R News

The Newsletter of the R Project — Volume 3/3, December 2003

Editorial

by Friedrich Leisch

Another year is over and it is time for change in the editorial board of R News. Thomas Lumley will take over as Editor-in-Chief for 2004, and Paul Murrell will join him and Doug Bates as board members.

Three years ago, in the editorial of issue 1/1, Kurt Hornik and I outlined possible topics for articles in the newsletter; I am pleased to say that issue 3/3 covers all of these types of topics and much more.

This issue starts with an introductory guide on dimension reduction for data mapping, followed by two articles on applications that describe how R can be used as a simulation tool in empirical ecology and for the analysis of student achievements over time. Of interest to many users is creation of web pages from within R, the topic of the R2HTML package article. For programmers and those interested in getting a little bit more out of R we have a Programmer's Niche column on regular expressions, an introduction to a new debugging facility, and an introduction to the lmeSplines package.

Certainly one of the key strengths of R is the packaging system. We now have about 300 packages on CRAN and another 50 in the Bioconductor repository, and the lists keep growing faster and faster. This wealth of packages can have a down side. In my experience the distinction between a "package" and a "library" or why it is hard to install a "source" package on Windows are confusing issues, especially for beginners. This issue's Help Desk column clarifies this vocabulary and has many upseful tips on how to make your R installation easier to maintain.

At the Technische Universität Wien we are eagerly anticipating the first R users conference, useR! 2004, which will be held here in May. Over the last few years we had several developer conferences but this will be the first meeting primarily for those interested in using R for data analysis without necessarily writing software (although, in S, a software user often quickly becomes a developer). Several R core members will give keynote lectures on important features of R; in addition we hope to attract a large number of contributions from which we can compile an attractive program. In some sense this will be a conference version of R News, with bits and pieces for all R users from novices to seasoned programmers.

Looks like 2004 will be another good year for the R project. Best wishes to everyone!

Friedrich Leisch
Technische Universität Wien, Austria
Friedrich.Leisch@R-project.org

Contents of this issue:

Dimensional Reduction for Data Mapping

A practical guide using R

by Jonathan Edwards and Paul Oman

Introduction

Techniques that produce two dimensional maps of multivariate data are of increasing importance, given the recent advances in processing capability, data collection and storage capacity. These techniques are mostly used in an exploratory capacity, to highlight relationships within the dataset, often as a precursor to further analysis. An example usage on a dataset of shape descriptors is given in Edwards et al. (2003). There are many statistical approaches to this problem, and this article serves as an initial overview and practical guide to their application in R. A small comparative study is also presented, to highlight the ease with which the analysis can be performed, and to demonstrate the capabilities of several of the more established techniques. The interested reader is directed to Ripley (1996) for thorough treatments and derivations.

Principal component analysis

Principal Component Analysis (PCS, Pearson, 1901) is the most widely used general dimension reduction technique, and can easily be applied to mapping multivariate data. PCA is a centring (to mean), scaling (to variance) and rotation (to principal axes) produced by an orthogonal linear transform (equation 1) of the data ($\{\mathbf{x}\}_{i=1}^{n} \in \mathbb{R}^{D}$), with the aim of generating a series of uncorrelated variables ($\mathbf{y}$).

$$\mathbf{y} = \mathbf{U}^{\mathbf{T}}(\mathbf{x} - \bar{\mathbf{x}}) \qquad (1)$$

A standard solution to finding $\mathbf{U}^{\mathbf{T}}$, and hence $\mathbf{y}$, is the spectral decomposition ($\boldsymbol{\Sigma} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathbf{T}}$) of the covariance matrix [1] ($\boldsymbol{\Sigma} = \mathbb{E}((\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^{T})$), where $\boldsymbol{\Lambda}$ is the diagonal eigenvalue matrix, with eigenvalues ordered $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$. For dimension reduction to dimension L (for mapping, normally 2), PCA has the desirable property that in a linear sense, the largest variance is explained by the first L components. Due to its ubiquity and also because it forms a basis for other approaches, many packages implement PCA. In practice they mainly rely on the **base** routines of `svd` and `eigen` to perform eigendecomposition [2]. The author's preferred implementations are the `prcomp` and `princomp` (with formula interface) functions supplied by the **mva** package (The interested reader should also examine the **ade4** package). These functions make the application of PCA to dataframes a trivial task. The following lines of R code perform PCA on the dataset `g54` (which is used as data in all the proceeding examples, see section "data generation" at the end of this article for details):

```
> out.pca   <- prcomp(g54[,1:4])
```

If we want to use correlations instead of covariances, scaling is required, so the function call becomes `prcomp(g54[,1:4], scale=TRUE)`.

The mapping process is then simply the plotting of the first two principal components. This can be performed using a `biplot`, or via the standard `plot` call

```
> plot(out.pca$x[,1:2],
+   main ="prcomp")
```

see Figure 1.

Multidimensional scaling

MultiDimensional Scaling (MDS) is a method of embedding the distance information of a multivariate dataset, in a reduced dimension L, by seeking a set of vectors ($\{\hat{\mathbf{x}}\}_{i=1}^{n} \in \mathbb{R}^{L}$) that reproduce these distances. One of the advantages of this direct approach is that an arbitrary distance metric can be used in the original multivariate domain. Both metric and non-metric approaches to MDS have been developed. Metric MDS aims to embed the distance directly in to the mapping domain. The classical analytical approach (Gower, 1966) (often referred to as Principal Coordinate Analysis (PCoA)) utilises the first two components of the spectral decomposition of the distance matrix ($\mathbf{U}\boldsymbol{\Sigma}^{\frac{1}{2}}$ generates the coordinates) to explain the maximal distance. If Euclidean distance is used then this approach is equivalent to covariance based PCA (see (Webb, 2002) page 346). Sammon mapping (Sammon jnr, 1969) is a more commonly used approach. A distortion measure (in MDS parlance referred to as *Stress*) is used to compare the dataset distances with an equal number of randomly distributed, two-dimensional points in the mapping space. The Stress measure S is represented by,

$$S_{sam} = \frac{1}{\sum_{i<j} d_{ij}^{2}} \sum_{i<j} \frac{(d_{ij} - \hat{d}_{ij})^2}{d_{ij}} \qquad (2)$$

[1] Correlation can also be used if one requires scaled variables in $\mathbf{x}$

[2] These techniques are $O(D^3)$ (Carreira-Perpinan, 1997, page 10). It appears that MATLAB versions of these functions allow the user to select the number of eigenvectors/values calculated (Nabney, 2001, page 230), the author is interested in how these approaches might be adopted in R.

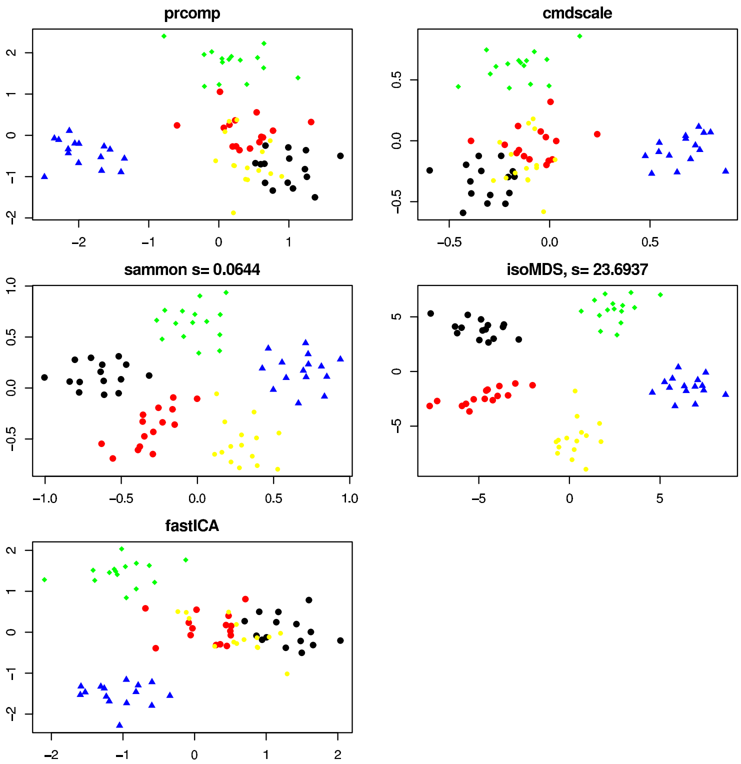

Figure 1: Maps of the Gaussian5 dataset (5 multivariate Gaussian clusters in 4-dimensional space)

d_{ij} is the distance vectors i and j,$\hat{d}$ is the estimated distance in dimension L. S_{sam} is minimised using an appropriate iterative error minimisation scheme, Sammon proposed a pseudo-Newtonian method with a step size, α normally set to 0.3. Ripley ((Ripley, 1996) page 309) discusses a crude but extremely effective line search for step size, which appears to significantly enhance the optimisation process.

Non-Metric MDS produces a map that maintains the rank of distances within the original dataset. The most widely known approach is due to Kruskal (Kruskal, 1964) which uses a similar iterative Stress function minimisation to the Sammon map, the stress function

$$S_{kruskal} = \sqrt{\frac{\sum_{i<j}(d_{ij} - \hat{d}_{ij})^2}{\sum_{i<j} \hat{d}_{ij}^2}} \qquad (3)$$

which is again minimised, using a suitable gradient descent technique.

