It is often the case that you can arrange for the index set to come out in the right order, and so remove the need for a call to `aperm`.

When the uncertain objects are more than two-dimensional then tensors are usually the only option, both for writing down the required calculation, and performing it. You can find an example in the Appendix of our forthcoming paper (Craig *et al*, 2001) which is expressed entirely in tensor-form. In this paper we have a mixture of one-, two- and three-dimensional objects which are also very large. For example, lurking in the Appendix you can find expressions such as

$$\mathsf{Cov}\left[B_{im},\, S_{i''r}\right]\, P_{i''ri'''r'}\, H^{\epsilon}_{i'i'''r'}(x)\, g_m(x),$$

a two-dimensional expression (in ii') which is a function of the vector x. To give you some idea of the scale of this calculation, there are about 60 i's, 45 m's, and 6 r's, and x is 40-dimensional. (We then have to integrate x out of this, but that's another story!).

In our current development of this work we will need to evaluate the object $\mathsf{E}\left[B_{im}\, B_{i'm'}\, B_{i''m''}\, B_{i'''m'''}\right]$. This has about 5×10^{13} components and so this calculation is currently 'on hold', but I am hoping that Luke Tierney's eagerly anticipated *hyper-spatial* memory manager will sort this out and spare me from having to thinking too deeply about the essential structure of these objects.

Implementation

Conceptually, a tensor product is a simple thing. Take two arrays `A` and `B`, and identify the extents in each to be collapsed together (make sure they match!). Permute the collapse extents of `A` to the end and the collapse extents of `B` to the beginning. Then re-shape both `A` and `B` as matrices and take the matrix product. Reshape the product to have the non-collapse extents of `A` followed by the non-collapse extents of `B` and you are done.

In order for these operations to be efficient, we need a rapid `aperm` operation. The `aperm` function in R 1.2 was not really fast enough for large objects, and so I originally wrote **tensor** entirely in `C`, using a Python approach for arrays called 'strides'. In R 1.3 however, there is a new `aperm` that itself uses strides, to achieve a reduction in calculation time of about 80%. The new version of **tensor**, version 1.2, therefore does the obvious thing as outlined above, and does it entirely in R. But the old version of **tensor**, version 1.1-2, might still be useful for dealing with very large objects, as it avoids the duplication that can happen when passing function arguments.

References

P. S. Craig, M. Goldstein, J. C. Rougier and A. H. Seheult (2001). Bayesian forecasting for complex systems using computer simulators, *Journal of the American Statistical Association*, forthcoming.

P. McCullagh (1987). *Tensor Methods in Statistics*, Chapman and Hall.

Jonathan Rougier
University of Durham, UK
J.C.Rougier@durham.ac.uk

On Multivariate t and Gauß Probabilities in R

by Torsten Hothorn, Frank Bretz and Alan Genz

Introduction

The numerical computation of a multivariate normal or t probability is often a difficult problem. Recent developments resulted in algorithms for the fast computation of those probabilities for arbitrary correlation structures. We refer to the work described in (3), (4) and (2). The procedures proposed in those papers are implemented in package **mvtnorm**, available at CRAN. Basically, the package implements two functions: `pmvnorm` for the computation of multivariate normal probabilities and `pmvt` for the computation of multivariate t probabilities, in both cases for arbitrary means (resp. noncentrality parameters), correlation matrices and hyperrectangular integration regions.

We first illustrate the use of the package using a simple example of the multivariate normal distribution. A little more details are given in Section 'Details'. Finally, an application of `pmvt` in a multiple testing problem is discussed.

A simple example

Assume that $X = (X_1, X_2, X_3)$ is multivariate normal with correlation matrix

$$\mathbf{\Sigma} = \begin{pmatrix} 1 & \frac{3}{5} & \frac{1}{3} \\ \frac{3}{5} & 1 & \frac{11}{15} \\ \frac{1}{3} & \frac{11}{15} & 1 \end{pmatrix}$$

and expectation $\mu = (0, 0, 0)^\top$. We are interested in the probability

$$P(-\infty < X_1 \leq 1, -\infty < X_2 \leq 4, -\infty < X_3 \leq 2).$$

This is computed as follows:

```
R> m <- 3
R> sigma <- diag(3)
R> sigma[2,1] <- 3/5
R> sigma[3,1] <- 1/3
R> sigma[3,2] <- 11/15
R> pmvnorm(lower=rep(-Inf, m), upper=c(1,4,2),
           mean=rep(0, m), sigma)
$value
[1] 0.8279847

$error
[1] 5.684033e-07

$msg
[1] "Normal Completion"
```

