension $d = 4$, the different test scenarios will involve a low level of pairwise association vs. a high level in a shape matrix of the form:

- $Q_{exch} = \begin{bmatrix} 1 & \rho & \rho & \rho \\ \rho & 1 & \rho & \rho \\ \rho & \rho & 1 & \rho \\ \rho & \rho & \rho & 1 \end{bmatrix}$ for $\rho = 0.1$ and $\rho = 0.9$.

We calculate the distributions in the hypercube bounded by (-2,2) in all four dimensions. The gold standard for the $\rho = 0.1$ case was 0.5148227 and 0.7075104 for the $\rho = 0.9$ case. The numerical integration of the former took 3 minutes whereas the latter took 1 hour – which portends that higher associations involve more computational difficulty. We back-calculated the number of samples needed to give the methods involving Monte Carlo (RAMC and PDMC) a 95% CI width that would fall within 0.001 and 0.0001 of the gold standard, and display the scatter plots of estimate and computational time in (Figure 3).

From Figure 3, some high-level conclusions can be drawn: higher pairwise associations require more computational resources and time, increasing the precision requires more computational resources and time, and sometimes the Monte Carlo methods are faster than PD, sometimes not. PD seems to be quite precise and possibly underestimating the gold standard.

Bonus: faster distribution calculations via a modified `QRVSN` algorithm

Insight: multivariate student's t distributions are a product distribution

The derivation of the univariate student's t distribution is commonly motivated with a ratio of two quantitites each involving random variables: a standard normal Z in the numerator and a $V \sim \chi^2(\nu)$ in the denominator:

$$T_\nu = \frac{Z}{\sqrt{V/\nu}} = Z\sqrt{\frac{\nu}{V}}$$

but what often is left out of the instruction is that $A_\nu = \frac{\nu}{V} \sim IG\left(\frac{\nu}{2}, \frac{\nu}{2}\right)$ has an inverse-gamma distribution, where $X \sim IG(r, s)$ with rate r, shape s, and density $f(x; r, s) = \frac{r^s}{\Gamma(s)} x^{(-s-1)} e^{-r/x}$. This implies we equivalently have a product distribution of the type $T_\nu = A_\nu^{1/2} Z$. This notion holds for the multivariate case as well, where for $G \sim MVN(0, Q)$ as before and A_ν is an inverse-gamma with $r = s = \nu/2$ then $H_\nu = A_\nu^{1/2} G$ is a d-dimensional student's t distribution with ν degrees of freedom and covariance matrix Q. This corresponds to Example 16 in in Hamdan (2000), and is equivalent to the 'chi-normal' (χ-Φ) formulation in Genz and Bretz (2002) (earning the namesake 'χ' due to the fact that $\sqrt{V} \sim \chi(\nu)$). For $d \geq 4$, the QRVSN algorithm is used in `mvtnorm::pmvt`. The reordering and rotational methodology that makes `pmvtnorm::pmvt` so fast is independent of the part that generates $\sqrt{1/A_v}$ random variates. This means that if one replaced $\sqrt{1/A_v}$ with $\sqrt{1/A}$ variates, `mvtnorm::pmvt` would produce *not* multivariate student's t distributions but **multivariate subgaussian stable distributions**. We implement a modified QRSVN algorithm for multivariate subgaussian stable distributions in a separate package, **mvgb** in honor of Genz and Bretz.