All the MDS techniques discussed above (including Ripley's adaptive stepsize optimisation scheme) are supported in R as a part of the **mva** and **MASS** packages (classical MDS (`cmdscale`), Sammon mapping (`sammon`) and non metric MDS (`isoMDS`)). Additionally, a further implementation of the Sammon map is available in the **multiv** package, however, this appears to be less numerically stable. All the MDS functions utilise a distance matrix (normally generated using the function `dist`) rather than a matrix or data.frame, hence there is considerable flexibility to employ arbitrary distance measures. Although the function's defaults are often adequate as a first pass, the two iterative techniques tend to require a more forceful optimisation regime, with a lower "desired" stress, and more iterations, from a random initial configuration, to achieve the best mapping. The R code below performs all three mapping techniques, using the more rigorous optimisation strategy

```
> dist.data <- dist(g54[,1:4])
> out.pcoa  <- cmdscale(dist.data)
> randstart <- matrix(runif(nrow(g54[,1:4])*2),
+   nrow(g54[,1:4]),2)
> out.sam <- sammon(dist.data,y=randstart,
+   tol=1e-6,niter=500)
> out.iso <- isoMDS(dist.data,y=randstart,
+   tol=1e-6,maxit=500)
```

Again, plotting can be done with the standard R plotting commands.

```
> plot(out.pcoa, main="cmdscale")
> plot(out.sam$points), ,ylab="",xlab="",
+   main=paste("sammon s=",as.character(
+   round(out.sam$stress,4))))
> plot(out.iso$points,xlab="",ylab="",
+   main=paste("IsoMDS, s=",
+   as.character(round(out.iso$stress,4))))
```

It is also worthwhile to produce a distortion plot (see Figure 2), this plots the original distances against the mapped distances, and can be used to assess the accuracy of distance reproduction in L.

```
> plot(as.matrix(dist.data),
+   as.matrix(dist(out.sam$points)),
+   main="Distortion plot: Sammon map",
+   ylab="2d distance",xlab="nd distance")
```

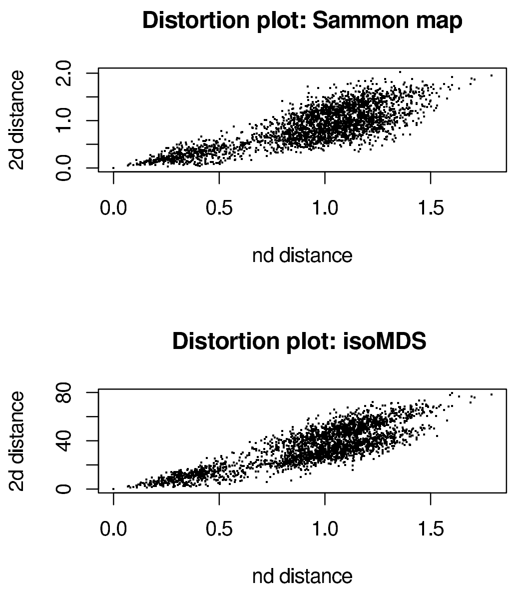

Figure 2: Distortion plot for the Gaussian5 dataset. Note pch=20 was used.

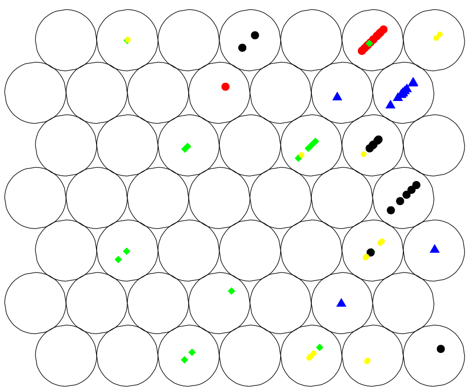

Figure 3: SOM of the Gaussian5 dataset.

The self organising map

The Self Organising Map (SOM) (Kohonen, 1982) is a variation on the K-means algorithm, which generates a map (typically a hexagonal lattice) of topologically arranged cluster centroids. The algorithm proceeds in the same manner as K-means with a *winner-takes-all* approach to updating the centroids (m_i where i is some positional reference). The update rule is

$$m_i(t+1) = m_i(t) + h_{ci}[x(t) - m_i(t)] \quad (4)$$

Neighbourhoods h_{ci} are normally calculated as $h_{ci} = \alpha(t)$ if $i \epsilon N$ else 0, where N is a set of centroids in proximity to the "winning" centroid. α is the learning rate, which is decreased to schedule the optimisation of the map. There are two packages that implement SOM's, **som** and **class**. These packages both appear to be interfaces to the standard SOMPAK C library (Kohonen et al., 1996).

Once the map has settled, several visualisation approaches can be applied (see (Vesanto, 1999) for overview). The most straightforward method in R (suggested in (Ripley, 1996)) is to allocate, via *nearest neighbour* search, each example in the dataset to a cluster centre. The following R code, using **class** (adapted from the `SOM` help file in R) produces a SOM and maps it using the nearest neighbour approach (see Figure 3):

```
> gr <- somgrid(topo = "hexagonal",
+   xdim=5,ydim=5)

> out.som <- SOM(g54[,1:4], gr, alpha =
+   list(seq(0.05, 0, len = 1e4),
+   seq(0.02, 0, len = 1e5)),
+   radii = list(seq(8, 1, len = 1e4),
+   seq(4, 1, len = 1e5)))

> plot(out.som$grid, type = "n")

> symbols(out.som$grid$pts[, 1],
+   out.som$grid$pts[, 2],
+   circles = rep(0.4, 25),
+   inches = FALSE, add = TRUE)

> bins <- as.numeric(knn1(out.som$codes,
+   g54[,1:4], 0:24))

> points(out.som$grid$pts[bins, ] +
+   rnorm(nrow(g54), 0, 0.1),
+   col=classcol[rep(1:5,e=15)],pch=20)
```

Independent component analysis and projection pursuit

Independent Component Analysis (ICA) (Jutten and Hérault, 1991) and Projection Pursuit (Huber, 1985) are variations on classical factor analysis incorporating a search for maximally "Interesting" projections. ICA takes the general form

$$\mathbf{x} = \mathbf{G}\mathbf{s} \quad (5)$$

where $\mathbf{G}$ is the *mixing matrix*, and $\mathbf{s} \in \mathbb{R}^{L<D}$ are the latent variables.

The task then is to find $\mathbf{G}$ which satisfies the above equation, and maximises an "interestingness" function on $\mathbf{s}$, $I(\mathbf{s})$. Predictably, given the subjectivity of the term "interesting", there are many forms of $I(\mathbf{s})$, exponents of ICA and Projection Pursuit both agree that an important aspect is deviation from Gaussianity. Originally algorithms focused on maximising kurtosis based measurements, as these can be efficiently calculated. However, this approach is not robust to outliers, and hence a variety of other approaches have been suggested (see (Hyvärinen, 1999b) for discussion on why non-Gaussianity is interesting, weaknesses of kurtosis and survey of alternative "interestingness" functions). A standard robust approach, which has been discussed in the literature of both Projection Pursuit and ICA, is to estimate (via approximation) the minimum mutual information via maximisation of the *negentropy*.

$$J(s) = (\mathbb{E}\, f(s) - \mathbb{E}\, f(v))^2 \qquad (6)$$

where v is a standard Normal *r.v*, and f is $f(u) = \log\cosh a_1 u$ ($a_1 \geq 1$ is a scaling constant). This method is used in the **fastICA** implementation of ICA (Hyvärinen, 1999a), and can be applied and plotted by the following R function calls:

```
> out.ica <- fastICA(g54[,1:4], 2,
+   alg.typ = "deflation",
+   fun = "logcosh", alpha = 1,
+   row.norm = FALSE, maxit = 200,
+   tol =0.0001)
```

again, the first two components (`out.ica$[,1:2]`)are plotted. XGobi and Ggobi (Swayne et al., 1998) are additional methods of performing Projection Pursuit, which although not strictly a part of R can be called using the functions in the R interface package **Rggobi**. This has less flexibility than a traditional R package, however there are some attractive features that are worth considering for exploratory multivariate analysis, particularly the *grand tour* capability, where the dataset is rotated along each of it's multivariate axes.

A small comparative study

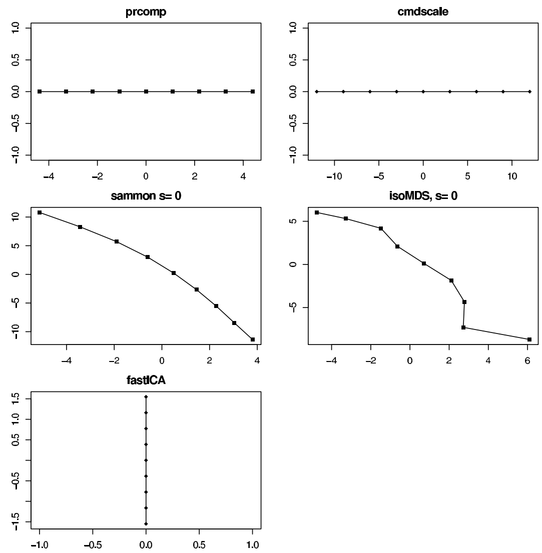

Figure 4: Maps for the Line dataset.

Dataset	n,s	c	Description
Line	9,9	×	line in 9d
Helix	3,30	×	3d spiral
Gaussian54	4,75	✓	5 Gaussians in a 4d simplex
Iris	4,150	✓	classic classifier problem

Table 1: The data sets, n, s are dimension and size.

To illustrate the capabilities of the above techniques (SOMs were not included as there are problems visualising geometric structures) a series of example maps have been generated using the datasets from Sammon's paper (Sammon jnr, 1969)(see Table 1) [3]. These datasets are used as they are easily understood and contrast the mapping performance. Interpretation of these results is fairly objective, for the geometric functions one requires a sensible reproduction, for datasets with class information (labeled 'c' in the table) a projection that maintains cluster information is desirable. For the iterative MDS techniques I will admit repeating the mapping process a few times (maximum 4!).

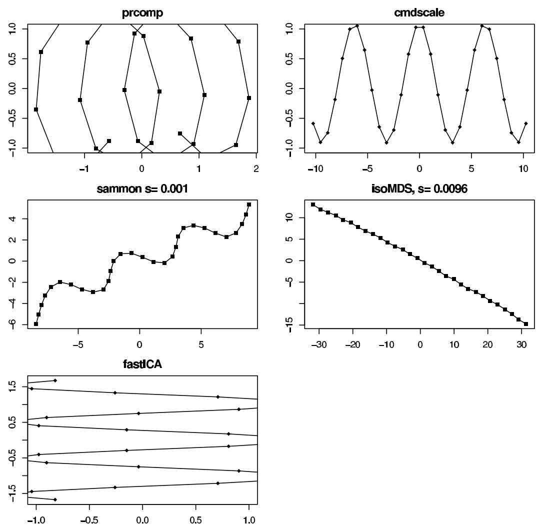

Figure 5: Maps for the Helix dataset.