The mean vector is passed to `pmvnorm` by the argument `mean`, and `sigma` is the correlation matrix (only the lower triangle being used) The region of integration is given by the vectors `lower` and `upper`, both can have elements `-Inf` or `+Inf`. The value of `pmvnorm` is a list with the following components:

`value`: the estimated integral value,

`error`: the estimated absolute error,

`msg`: a status message, indicating whether or not the algorithm terminated correctly.

From the results above it follows that $P(-\infty < X_1 \leq 1, -\infty < X_2 \leq 4, -\infty < X_3 \leq 2) \approx 0.82798$ with an absolute error estimate of 2.7×10^{-7}.

Details

This section outlines the basic ideas of the algorithms used. The multivariate t distribution (MVT) is given by

$$\mathbf{T}(\mathbf{a}, \mathbf{b}, \mathbf{\Sigma}, \nu) = \frac{2^{1-\frac{\nu}{2}}}{\Gamma(\frac{\nu}{2})} \int_0^\infty s^{\nu-1} e^{-\frac{s^2}{2}} \mathbf{\Phi}\left(\frac{s\mathbf{a}}{\sqrt{\nu}}, \frac{s\mathbf{b}}{\sqrt{\nu}}, \mathbf{\Sigma}\right) ds,$$

where the multivariate normal distribution function (MVN)

$$\mathbf{\Phi}(\mathbf{a}, \mathbf{b}, \mathbf{\Sigma}) = \frac{1}{\sqrt{|\mathbf{\Sigma}|(2\pi)^m}} \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_m}^{b_m} e^{-\frac{1}{2}\mathbf{x}^t \mathbf{\Sigma}^{-1} \mathbf{x}} d\mathbf{x},$$

$\mathbf{x} = (x_1, x_2, ..., x_m)^t$, $-\infty \leq a_i < b_i \leq \infty$ for all i, and $\mathbf{\Sigma}$ is a positive semi-definite symmetric $m \times m$ matrix. The original integral over an arbitrary m-dimensional, possibly unbounded hyper-rectangle is transformed to an integral over the unit hypercube. These transformations are described in (3) for the MVN case and in (2) for the MVT case. Several suitable standard integration routines can be applied to this transformed integral. For the present implementation randomized lattice rules were used. Such lattice rules seek to fill the integration region evenly in a deterministic process. In principle, they construct regular patterns, such that the projections of the integration points onto each axis produce an equidistant subdivision of the axis. Robust integration error bounds are obtained by introducing additional shifts of the entire set of integration nodes in random directions. Since this additional randomization step is only performed to introduce a robust Monte Carlo error bound, 10 simulation runs are usually sufficient. For a more detailed description (2) might be referred to.

Applications

The multivariate t distribution is applicable in a wide field of multiple testing problems. We will illustrate this using a example studied earlier by (1). For short, the effects of 5 different perfusates in capillary permeability in cats was investigated by (5). The data met the assumptions of a standard one-factor ANOVA. For experimental reasons, the investigators were interested in a simultaneous confidence intervals for the following pairwise comparisons: $\beta_1 - \beta_2, \beta_1 - \beta_3, \beta_1 - \beta_5, \beta_4 - \beta_2$ and $\beta_4 - \beta_3$. Therefore, the matrix of contrast is given by

$$\mathbf{C} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 \end{pmatrix}.$$

Reference (1) assumed that $\beta = (\beta_1, \ldots, \beta_5)$ is multivariate normal with mean β and covariance matrix $\sigma^2\mathbf{V}$, where $\mathbf{V}$ is known. Under the null hypothesis $\beta = 0$, we need knowledge about the distribution of the statistic

$$W = \max_{1 \leq j \leq 5} \left\{ \frac{|c_j(\hat{\beta} - \beta)|}{\hat{\sigma}\sqrt{c_j \mathbf{V} c_j^\top}} \right\}$$

