What's the Point of 'tensor'?

by Jonathan Rougier

One of the packages on CRAN is called **tensor**, and it exists to compute tensor products of arrays. The tensor product is a generalisation of the matrix product, but you can go a lot further if you want: to experience the full power of tensor methods in statistics you should consult McCullagh (1987), not for the faint-hearted.

Tensor notation

In statistics it is often necessary to compute measures based on linear combinations of vectors of uncertain quantities. For example, if x and y are uncertain we might like to know $\mathsf{Cov}[Ax, By]$ where A and B are known matrices. In this case we know that the answer in matrix-form is $A\,\mathsf{Cov}[x, y]\,B^\top$.

Tensors provide an alternative to the matrix-form for representing linear combinations. The convention with tensors is to use a representitive component to stand for the whole object, and to let repeated indices denote sums of products across objects. Thus instead of writing $v = Ax$ in matrix-form, we would write $v_i = A_{ij}\,x_j$ in tensor-form, by which we mean $v_i = \sum_j a_{ij}\,x_j$, for $i = 1, \dots, m$ where A has m rows.

The great advantage of tensors is their transparency in use with linear operators. To compute $\mathsf{Cov}[Ax, By]$, for example, first write it as a tensor then treat everything as scalars:

$$\begin{aligned}\mathsf{Cov}[Ax, By] &= \mathsf{Cov}[A_{ij}\,x_j,\ B_{k\ell}\,y_\ell] \\ &= A_{ij}\,\mathsf{Cov}[x_j,\ y_\ell]\,B_{k\ell}.\end{aligned}$$

To do a tensor calculation the `tensor` function has to know what the objects are and what their common indices are. Using `t` and `%*%` would be faster here, but in tensor-form we might do `tensor(tensor(A, Cxy, 2, 1), B, 2, 2)`. The inner call zips up the j index between A_{ij} and $\mathsf{Cov}[x_j,\ y_\ell]$, and creates an object with index set $i\ell$, and the outer call zips up the ℓ index with $B_{k\ell}$ to leave an object with index set ik. Another way to do the calculation would be `tensor(A %o% B, Cxy, c(2, 4), c(1, 2))`, where '%o%' computes the outer product (this is still two tensor products but only one is apparent: the outer product is a degenerate tensor product with no summing). This way creates an object with index set $ijk\ell$ and then zips up $j\ell$ with $\mathsf{Cov}[x_j,\ y_\ell]$.

Here's another example of the transparency of tensor-form. I am always forgetting the expectation of the quadratic form $x^\top Ax$ in terms of the mean μ and variance Σ of x, which, for reference, is usually written $\mathsf{E}[x^\top Ax] = \mu^\top A\mu + \mathrm{tr}(A\Sigma)$. Write the quadratic form as a tensor, however, and we see that

$$\begin{aligned}\mathsf{E}[x^\top Ax] &= \mathsf{E}[x_i\,A_{ii'}\,x_{i'}] \\ &= A_{ii'}\,\mathsf{E}[x_i\,x_{i'}] \\ &= A_{ii'}\,\{\mu_i\,\mu_{i'} + \Sigma_{ii'}\}\,.\end{aligned}$$

So although you might be tempted by `t(m) %*% A %*% m + sum(diag(A %*% S))` from the textbook, the smart money is on `sum(A * (m %o% m + S))`. We do not need the `tensor` function here, but it would be `tensor(A, m %o% m + S, c(1, 2), c(1, 2))`.

Rectangular objects

The greatest strength of tensor-form occurs in the generalisation from matrices to arrays. Often uncertain objects possess a rectangular structure, e.g. $X = \{x_{ijk}\}$ might have a natural three-dimensional structure. The mean of X is an array of the same shape as X, while the variance of X is an array with the same shape as the outer product of X with itself. Thus the variance of X_{ijk} is, in tensor form, the six-dimensional object

$$\begin{aligned}C_{ijki'j'k'} &= \mathsf{Cov}[X_{ijk},\ X_{i'j'k'}] \\ &= \mathsf{E}[x_{ijk}\,x_{i'j'k'}] - \mathsf{E}[x_{ijk}]\,\mathsf{E}[x_{i'j'k'}]\end{aligned}$$

where primed indices are used to indicate similarity.

Even if X is two-dimensional (i.e. a matrix), sticking with the matrix-form can cause problems. For example, to compute $\mathsf{Cov}[AXB,\ X]$ we might use the identity $\mathrm{vec}(AXB) = (B^\top \otimes A)\,\mathrm{vec}\,X$, where '$\otimes$' denotes the kronecker product and '$\mathrm{vec}\,X$' creates a vector by stacking the columns of X. Originally, that is why I contributed the function `kronecker` to R. Then we would have

$$\mathsf{Cov}[\,\mathrm{vec}(AXB),\ \mathrm{vec}\,X] = (B^\top \otimes A)\,\mathsf{Var}\,[\,\mathrm{vec}\,X]$$

which requires us to vectorise the naturally two-dimensional object X to give us a two-dimensional representation of the naturally four-dimensional object $\mathsf{Var}\,[X]$. In tensor-form we would have instead

$$\mathsf{Cov}[A_{ij}\,X_{jk}\,B_{kl},\ X_{j'k'}] = A_{ij}\,B_{kl}\,\mathsf{Cov}[X_{jk},\ X_{j'k'}]$$

where $\mathsf{Cov}[X_{jk},\ X_{j'k'}]$ preserves the natural shape of the variance of X. We might compute this by

```
tmp <- tensor(tensor(A, varX, 2, 1), B, 2, 1)
aperm(tmp, c(1, 4, 2, 3))
```

The temporary object `tmp` has index set $ij'k'l$ which is permuted using `aperm` to the set $ilj'k'$. The same result could be achieved more spectacularly using

```
tensor(A %o% B, varX, c(2, 3), c(1, 2))
```

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

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

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 $\mathrm{E}\left[B_{im}\, B_{i'm'}\, B_{i''m''}\, B_{i'''m'''}\right]$. 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

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