Results

Line Figure 4, all techniques generate a line, apart from Non-Metric MDS which has greater freedom in the positioning of points, and hence produces a somewhat "wavy" line.

Helix Figure 5, this is an excellent dataset for a comparative study as the data is easily understood, and each algorithm gives a different interpretation. At a fundamental level the non-metric MDS (`isoMDS`) approach gives the correct interpretation since the helix is expressed in it's most "compressed" form. This approach has done too well! and isn't using all of the available dimensions. Of the other mapping techniques, most capture the sinusoidal vari-

[3] The non-linear helix dataset described in Sammon's paper has been omitted from this study as the results are similar to the those for the Helix dataset

ation (especially PCA), and hence subjectively present a better "story".

Gaussian5 Figure 1, again this data is excellent for highlighting the power of MDS techniques. *If* a reasonable minima can be found -MDS techniques offer a significant advantage in maintaining clustering information. One of the main characteristics of a cluster is that members tend to be relatively close together, hence MDS, fundamentally a *distance* based approach, has a greater capacity to maintain this relationship.

Iris Figure 6, there appears to be only minor difference in the maps produced. Of note perhaps is the tighter clustering in the MDS based approaches.

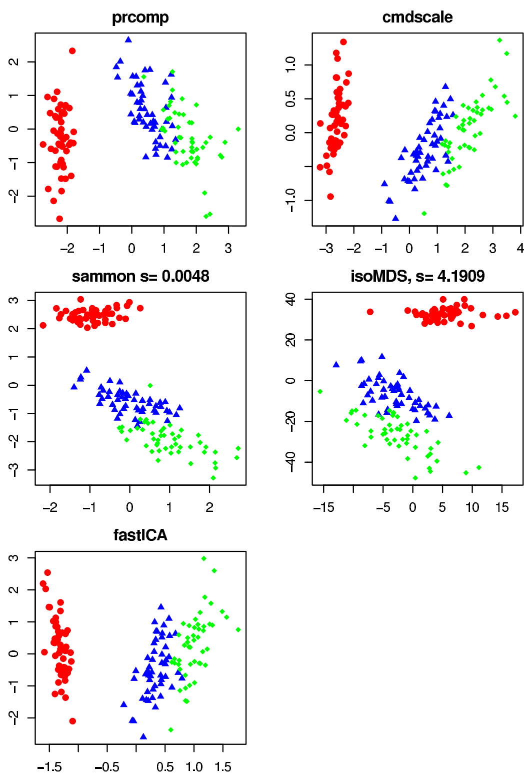

Figure 6: Maps of the Iris dataset.

Summary

As the results of the above study show, R is an excellent tool to perform dimension reduction and mapping. The **mva** package in particular provides an excellent set of classical approaches which often give the most revealing results, particularly when data contains class information. **fastICA** and the `SOM` function from **class** provide a welcome addition to the repertoire, even if they are perhaps aimed more at alternative problems. Clearly, there is scope to extend the number of projection pursuit indices, in line with those implemented in XGobi.

Dimensional reduction is an extremely active area of research, mainly within the Neural Network community. Few comparative practical studies of the application of recent techniques to mapping exist (particularly between newer techniques and "classical" MDS approaches), hence there is no strong indication of what might be usefully ported to R. A free MATLAB implementation of many of the newer techniques (Probabilistic Principal Component Analysis (Tipping and Bishop, 1997) (PPCA) mixtures of PPCA (Tipping and Bishop, 1999), Generative Topographic Mapping (GTM) (Bishop et al., 1998) and NeuroScale (Tipping and Lowe, 1998)), with excellent supporting documentation is provided by the NETLAB package (Nabney, 2001). This highlights some of the advances that have yet to appear in R, most notably techniques that are derived from a Bayesian prospective. Further notable omissions are Local Linear Embedding (Saul and Roweis, 2003) and Isomap (Tenenbaum et al., 2000) which generate a series of local representations, which are then linked together into a global mapping.

Data generation

The datasets used in this study were generated using the following functions:

```
helix <- function( size = 30 )
{
# input :  size, sample size
# output:  3d helix
    t <- 0:(size-1)
    z <- sqrt(2)/2*t
    y <- sin(z)
    x <- cos(z)
    cbind(x,y,z)
}

gauss54 <- function ( s=15 , sig=0.02 )
{
# input :  s, sample size (per Gaussian)
#          sigma, cluster _covariance_
# output:  5 4d Gaussian clusters

 simp4d <-  matrix( c(0,0,0,0,1,0,
 0, 0, 1/2, sqrt(3)/2, 0, 0, 1/2,
 sqrt(3)/6 ,sqrt(2/3),0 ,1/2 ,sqrt(3)/6 ,
 sqrt(2)/(4*sqrt(3)),sqrt(5/8))
 ,5,4,byrow=T)
 #simplex4d can easily be checked
 #using 'dist'
 rbind(
 mvrnorm(simp4d[1,],S=diag(4)*sig,n=s),
 mvrnorm(simp4d[2,],S=diag(4)*sig,n=s),
 mvrnorm(simp4d[3,],S=diag(4)*sig,n=s),
 mvrnorm(simp4d[4,],S=diag(4)*sig,n=s),
 mvrnorm(simp4d[5,],S=diag(4)*sig,n=s))
}
```

```
l9  <- matrix(rep(1:9,9),9,9)
h30 <- helix()
#g54, last column is class label
g54 <- cbind(gauss54(),rep(1:5,e=15))
#iris data is a standard data-set
data(iris)
#check for duplicates else
#distance algorithms 'blow up'
iris.data <- unique(iris[,1:4])
iris.class <- iris[row.names(iris.data),5]
#some colours for the maps
classcol <- c("red","blue","green",
"black","yellow")
```

Bibliography

C. M. Bishop, M. S., and C. K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998. URL citeseer.nj.nec.com/bishop98gtm.html. 6

M. Carreira-Perpinan. A review of dimension reduction techniques. Technical Report CS–96–09, Dept. of Computer Science, University of Sheffield, January 1997. 2

J. Edwards, K. Riley, and J. Eakins. A visual comparison of shape descriptors using multi-dimensional scaling. In *CAIP 2003, the 10th interantional conference on computer analysis of images and patterns*, pages 393–402, 2003. 2

J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, (53):325–328, 1966. 2

P. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985. 4

A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999a. URL citeseer.nj.nec.com/hyv99fast.html. 5

A. Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128, 1999b. 5

C. Jutten and J. Hérault. Blind separation of sources. *Signal Processing*, 24:1–10, 1991. 4

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. 4

T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. Som pak: The self-organizing map program package, 1996. URL citeseer.nj.nec.com/kohonen96som.html. 4

J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 127(29):115–129, 1964. 3

I. Nabney. *NETLAB: Algorithms for Pattern Recognition*. Springer, 2001. 2, 6

K. Pearson. Principal components analysis. *London Edinburgh and Dublin Philosophical Magazine and Journal*, 6(2):559, 1901. 2

B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. 2, 3, 4

J. Sammon jnr. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18: 401–409, 1969. 2, 5

L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4: 119–155, 2003. 6

D. F. Swayne, D. Cook, and A. Buja. XGobi: Interactive dynamic data visualization in the X Window System. *Journal of Computational and Graphical Statistics*, 7(1):113–130, 1998. URL citeseer.nj.nec.com/article/swayne98xgobi.html. 5

J. B. Tenenbaum, V. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(22):2319–2323, 2000. 6

M. Tipping and C. Bishop. Probabilistic principal component analysis. Technical Report NCRG/97/010, Neural Computing Research Group, Aston University, 1997. 6

M. Tipping and D. Lowe. Shadow targets: a novel algorithm for topographic projections by radial basis functions. *Neurocomputing*, 19(1):211–222, 1998. 6

M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999. URL citeseer.nj.nec.com/tipping98mixtures.html. 6

J. Vesanto. SOM-based data visualization methods. *Intelligent-Data-Analysis*, 3:111–26, 1999. URL citeseer.nj.nec.com/392167.html. 4

A. Webb. *Statistical Pattern Recognition*. Wiley, 2002. ISBN 0470845139. 2

Jonathan Edwards & Paul Oman
Department of Informatics, University of Northumbria
Newcastle upon tyne, UK
{jonathan.edwards,paul.oman}@unn.ac.uk

R as a Simulation Platform in Ecological Modelling

Thomas Petzoldt

Introduction

In recent years, the R system has evolved to a mature environment for statistical data analysis, the development of new statistical techniques, and, together with an increasing number of books, papers and online documents, an impressing basis for learning, teaching and understanding statistical techniques and data analysis procedures.

Moreover, due to its powerful matrix-oriented language, the clear interfaces and the overwhelming amount of existing basic and advanced libraries, R became a major platform for the development of new data analysis software (Tierney, 2003).

Using R firstly for post-processing, statistical analysis and visualization of simulation model results and secondly as a pre-processing tool for externally running models the question arose, whether R can serve as a general simulation platform to implement and run ecological models of different types, namely differential equation and individual-based models.

From the perspective of an empirical ecologist, the suitability of a simulation platform is often judged according to the following properties:

1. learning curve and model development time,
2. execution speed,
3. readability of the resulting code,
4. applicability to one special model type only or to a wide variety of simulation methods,
5. availability of libraries, which support model development, visualization and analysis of simulation results,
6. availability of interfaces to external code and external data,
7. portability and licensing issues.

While questions 5 to 7 can be easily answered in a positive sense for the R system, e.g. portability, free GNU Public License, the availability of external interfaces for C, Fortran and other languages and numerous methods to read and write external data, the remaining questions can be answered only with some practical experience. As a contribution, I present some illustrative examples on how ecological models can be implemented within R and how they perform. The documented code, which can be easily adapted to different situations will offer the reader the possibility to approach their own questions.

Examples

The examples were selected to show different types of ecological models and possible ways of implementation within R. Although realism and complexity are limited due to the intention to give the full source code here, they should be sufficient to demonstrate general principles and to serve as an onset for further work[1].