where c_j is the jth row of $\mathbf{C}$. By assumption, $\hat{\sigma}$ is χ_ν distributed, so under hypothesis W the argument to max follows a multivariate t distribution. Confidence intervals can be obtained by $c_j\hat{\beta} \pm w_\alpha \hat{\sigma}\sqrt{c_j \mathbf{V} c_i^\top}$, where w_α is the $1 - \alpha$ quantile of the null distribution of W. Using `pmvt`, one can easily compute the quantile for the example cited above.

```
R> n <- c(26, 24, 20, 33, 32)
R> V <- diag(1/n)
R> df <- 130
R> C <- c(1,1,1,0,0,-1,0,0,1,0,0,-1,0,0,
```

```
          1,0,0,0,-1,-1,0,0,-1,0,0)
R> C <- matrix(C, ncol=5)
R> cv <- C %*% V %*% t(C)
R> cr <- matrix(rep(0, ncol(cv)^2),
                ncol=ncol(cv))
R> for (i in 1:5) {
     for (j in 1:5) {
       cr[i,j] <- cv[i,j] / sqrt(cv[i,i]*cv[j,j])
     }
   }
R> delta <- rep(0,5)
R> myfct <- function(q, alpha) {
     lower <- rep(-q, ncol(cv))
     upper <- rep(q, ncol(cv))
     pmvt(lower, upper, df, cr, delta,
          abseps=0.0001)$value - alpha
   }
R> round(uniroot(myfct, lower=1, upper=5,
                 alpha=0.95)$root, 3)
[1] 2.561
```

Here `n` is the sample size vector of each level of the factor, `V` is the covariance matrix of β. With the contrasts $\mathbf{C}$ we can compute the correlation matrix `cr` of $\mathbf{C}\beta$. Finally, we are interested in the 95% quantile of W. A wrapper function `myfct` computes the difference of the multivariate t probability for quantile `q` and `alpha`. The `alpha` quantile can now be computed easily using `uniroot`. The 95% quantile of W in this example is 2.561; reference (1) obtained 2.562 using 80.000 simulation runs. The computation needs 8.06 seconds total time on a Pentium III 450 MHz with 256 MB memory.

Using package **mvtnorm**, the efficient computation of multivariate normal or t probabilities is now available in R. We hope that this is helpful to users / programmers who deal with multiple testing problems.

Bibliography

[1] Don Edwards and Jack J. Berry. The efficiency of simulation-based multiple comparisons. *Biometrics*, 43:913–928, December 1987. 28, 29

[2] A. Genz and F. Bretz. Numerical computation of multivariate t-probabilities with application to power calculation of multiple contrasts. *Journal of Statistical Computation and Simulation*, 63:361–378, 1999. 27, 28

[3] Alan Genz. Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*, 1:141–149, 1992. 27, 28

[4] Alan Genz. Comparison of methods for the computation of multivariate normal probabilities. *Computing Science and Statistics*, 25:400–405, 1993. 27

[5] P. D. Watson, M. B. Wolf, and I. S. Beck-Montgemery. Blood and isoproterenol reduce capillary permeability in cat hindlimb. *The American Journal of Physiology*, 252:H47–H53, 1987. 28

Torsten Hothorn
Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Medizininformatik, Biometrie und Epidemiologie
Waldstraße 6, D-91054 Erlangen
Torsten.Hothorn@rzmail.uni-erlangen.de

Frank Bretz
Universität Hannover
LG Bioinformatik, FB Gartenbau
Herrenhäuser Str. 2
D-30419 Hannover
bretz@ifgb.uni-hannover.de

Alan Genz
Department of Mathematics
Washington State University
Pullman, WA 99164-3113 USA
alangenz@wsu.edu

The first author gratefully acknowledges support by Deutsche Forschungsgemeinschaft, grant SFB 539 / A4.

Programmer's Niche

Edited by Bill Venables

Save the environment

When you start to learn how to program in S you don't have to get very far into it before you find that the scoping rules can be rather unintuitive. The sort of difficulty that people first encounter is often something like the following (on S-PLUS 2000):

```
> twowaymeans <- function(X, f)
    apply(X, 2, function(x) tapply(x, f, mean))
> twowaymeans(iris[,1:2], iris$Species)
Error in FUN(X): Object "f" not found
Dumped
```

The dismay expressed by disappointed neophytes on S-news is often palpable. This is not helped by the people who point out that on R it does work because of the more natural scoping rules:

```
> twowaymeans(iris[,1:2], iris$Species)
```