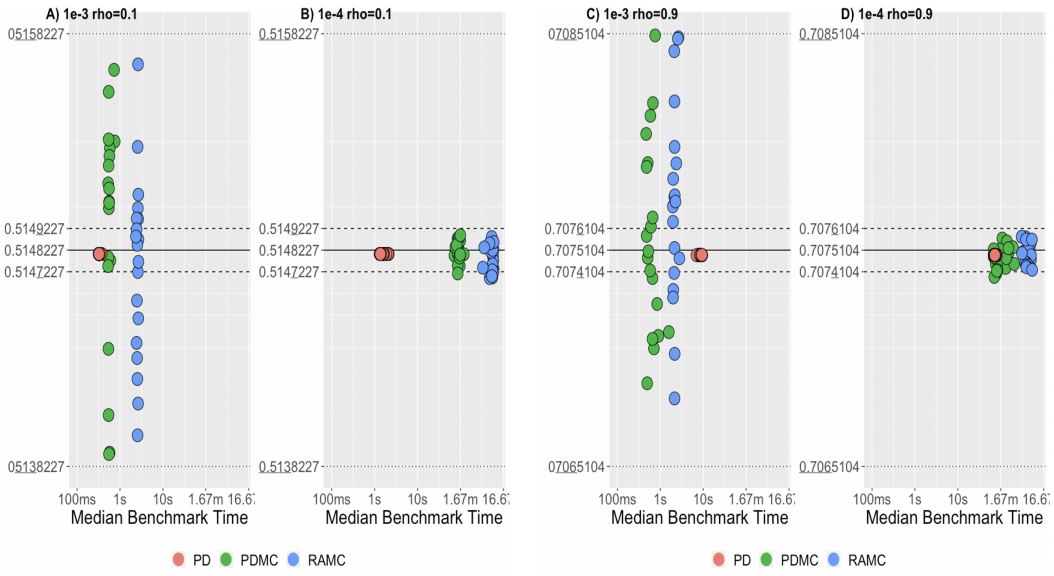

Figure 3: Speed and accuracy of distribution calculations. Consider a multivariate subgaussian stable distribution of $d = 4$, $\alpha = 1.7$, with limits of integration being (-2,2) for each dimension. In panels A) and B) we have an exchangeable shape matrix with $\rho = 0.1$, and specified precision of 1e-3 and 1e-4 for `pmvss` (PD), respectively. Analogously, panels C) and D) display results for an exchangeable shape matrix with $\rho = 0.9$. Concurrently in each panel, are the results for Robust Analysis' `pmvstable.MC` (RAMC) and `pmvss_mc` (PDMC) with enough simulated variates to produce a 95 CI width that matches the precision. Each point is an independent call and the calculated distribution is on the Y-axis vs the median benchmark time on the X-axis. There are 20 calls per function per scenario. The dotted line is the 1e-3 boundary of the gold standard and the dashed line is the 1e-4 boundary.

Implementation of `mvgb::pmvss`

Generating random variates of A requires two independent uniform random variates, and only one of which is Quasi-Random in our implmentation. Regardless, this modified QRSVN approach enables the potential advantage of the rotation of the distribution and the reordering of integration limits. The takeaway is, that for similar precisions, `mvgb::pmvss` may be much faster than `mvpd::pmvss`, such as 10 seconds vs 500 seconds for 4 digits of precision in the following example:

```
R> set.seed(321)
R> library(mvgb)
R> tictoc::tic() ## mvgb takes about 10 secs.
R> gb_4digits <-
+   mvgb::pmvss(lower=rep(-2,5),
+               upper=rep( 2,5),
+               alpha=fitmv$mult_alpha,
+               Q=fitmv$mult_Q_posdef,
+               delta=fitmv$univ_deltas,
+               abseps = 1e-4,
+               maxpts = 25000*350)
R> tictoc::toc()
9.508 sec elapsed
> gb_4digits
[1] 0.6768
R> tictoc::tic() ## mvpd takes about 10 MINUTES.
R> pd_4digits <-
+   mvpd::pmvss(lower=rep(-2,5),
+               upper=rep( 2,5),
+               alpha=fitmv$mult_alpha,
+               Q=fitmv$mult_Q_posdef,
+               delta=fitmv$univ_deltas,
+               abseps.pmvnorm = 1e-6,
+               maxpts.pmvnorm = 25000*1000,
+               abs.tol.si = 1e-4)
```

```
R> tictoc::toc()
518.84 sec elapsed
R> pd_4digits[1]
[1] 0.6768
```

Although currently on CRAN, we include `mvgb::pmvss` here as a proof-of-concept and as an area of future work. More research is needed into its computational features and accuracy, and this is only encouraged by promising preliminary results.

Acknowledgements

This work utilized the computational resources of the NIH HPC Biowulf cluster (http://hpc.nih.gov). We thank Robust Analysis for providing their **stable** `R` package via a software grant.

Bibliography

A. Genz and F. Bretz. Comparison of methods for the computation of multivariate t probabilities. *Journal of Computational and Graphical Statistics*, 11(4):950–971, 2002. [p7]

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag, Heidelberg, 2009. ISBN 978-3-642-01688-2. [p3]

A. Genz, F. Bretz, T. Miwa, X. Mi, F. Leisch, F. Scheipl, and T. Hothorn. *mvtnorm: Multivariate Normal and t Distributions*. The organization, 2020. URL https://CRAN.R-project.org/package=mvtnorm. R package version 1.1-0. [p3]

H. N. Hamdan. *PhD Thesis: Characterizing and approximating scale mixtures*. American University, 2000. [p3, 7]

J. P. Nolan. Multivariate elliptically contoured stable distributions: theory and estimation. *Computational Statistics*, 28(5):2067–2089, 2013. [p1, 3, 4]

J. P. Nolan. *Univariate stable distributions*. Springer, 2020. [p1]

M. Teimouri, S. Rezakhah, and A. Mohammadpour. Parameter estimation using the em algorithm for symmetric stable random variables and sub-gaussian random vectors. *Journal of Statistical Theory and Applications*, 17(3):439–461, 2018. [p1]

M. Teimouri, A. Mohammadpour, and S. Nadarajah. *alphastable: Inference for Stable Distribution*, 2019. URL https://CRAN.R-project.org/package=alphastable. R package version 0.2.1. [p1]

Bruce J. Swihart
National Institutes of Health
National Institute of Allergy and Infectious Diseases
5601 Fishers Lane Baltimore MD, 20852
United States of America
(ORCiD 0000-0002-4216-9942)
bruce.swihart@nih.gov

John P. Nolan
American University
Department of Math & Statistics
4400 Mass. Ave, Washington DC 20016
United States of America
(ORCiD 0000-0002-9669-382X)
jpnolan@american.edu