Differential equations

The implementation of first-order ordinary differential equation models (ODEs) is straightforward and can be done either with an integration algorithm written in pure R, for example the classical Runge-Kutta 4th order method, or using the `lsoda` algorithm (Hindmarsh, 1983; Petzold, 1983), which thanks to Woodrow Setzer are both available in the **odesolve**-library. Compared to other simulation packages this assortment is still relatively limited but should be sufficient for Lotka-Volterra type and other small and medium scale ecological simulations.

As an example I present the implementation of a recently published Lotka-Volterra-type model (Blasius et al., 1999; Blasius and Stone, 2000), the constant period – chaotic amplitude (UPCA) model. The model consists of three equations for resource u, herbivorous v, and carnivorous w animals:

$$\frac{du}{dt} = au - \alpha_1 f_1(u,v) \tag{1}$$

$$\frac{dv}{dt} = -bv + \alpha_1 f_1(u,v) - \alpha_2 f_2(v,w) \tag{2}$$

$$\frac{dw}{dt} = -c(w - w^*) + \alpha_2 f_2(v,w) \tag{3}$$

where f_1, f_2 represent either the Lotka-Volterra term $f_i(x,y) = xy$ or the Holling type II term $f_i(x,y) = xy/(1 + k_i x)$ and w^* is an important stabilizing minimum predator level when the prey population is rare.

To run the model as an initial value simulation we must provide R with (1) the model written as an R-function, (2) a parameter vector, (3) initial (start) values, and (4) an integration algorithm.

[1] Supplementary models and information are available on `http://www.tu-dresden.de/fghhihb/petzoldt/modlim/`

After loading the required libraries, the model equations (the Holling equation `f` and the derivatives `model`), can be written very similar to the mathematical notation. To make the code more readable, the state vector `xx` is copied to named state variables and the parameters are extracted from the parameter vector `parms` using the `with`environment. The results of the derivative function are returned as list.

```
library(odesolve)
library(scatterplot3d)

f <- function(x, y, k){x*y / (1+k*x)} #Holling II

model <- function(t, xx, parms) {
    u      <- xx[1]
    v      <- xx[2]
    w      <- xx[3]
    with(as.list(parms),{
    du <-  a * u  - alpha1 * f(u, v, k1)
    dv <- -b * v          + alpha1 * f(u, v, k1) +
                          - alpha2 * f(v, w, k2)
    dw <- -c * (w - wstar) + alpha2 * f(v, w, k2)
    list(c(du, dv, dw))
  })
}
```

As a next step, three vectors have to be defined: the `times` vector for which an output value is requested, the parameter vector (`parms`), and the start values of the state variables (`xstart`) where the names within the vectors correspond to the names used within the `model` function.

```
times  <- seq(0, 200, 0.1)
parms  <- c(a=1, b=1, c=10,
            alpha1=0.2, alpha2=1,
            k1=0.05, k2=0, wstar=0.006)
xstart <- c(u=10, v=5, w=0.1)
```

Now the simulation can be run using either `rk4` or `lsoda`:

```
out <- as.data.frame(lsoda(xstart, times,
                           model, parms))
```

This should take only two seconds on a standard computer and finally we can plot the simulation results as time series or state trajectory (fig. 1):

```
par(mfrow=c(2,2))
plot(times, out$u, type="l", col="green")
lines(times, out$v, type="l", col="blue")
plot(times, out$w, type="l", col="red")
plot(out$w[-1], out$w[-length(out$w)], type="l")
scatterplot3d(out$u, out$v, out$w, type="l")
```

Of course, similar results can be obtained with any other simulation package. However, the great advantage using a matrix oriented language like R or Octave[2] is, that the core model can be easily integrated within additional simulation procedures. As an example a bifurcation (Feigenbaum)-diagram of the chaotic amplitude of the predator population can be computed. First a small helper function which identifies the peaks (maxima and minima) of the time series is needed. Within a vectorized environment this can be implemented very easily by selecting all those values which are greater (resp. lower for minima) as their immediate left and right neighbours:

```
peaks <- function(x) {
  l <- length(x)
  xm1 <- c(x[-1], x[l])
  xp1 <- c(x[1], x[-l])
  x[x > xm1 & x > xp1 | x < xm1 & x < xp1]
}
```

As a next step the integration procedure is included within a main loop which increments the variable parameter `b`, evaluates the peaks and updates the plot:

```
plot(0,0, xlim=c(0,2), ylim=c(0,1.5),
          type="n", xlab="b", ylab="w")
for (b in seq(0.02,1.8,0.02)) {
  parms["b"] <- b
  out <- as.data.frame(lsoda(xstart, times,
                       model, parms, hmax=0.1))

  l <- length(out$w) %/% 3
  out <- out[(2*l):(3*l),]
  p <- peaks(out$w)
  l <- length(out$w)
  xstart <- c(u=out$u[l], v=out$v[l], w=out$w[l])
  points(rep(b, length(p)), p, pch=".")
}
```

After a stabilization phase only the last third of the detected peaks are plotted to show the behavior "at the end" of the time series. The resulting bifurcation diagram (fig. 2) shows the dependence of the amplitude of the predator w in dependence of the prey parameter b which can be interpreted as emigration parameter or as predator independent mortality. The interpretation of this diagram is explained in detail by Blasius and Stone (2000), but from the technical viewpoint this simulation shows, that within the matrix-oriented language R the integration of a simulation model within an analysis environment can be done with a very small effort of additional code. Depending on the resolution of the bifurcation diagram the cpu time seems to be relatively high (several minutes up to one hour) but considering that in the past such computations were often done on supercomputers, it should be acceptable on a personal computer.

[2] http://www.octave.org

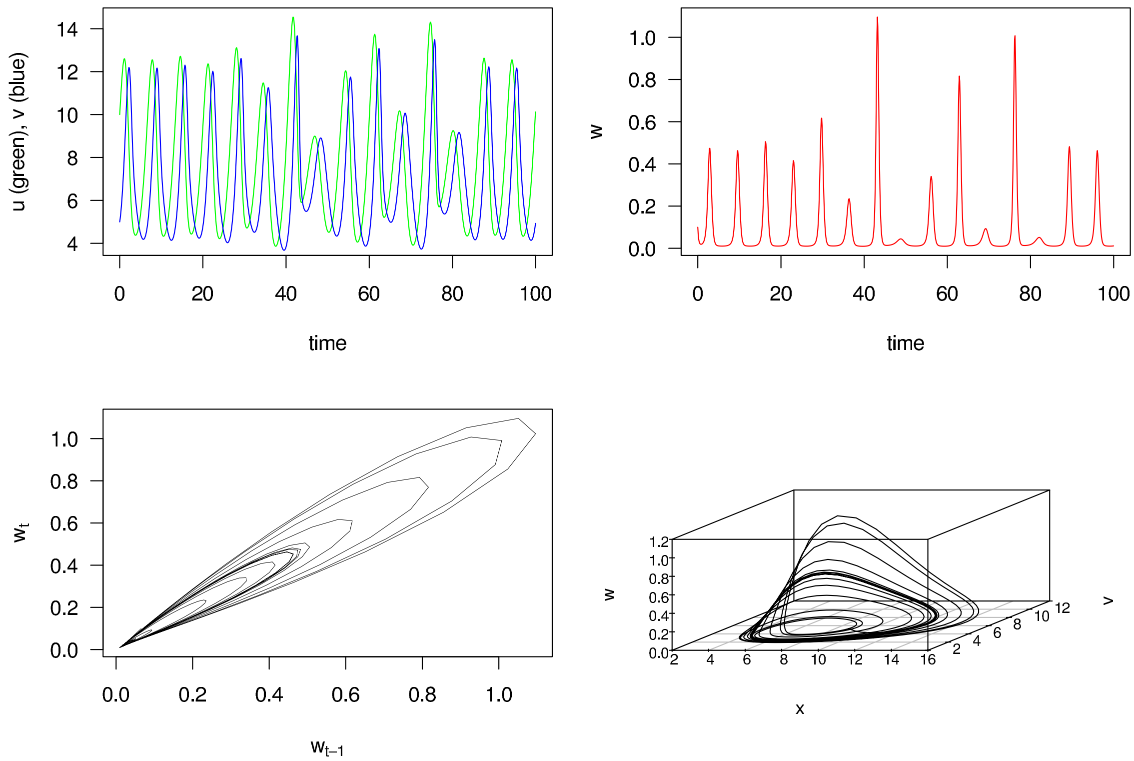

Figure 1: Simulation of an UPCA model, top left: resource (green) and prey (blue), top right: predator, bottom left: predator with lagged coordinates, bottom right: three dimensional state trajectory showing the chaotic attractor.

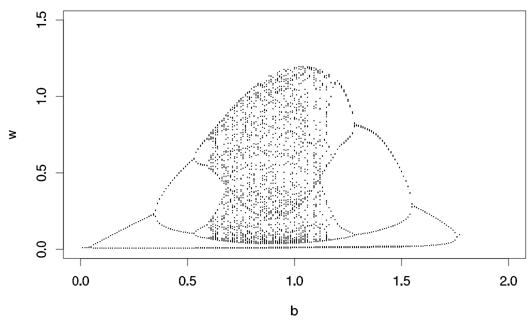

Figure 2: Bifurcation diagram for the predator w in dependence of prey parameter b.

Individual-based models

In contrast to ODE models, which in most cases work on an aggregated population level (abundance, biomass, concentration), individual based models are used to derive population parameters from the behavior of single individuals, which are commonly a stochastic sample of a given population. During the last decade this technique, which is in fact a creative collection of discrete event models, became widely used in theoretical and applied ecology (DeAngelis and Gross, 1992, and many others). Among them are very different approaches, which are spatially aggregated, spatially discrete (grid-based or cellular automata), or with a continuous space representation (particle diffusion type models).

Up to now there are different simulation tools available, which are mostly applicable to a relatively small class of individual-based approaches, e.g. SARCASim[3] and EcoBeaker[4] for cellular automata or simulation frameworks like OSIRIS (Mooij and Boersma, 1996). However, because the ecological creativity is higher than the functionality of some of these tools, a large proportion of individual-based models is implemented using general-purpose languages like C++ or Delphi, which in most cases, requires the programmer to take care of data input and output, random number generators and graphical visualization routines.

A possible solution of this tradeoff are matrix oriented languages like Matlab[5] (for individual-based models see Roughgarden, 1998), Octave or the S language. Within the R system a large collection of fundamental statistical, graphical and data manipu-

[3] http://www.collidoscope.com/ca/
[4] http://www.ecobeaker.com/
[5] http://www.mathworks.com/

lation functions is readily available and therefore it is not very difficult to implement individual-based models.

Individual-based population dynamics

The population dynamics of a *Daphnia* (water flee) population is shown here as an example. *Daphnia* plays an important role for the water quality of lakes (e.g. Benndorf et al., 2001), can be held very easily in the laboratory and the transparent skin makes it possible to observe important life functions using the microscope (e.g. food in the stomach or egg numbers in the brood pouch), so the genus *Daphnia* became an outstanding model organism in limnology, comparable to *Drosophila* in genetic research.

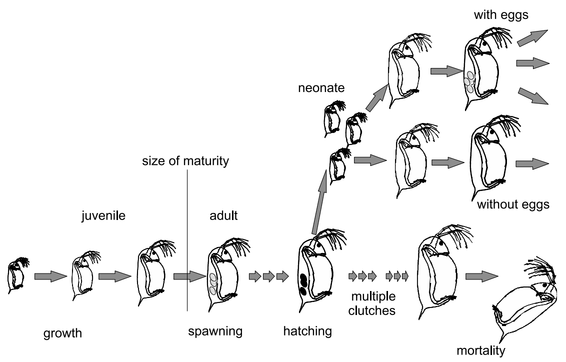

Figure 3: Parthenogenetic life cycle of *Daphnia*.

Under normal circumstances the life cycle of *Daphnia* is asexual (parthenogenetic) and begins with one new-born (neonate) female which grows up to the size of maturity (fig. 3). Then a number of asexual eggs are deposited into the brood pouch (spawning) which then grow up to neonates and which are released after a temperature-dependent time span into the free water (hatching). There are several approaches to model this life cycle in the literature (e.g. Kooijman, 1986; Gurney et al., 1990; Rinke and Petzoldt, 2003). Although the following is an absolutely elementary one which neglects food availability and simplifies size-dependent processes, it should be sufficient to demonstrate the principal methodology.

For the model we take the following assumptions (specified values taken from Hülsmann and Weiler (2000) and Hülsmann (2000)):

- The egg depelopment time is a function of water temperature according to Bottrell et al. (1976).
- Neonates have a size of 510 μm and the size of maturity is assumed to be 1250 μm.
- Juvenile length growth is linear with a growth rate of 83 $\mu m\, d^{-1}$ and due to the reproductive expense the adult growth rate is reduced to 80% of the juvenile value.
- The clutch size of each individual is taken randomly from an observed probability distribution of the population in Bautzen Reservoir (Germany) during spring 1999.
- The mortality rate is set to a fixed size-independent value of 0.02 d^{-1} and the maximum age is set to a fixed value of 30 d.

The implementation consists of six parts. After defining some constants, the vector with the cumulative distribution of the observed egg-frequencies, the simulation control parameters (1) and necessary helper functions, namely a function for an inverse sampling of empiric distributions (2), the life functions (3) of *Daphnia*: `grow`, `hatch`, `spawn` and `die` are implemented. Each individual is represented as one row of a data frame `inds` where the life functions either update single values of the individuals or add rows via `rbind` for each newborn individual (in `hatch`) or delete rows via `subset` for dead individuals. With the exception of the `hatch` function all procedures are possible as vectorized operations without the need of loops.

Now the population data frame is created (4) with some start individuals either as a fixed start population (as in the example) or generated randomly. The life loop (5) calls each life function for every time step and, as the `inds` data frame contains only one actual state, collects the desired outputs during the simulation, which are analysed graphically or numerically immediately after the simulation (6, see fig. 4). The example graphics show selected population statistics: the time series of abundance with an approximately exponential population growth, the mean length of the individuals as a function of time (indicating the presence of synchronized cohorts), the frequency distribution of egg numbers of adult individuals and a boxplot of the age distribution.

Additionally or as an alternative it is possible to save logfiles or snapshot graphics to disk and to analyse and visualise them with external programs.

```
#===============================================
# (1) global parameters
#===============================================
# cumulative egg frequency distribution
eggfreq     <- c(0.39, 0.49, 0.61, 0.74,
                 0.86, 0.95, 0.99, 1.00)
son         <- 510              # um
primipara   <- 1250             # um
juvgrowth   <- 83               # um/d
adgrowth    <- 0.8*juvgrowth # um/d
mort        <- 0.02             # 1/d
temp        <- 15               # deg C
timestep    <- 1                # d
steps       <- 40
#
# template for one individual
newdaphnia <- data.frame(age = 0,
                         size = son,
```

```
                        eggs = 0,
                      eggage = 0)
#===============================================
# (2) helper functions
#===============================================
botrell <- function(temp) {
    exp(3.3956 + 0.2193 *
       log(temp)-0.3414 * log(temp)^2)
}

# inverse sampling from an empiric distribution
clutchsize <- function(nn) {
    approx(c(0,eggfreq), 0:(length(eggfreq)),
           runif(nn), method="constant", f=0)$y
}
#===============================================
# (3) life methods of the individuals
#===============================================
grow <- function(inds){
    ninds       <- length(inds$age)
    inds$age    <- inds$age + timestep
    inds$eggage <- ifelse(inds$size > primipara,
                   inds$eggage + timestep, 0)
    inds$size   <- inds$size + timestep *
                   ifelse(inds$size < primipara,
                   juvgrowth, adgrowth)
    inds
}
die  <- function(inds) {
    subset(inds,
      runif(inds$age) > mort & inds$age <= 30)
}
spawn <- function(inds) {
  # individuals with size > primipara
  # and eggage==0 can spawn
  ninds          <- length(inds$age)
  neweggs        <- clutchsize(ninds)
  inds$eggs      <- ifelse(inds$size > primipara
                           & inds$eggage==0,
                           neweggs, inds$eggs)
  inds
}
hatch <- function(inds) {
    # individuals with eggs
    # and eggage > egg development time hatch
    ninds          <- length(inds$age)
    newinds        <- NULL
    edt <- botrell(temp)
    for (i in 1: ninds) {
        if (inds$eggage[i] > edt) {
          if (inds$eggs[i] > 0) {
            for (j in 1:inds$eggs[i]) {
                newinds <- rbind(newinds,
                             newdaphnia)
             }
          }
          inds$eggs[i]   <- 0
          inds$eggage[i] <- 0
        }
    }
    rbind(inds, newinds)
}
#===============================================
# (4) start individuals
#===============================================
inds <- data.frame(age  = c(7, 14, 1, 11, 8,
                            27, 2, 20, 7, 20),
                   size = c(1091, 1339, 618, 1286,
                            1153, 1557, 668, 1423,
                            1113, 1422),
                   eggs = c(0, 0, 0, 5, 0,
                            3, 0, 3, 0, 1),
                 eggage = c(0, 0, 0, 1.6, 0,
                            1.5, 0, 1.7, 0, 1.2))
#===============================================
# (5) life loop
#===============================================
sample.n        <- NULL
sample.size     <- NULL
sample.eggs     <- NULL
sample.agedist  <- NULL

for (k in 1:steps) {
    print(paste("timestep",k))
    inds <- grow(inds)
    inds <- die(inds)
    inds <- hatch(inds)
    inds <- spawn(inds)
    sample.n        <- c(sample.n,
                         length(inds$age))
    sample.size     <- c(sample.size,
                         mean(inds$size))
    sample.eggs     <- c(sample.eggs,
                         inds$eggs[inds$size >
                         primipara])
    sample.agedist  <- c(sample.agedist,
                         list(inds$age))
}
#===============================================
# (6) results and graphics
#===============================================
par(mfrow=c(2,2))
plot(sample.n, xlab="time (d)",
     ylab="abundance", type="l")
plot(sample.size, xlab="time (d)",
     ylab="mean body length (ţm)", type="l")
hist(sample.eggs, freq=FALSE, breaks=0:10,
     right=FALSE, ylab="rel. freq.",
     xlab="egg number", main="")
time <- seq(1,steps,2)
boxplot(sample.agedist[time],
        names=as.character(time), xlab="time (d)",
        ylab="age distribution (d)")
```

Particle diffusion models

In individual based modelling two different approaches are used to represent spatial movement: cellular automata and continuous coordinates (diffusion models). Whereas in the first case movement is realized by a state-transition of cells depending on their neighbourhood, in diffusion models the individuals are considered as particles with x and y coordinates which are updated according to a correlated or a random walk. The movement rules use polar coordinates with angle (α) and distance (r) which can be converted to cartesian coordinates using complex

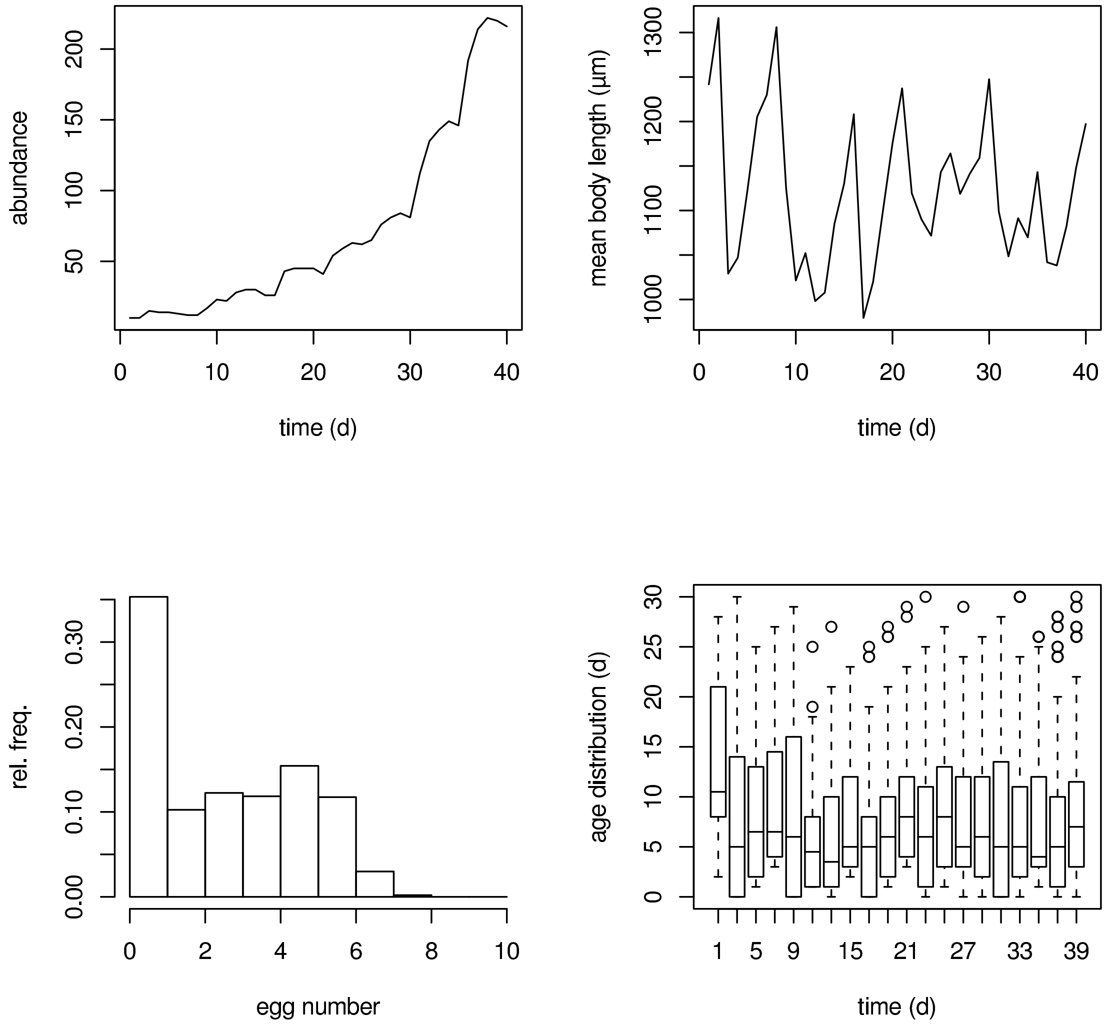

Figure 4: Results of an individual-based *Daphnia* model: exponential growth of abundance (top left), fluctuation of the mean body length indicating synchronised hatching (top right), simulated egg counts of adult individuals (bottom left), boxplot of age distribution as a function of time (bottom right).

numbers or sine and cosine transformation, what is essentially the same. Changes in direction are realized by adding a randomly selected turning angle to the angle of the time step before. An angle which is uniformly distributed within the interval $(0, 2\pi)$ results in an uncorrelated random walk, whereas an angle of less than a full circle or a nonuniform (e.g. a normal) distribution leads to a correlated random walk. To keep the individuals within the simulation area either a wrap around mode or reflection at the borders can be used.

The example shows a diffusion example with a coordinate system of $x, y = (0, 100)$ and an area of decreased movement speed in the middle $y = (45, 55)$, which can be interpreted as refugium (e.g. a hedge within a field for beatles), a region of increased food availability (the movement speed of herbivorous animals is less while grazing than foraging) or as a diffusion barrier (e.g. reduced eddy diffusion within the thermocline of stratified lakes).

The implementation is based on a data frame of individuals (`inds`) with cartesian coordinates (`x,y`) and a movement vector given as polar coordinates (`a, r`), the movement rules and the simulation loop.

The simulation runs fast enough for a realtime visualization of several hundred particles and gives an impression, how an increased abundance within refugia or diffusion barriers, observed very often in the field, can be explained without any complicated or intelligent behavior, solely on the basis of random walk and variable movement speed (fig. 5). This model can be extended in several ways and may include directed movement (sedimentation), reproduction or predatorprey interactions [6]. Furthermore, the principle of random walk can be used as a starting point for more complex theoretical models or, combined with population dynamics or bioenergetical models, as models of realworld systems (e.g. Hölker et al., 2002).

```
#===============================================
# simulation parameters and start individuals
#===============================================
n <- m <- 100  # size of simulation area
nruns  <- 2000 # number of simulation runs
nind   <- 100  # number of inds
inds   <- data.frame(x = runif(nind)*n,
                     y = runif(nind)*m,
                     r = rnorm(nind),
                     a = runif(nind)*2*pi)
#===============================================
# Movement
#===============================================
move <- function(inds) {
  with(inds, {
      ## Rule 1: Refugium
      speed <- ifelse(inds$y > 45
                   & inds$y < 55, 0.2, 2)
      ## Rule 2: Random Walk
      inds$a <- a + 2 * pi / runif(a)
      dx <- speed * r * cos(a)
      dy <- speed * r * sin(a)
      x<-inds$x + dx
      y<-inds$y + dy
      ## Rule 3: Wrap Around
      x<-ifelse(x>n,0,x)
      y<-ifelse(y>m,0,y)
      inds$x<-ifelse(x<0,n,x)
      inds$y<-ifelse(y<0,m,y)
      inds
  })
}
#===============================================
# main loop
#===============================================
for(i in 1:nruns) {
  inds <- move(inds)
  plot(inds$x, inds$y, col="red", pch=16,
     xlim=c(0, n), ylim=c(0, m),
     axes=FALSE, xlab="", ylab="")
}
```

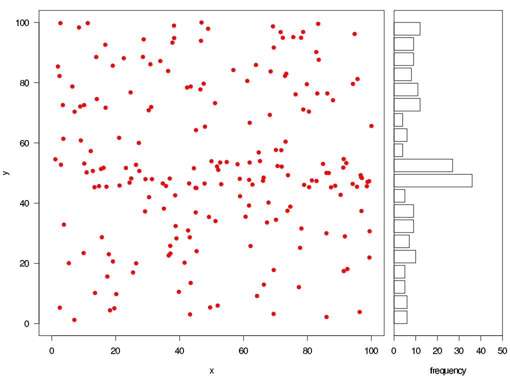

Figure 5: State of the particle diffusion model after 2000 time steps.

Cellular automata

Cellular automata (CA) are an alternative approach for the representation of spatial processes, widely used in theoretical and applied ecological modelling. As one of the most fundamental systems the well-known deterministic CA "Conway's Game of Life" (Gardner, 1970) is often used as an impressive simulation to show, that simple rules can lead to complex behavior. To implement a rectangular CA within R, a grid matrix (`z`), state transition rules implemented as `ifelse` statements and the `image` plot function are the essential building blocks. Furthermore a function (`neighbourhood`) is needed, which counts the number of active (nonzero) cells in the neighbourhood of each cell. In the strict sense only the eight adjacent cells are considered as neighbours, but in a more general sense a weighted neighbourhood matrix can be used. The neighbourhood count has to be evaluated for each cell in each time step and is therefore a computation intensive part, so the implementation as an

[6] see supplementary information

external C-function was necessary and is now available as part of the experimental **simecol** package.

Using this, Conway's Game of Life can be implemented with a very short piece of code. Part (1) defines the grid matrix with a set of predefined or random figures respectively, and part (2) is the life loop, which contains only the call of the neighbourhood function, the state transition rules and the visualization. The simulation is much slower than specialized Conway-programs (20 s on an AMD Athlon 1800+ for the example below) but still fast enough for real-time animation (0.2 s per time step). Approximately 85% of the elapsed time is needed for the visualization, only 3% for the neighbourhood function and the rest for the state transition rules.

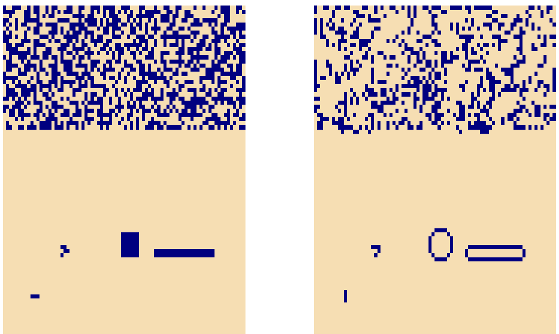

Figure 6: Initial state of matrix *z* before the simulation (left) and state after the first simulation step (right) .

However, despite the slower simulation speed it is a great advantage of this implementation, that state matrices and transition rules are under full control of the user and that the system is completely open for extension towards grid-based models with an ecological background. Furthermore the `neighbours`-function of the **simecol**-package allows the use of a generalized neighbourhood via a weight matrix to implement e.g. random or directed movement of animals, seed dispersal, forest fire propagation or epidemic models.

```
library(simecol)
#================================================
# (1) start individuals
#================================================
n <- m <- 80
z <- matrix(0, nrow=n, ncol=m)
z[40:45,20:25] <-1 # filled square 6x6 cells
z[51:70,20:21] <-1 # small rectangle 20 x 2 cells
z[10:12,10]    <-1 # short bar 3x1 cells
z[20:22,20:22] <- c(1,0,0,0,1,1,1,1,0) # glider
z[1:80,51:80]  <-round(runif(2400))    # random
image(z, col=c("wheat", "navy"), axes=FALSE)
#================================================
# (2) life loop
#================================================
for (i in 1:100){
  nb <- eightneighbours(z)
  ## survive with 2 or 3 neighbours
  zsurv <- ifelse(z > 0 & (nb == 2 | nb ==3), 1, 0)
  ## generate for empty cells with 3 neigbors
  zgen <- ifelse(z == 0 & nb == 3, 1, 0)
  z  <- matrix((zgen + zsurv), nrow=n)
  image(z, col=c("wheat","navy"),
        axes=FALSE, add=TRUE)
}
```

Conclusions

The examples described so far, plus the experience with R as data analysis environment for measurement and simulation data, allows to conclude that R is not only a great tool for statistical analysis and data processing but also a general-purpose simulation environment, both in research and especially in teaching.

It is not only an advantage to replace a lot of different tools on the modelers PC for different types of models, statistical analysis and data management with one single platform. The main advantage lies in the synergistic effects, which result from the combination and integration of these simulation models together with powerful statistics and publication quality graphics.

Being an interpreted language with a clear syntax and very powerful functions, R is easier to learn than other languages and allows rapid prototyping and and interactive work with the simulation models. Due to the ability to use vectorized functions, the execution speed is fast enough for non-trivial simulations and for real-time animation of teaching models. In many cases the code is compact enough to demonstrate field ecologists the full ecological functionality of the models. Furthermore, within a workgroup where field ecologists and modellers use the same computing environment for either statistical data analysis or the construction of simulation models, R becomes a communication aid to discuss ecological principles and to exchange ideas.

Acknowledgements

I am grateful to my collegues Karsten Rinke, Stephan Hülsmann, Robert Radke and Markus Wetzel for helpful discussions, to Woodrow Setzer for implementing the odesolve package and to the R Development Core Team for providing the great R system.

Bibliography

Benndorf, J., Kranich, J., Mehner, T., and Wagner, A. (2001). Temperature impact on the midsummer decline of *Daphnia galeata* from the biomanipulated Bautzen Reservoir (Germany). *Freshwater Biology*, 46:199–211. 11

Blasius, B., Huppert, A., and Stone, L. (1999). Complex dynamics and phase synchronization in spatially extended ecological systems. *Nature*, 399:354–359. 8

Blasius, B. and Stone, L. (2000). Chaos and phase synchronization in ecological systems. *International Journal of Bifurcation and Chaos*, 10:2361–2380. 8, 9

Bottrell, H. H., Duncan, A., Gliwicz, Z. M., Grygierek, E., Herzig, A., Hillbricht-Ilkowska, A., Kurasawa, H., Larsson, P., and Weglenska, T. (1976). A review of some problems in zooplankton production studies. *Norvegian Journal of Zoology*, 24:419–456. 11

DeAngelis, D. L. and Gross, L. J., editors (1992). *Individual-Based Models and Approaches in Ecology: Populations, Communities, and Ecosystems.*, Knoxwille, Tennessee. Chapmann and Hall. Proceedings of a Symposium/Workshop. 10

Gardner, M. (1970). The fantastic combinations of John Conway's new solitaire game 'life'. *Scientific American*, 223:120–123. 14

Gurney, W. C., McCauley, E., Nisbet, R. M., and Murdoch, W. W. (1990). The physiological ecology of *Daphnia*: A dynamic model of growth and reproduction. *Ecology*, 71:716–732. 11

Hindmarsh, A. C. (1983). ODEPACK, a systematized collection of ODE solvers. In Stepleman, R., editor, *Scientific Computing*, pages 55–64. IMACS / North-Holland. 8

Hölker, F., Härtel, S. S., Steiner, S., and Mehner, T. (2002). Effects of piscivore-mediated habitat use on growth, diet and zooplankton consumption of roach: An individual-based modelling approach. *Freshwater Biology*, 47:2345–2358. 14

Hülsmann, S. (2000). *Population Dynamics of Daphnia galeata in the Biomanipulated Bautzen Reservoir: Life History Strategies Against Food Deficiency and Predation*. PhD thesis, TU Dresden Fakultät Forst-, Geo-, Hydrowissenschaften, Institut für Hydrobiologie. 11

Hülsmann, S. and Weiler, W. (2000). Adult, not juvenile mortality as a major reason for the midsummer decline of a *Daphnia* population. *Journal of Plankton Research*, 22(1):151–168. 11

Kooijman, S. (1986). Population dynamics on the basis of budgets. In Metz, J. and Diekman, O., editors, *The dynamics of physiologically structured populations*, pages 266–297. Springer Verlag, Berlin. 11

Mooij, W. M. and Boersma, M. (1996). An object oriented simulation framework for individual-based simulations (OSIRIS): *Daphnia* population dynamics as an example. *Ecol. Model.*, 93:139–153. 10

Petzold, L. R. (1983). Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *Siam J. Sci. Stat. Comput.*, 4:136–148. 8

Rinke, K. and Petzoldt, T. (2003). Modelling the effects of temperature and food on individual growth and reproduction of *Daphnia* and their consequences on the population level. *Limnologica*, 33(4):293–304. 11

Roughgarden, J. (1998). *Primer of Ecological Theory*. Prentice Hall. 10

Tierney, L. (2003). Name space management for R. *R News*, 3(1):2–6. 8

Thomas Petzoldt
Dresden University of Technology
Institute of Hydrobiology
Dresden, Germany
petzoldt@rcs.urz.tu-dresden.de

Using R for Estimating Longitudinal Student Achievement Models

by J.R. Lockwood[1], Harold Doran and Daniel F. McCaffrey

Overview

The current environment of test-based accountability in public education has fostered increased interest in analyzing longitudinal data on student achievement. In particular, "value-added models" (VAM) that use longitudinal student achievement data linked to teachers and schools to make inferences about teacher and school effectiveness have burgeoned. Depending on the available data and desired inferences, the models can range from straightforward hierarchical linear models to more complicated and computationally demanding cross-classified models. The purpose of this article is to demonstrate how R, via the `lme` function for linear mixed effects models in the **nlme** package (Pinheiro and Bates, 2000), can be used to estimate all of the most common value-added models used in educational research. After providing background on the substantive problem, we develop notation for the data and model structures that are considered. We then present a sequence of increasingly complex models and demonstrate how to estimate the models in R. We conclude with a discussion of the strengths and limitations of the R facilities for modeling longitudinal student achievement data.

Background

The current education policy environment of test-based accountability has fostered increased interest in collecting and analyzing longitudinal data on student achievement. The key aspect of many such data structures is that students' achievement data are linked to teachers and schools over time. This permits analysts to consider three broad classes of questions: what part of the observed variance in student achievement is attributable to teachers or schools; how effective is an individual teacher or school at producing growth in student achievement; and what characteristics or practices are associated with effective teachers or schools. The models used to make these inferences vary in form and complexity, and collectively are known as "value added models" (VAM, McCaffrey et al., 2004).

However, VAM can be computationally demanding. In order to disentangle the various influences on achievement, models must account simultaneously for the correlations among outcomes within students over time and the correlations among outcomes by students sharing teachers or schools in the current or previous years. The simplest cases have student outcomes fully nested within teachers and schools, in which case standard hierarchical linear models (Raudenbush and Bryk, 2002; Pinheiro and Bates, 2000) are appropriate. When students are linked to changing teachers and/or schools over time, more complicated and computationally challenging cross-classified methods are necessary (Bates and DebRoy, 2003; Raudenbush and Bryk, 2002; Browne et al., 2001). Although the `lme` function of the **nlme** library is designed and optimized for nested structures, its syntax is flexible enough to specify models for more complicated relational structures inherent to educational achievement data.

Data structures

The basic data structures supporting VAM estimation are longitudinal student achievement data $\mathbf{Y}_k = (Y_{k0}, \ldots, Y_{kT})$ where k indexes students. Typically the data represent scores on an annual standardized examination for a single subject such as mathematics or reading, with $t = 0, \ldots, T$ indexing years. For clarity, we assume that all students are from the same cohort, so that year is coincident with grade level. More careful consideration of the distinction between grade level and year is necessary when modeling multiple cohorts or students who are held back. We further assume that the scores represent achievement measured on a single developmental scale so that Y_{kt} is expected to increase over time, with the gain scores $(Y_{kt} - Y_{k,t-1})$ representing growth along the scale. The models presented here can be estimated with achievement data that are not scaled as such, but this complicates interpretation (McCaffrey et al., 2004).

As noted, the critical feature of the data underlying VAM inference is that the students are linked, over time, to higher-level units such as teachers and schools. For some data structures and model specifications, these linkages represent a proper nesting relationship, where outcomes are associated with exactly one unit in each hierarchical level (e.g. outcomes nested within students, who are nested within schools). For more complex data, however, the

[1]This material is based on work supported by the National Science Foundation under Grant No. 99-86612. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

stuid	schid	y0.tchid	y1.tchid	y2.tchid	y0	y1	y2
1	1	3	102	201	557	601	629
2	1	3	101	203	580	632	660
3	1	2	102	202	566	620	654
			...				
287	4	14	113	215	590	631	667
288	4	14	114	213	582	620	644
289	4	13	113	214	552	596	622
			...				

Table 1: Sample records from `wdat`

linkages represent a mixture of nested and cross-classified memberships. For example, when students change teachers and/or schools over time, the repeated measures on students are associated with different higher-level units. Complicating matters is that some models may wish to associate outcomes later in the data series with not only the current unit, but past units as well (e.g. letting prior teachers affect current outcomes).

Depending on the type of model being fit, the data will need to be in one of two formats for use by `lme`. In the first format, commonly called "wide" or "person-level" format, the data are stored in a `dataframe` where each student has exactly one record (row), and repeated measures on that student (i.e. scores from different years) are in different fields (columns). In the other form, commonly called "long" or "person-period" format, repeated measures on students are provided in separate records, with records from the same student in consecutive, temporally ordered rows. Interconversion between these formats is easily carried out via the `reshape` function. In some settings (particularly with fully nested data structures) making the `dataframe` a `groupedData` object, a feature provided by the **nlme** library, will simplify some model specifications and diagnostics. However, we do not use that convention here because some of the cross-classified data structures that we discuss do not benefit from this organization of the data.

Throughout the article we specify models in terms of two hypothetical dataframes `wdat` and `ldat` which contain the same linked student achievement data for three years in wide and long formats, respectively. For clarity we assume that the outcome data and all students links to higher level units are fully observed; we discuss missing data at the end of the article. In the wide format, the outcomes for the three years are in separate fields `y0`, `y1`, and `y2`. In the long format, the outcomes for all years are in the field `yt`, with year indicated by the numeric variable `year` taking on values of 0, 1 and 2. The student identifiers and school identifiers for both dataframes are given by `stuid` and `schid`, respectively, which we assume are constant across time within students (i.e. students are assumed to be nested in schools). For the wide format dataframe, teacher identifiers are in separate fields `y0.tchid`, `y1.tchid`, and `y2.tchid` which for each student link to the year 0, 1 and 2 teachers, respectively. These fields will be used to create the requisite teacher variables of the long format dataframe later in the discussion. All identifiers are coded as factors, as this is required by some of the syntax presented here. Finally, in order to specify the cross-classified models, it is necessary to augment `ldat` with a degenerate grouping variable `dumid` that takes on a single value for all records in the dataframe. This plays the role of a highest-level nesting variable along which there is no variation in the data. Sample records from the two hypothetical dataframes are provided in Tables 1 and 2.

stuid	schid	dumid	year	yt
1	1	1	0	557
1	1	1	1	601
1	1	1	2	629
2	1	1	0	580
2	1	1	1	632
2	1	1	2	660
		...		
287	4	1	0	590
287	4	1	1	631
287	4	1	2	667
288	4	1	0	582
288	4	1	1	620
288	4	1	2	644
		...		

Table 2: Sample records from `ldat`

Model specifications

In this section we consider three increasingly complex classes of VAM, depending on the dimension of the response variable (univariate versus multivariate) and on the relational structure among hierarchical units (nested versus cross-classified). We have chosen this organization to span the range of models considered by analysts, and to solidify, through simpler models, some of the concepts and syntax required by more complicated models. All of the models discussed here specify student, teacher and school effects as random effects. We focus on specification of the random effects structures, providing a discussion of fixed effects for student, teacher or school-level characteristics in a later section. It is outside of the scope of this article to address the many complicated issues that might threaten the validity of the models (McCaffrey et al., 2004); rather we focus on issues of implementation only. It is also outside of the scope of this article to discuss the variety of functions, both in R in general and in the **nlme** library in particular, that can be used to extract relevant diagnostics and inferences from model objects. The book by Pinheiro and Bates (2000) provides an excellent treatment of these topics.

In specifying the statistical models that follow, for clarity we use the same symbols in different models to draw correspondences among analogous terms (e.g. μ for marginal means, ϵ for error terms) but it should be noted that the actual interpretations of these parameters will vary by model. Finally, we use subscripts i, j, k to index schools, teachers and students, respectively. To denote linkages, we use notation of the form $m(n)$ to indicate that observations or units indexed by n are associated with higher-level units indexed by m. For example, $j(k)$ denotes the index j of the teacher linked to an outcome from student k.

1. Nested univariate models

Though VAM always utilizes longitudinal student level data linked to higher level units of teachers, schools and/or districts, analysts often specify simple models that reduce the multivariate data to a univariate outcome. They also reduce a potentially cross-classified membership structure to a nested one by linking that outcome with only one teacher or school. Such strategies are most commonly employed when only two years of data are available, though they can be used when more years are available, with models being repeated independently for adjacent pairs of years. As such, without loss of generality, we focus on only the first two years of the hypothetical three-year data.

The first common method is the "gain score model" that treats $G_{k1} = (Y_{k1} - Y_{k0})$ as outcomes linked to current year (year 1) teachers. The formal model is:

$$G_{k1} = \mu + \theta_{j1(k)} + \epsilon_{k1} \qquad (1)$$

where μ is a fixed mean parameter, θ_{j1} are iid $N(0, \sigma^2_{\theta 1})$ random year one teacher effects and ϵ_{k1} iid $N(0, \sigma^2_{\epsilon 1})$ year one student errors. Letting `g1` denote the gain scores, the model is a simple one-way random effects model specified in `lme` by

```
lme(fixed=g1~1,data=wdat,random=~1|y1.tchid)
```

In the traditional mixed effects model notation $\boldsymbol{Y}_k = \boldsymbol{X}_k\boldsymbol{\beta} + \boldsymbol{Z}_k\boldsymbol{\theta}_k + \boldsymbol{\epsilon}_k$ (Pinheiro and Bates, 2000), the `fixed` argument to `lme` specifies the response and the fixed effects model $\boldsymbol{X}_k\boldsymbol{\beta}$, and the `random` argument specifies the random effects structure $\boldsymbol{Z}_k\boldsymbol{\theta}_k$. The `fixed` argument in the model above specifies that the gain scores have a grand mean or intercept common to all observations, and the `random` argument specifies that there are random intercepts for each year one teacher.

The other common data reduction strategy for multivariate data is the "covariate adjustment" model that regresses Y_{k1} on Y_{k0} and current year teacher effects:

$$Y_{k1} = \mu + \beta Y_{k0} + \theta_{j1(k)} + \epsilon_{k1} \qquad (2)$$

This model is also easily specified by

```
lme(fixed=y1~y0,data=wdat,random=~1|y1.tchid)
```

As a matter of shorthand notation used throughout the article, in both the `fixed` and `random` arguments of `lme`, the inclusion of a covariate on the right side of a tilde implies the estimation of the corresponding intercept even though the `+1` is not explicitly present.

Both of these models are straightforwardly extended to higher levels of nesting. For example, in the covariate adjustment model Equation (2) it may be of interest to examine what part of the marginal teacher variance lies at the school level via the model:

$$\begin{aligned} Y_{k1} &= \mu + \beta Y_{k0} + \theta_{j1(k)} + \epsilon_{k1} \\ \theta_{j1} &= \gamma_{i(j1)} + e_{j1} \end{aligned} \qquad (3)$$

with γ_i, e_{j1} and ϵ_{k1} independent normal random variables with means 0 and variances σ^2_γ, σ^2_{e1} and $\sigma^2_{\epsilon 1}$ respectively. Thus $\sigma^2_{\theta 1}$ from the previous model is decomposed into the between-school variance component σ^2_γ and the within-school variance component σ^2_{e1}. Using standard nesting notation for statistical formulae in R this additional level of nesting is specified with

```
lme(fixed=y1~y0,data=wdat,
    random=~1|schid/y1.tchid)
```

where the `random` statement specifies that there are random intercepts for schools, and random intercepts for year one teachers within schools.

2. Nested multivariate models

The next most complicated class of models directly treats the longitudinal profiles $\boldsymbol{Y}_k = (Y_{k0}, \ldots, Y_{kT})$ as outcomes, rather than reducing them to univariate outcomes, and the challenge becomes modeling the mean and covariance structures of the responses. In this section we assume that the multivariate outcomes are fully nested in higher level units; in the hypothetical example, students are nested in schools, with teacher links ignored (in actual data sets teacher links are often unavailable). For multivariate modeling of student outcomes, it is common for analysts to parameterize growth models for the trajectories of the outcomes, and to examine the variations of the parameters of the growth models across students and schools.

The class of models considered here is of the form

$$Y_{kt} = \mu_t + s_{it(k)} + \epsilon_{kt} \quad (4)$$

where μ_t is the marginal mean structure, s_{it} are schoolspecific growth trajectories, and ϵ_{kt} are studentspecific residual terms which may themselves contain parameterized growth trajectories. For clarity and conciseness we focus on a limited collection of linear growth models only. Other linear growth models, as well as nonlinear models specified with higherorder or piecewise polynomial terms in time, can be specified using straightforward generalizations of the examples that follow.

The first growth model that we consider assumes that in Equation (4), $\mu_t = \alpha + \beta t$ and $s_{it} = \alpha_i + \beta_i t$. This model allows each school to have its own linear growth trajectory through the random intercepts α_i and random slopes β_i, which are assumed to be bivariate normal with means zero, variances σ^2_α and σ^2_β, and covariance $\sigma_{\alpha\beta}$. This model is specified as

```
lme(fixed=yt~year,data=ldat,
    random=~year|schid)
```

Note that because the model is multivariate, the long data `ldat` is necessary.

This model is probably inappropriate because it assumes that deviations of individual student scores from the schoollevel trajectories are homoskedastic and independent over time within students. More realistically, the student deviations ϵ_{kt} are likely to be correlated within students, and potentially have different variances across time. These features can be addressed either through studentspecific random effects, or through a richer model for the within-student covariance structure. To some extent there is an isomorphism between these two pathways; the classic example is that including random student intercepts as part of the mean structure is equivalent to specifying a compound symmetry correlation structure. Which of the two pathways is more appropriate depends on the application and desired inferences, and both are available in `lme`.

As an example of including additional random student effects, we consider random student linear growth so that $\epsilon_{kt} = \gamma_k + \delta_k t + \xi_{kt}$. Like (α_i, β_i) at the school level, (γ_k, δ_k) are assumed to be bivariate normal with mean zero and an unstructured covariance matrix. The random effects at the different levels of nesting are assumed to be independent, and the residual error terms ξ_{kt} are assumed to be independent and homoskedastic. The change to the syntax from the previous model to this models is very slight:

```
lme(fixed=yt~year,data=ldat,
    random=~year|schid/stuid)
```

The `random` statement is analogous to that used in estimating the model in Equation (3); in this case, because there are two random terms at each level, `lme` estimates two (2×2) unstructured covariance matrices in addition to the residual variance.

The other approach to modeling dependence in the student terms ϵ_{kt} is through their covariance structure. `lme` accommodates heteroskedasticity and withinstudent correlation via the `weights` and `correlation` arguments. For the `weights` argument, a wide variety of variance functions that «««< vam.tex can depend on model covariates as described in Pinheiro and Bates Pinheiro and Bates (2000) are available. For the basic models we are ======= can depend on model covariates as described in Pinheiro and Bates (2000) are available. For the basic models we are »»»> 1.2 considering here, the most important feature is to allow the residual terms to have different variances in each year. For the `correlation` argument, all of the common residual correlation structures such as compound symmetry, autoregressive and unrestricted (i.e. general positive definite) are available. The model with random growth terms at the school level and an unstructured covariance matrix for $\boldsymbol{\epsilon}_k = (\epsilon_{k0}, \epsilon_{k1}, \epsilon_{k2})$ shared by all students is specified by

```
lme(fixed=yt~year,data=ldat,
    random=~year|schid,
    weights=varIdent(form=~1|year),
    correlation=corSymm(form = ~1|stuid))
```

A `summary` of the model object will report the estimated residual standard deviation for year 0, along with the ratio of the estimated residual standard deviations from other years to that of year 0. It will also report the estimate of the withinstudent correlation matrix. The functions `varIdent` and `corSymm` combine to specify the unstructured covariance matrix; other combinations of analogous functions can be use to specify a broad array of other covariance structures (Pinheiro and Bates, 2000).

3. Cross classified multivariate models

All of the models in the previous two sections involved only fully nested data structures: observations nested within students, who are nested in higher level units such as schools. Modeling crossed data structures – e.g., accounting for the successive teachers taught by a student – is more challenging. The two most prominent examples of crossclassified models for longitudinal student achievement data are «««< vam.tex that of the Tennessee Value Added Assessment System (TVAAS) Ballou et al. (2004) and the crossclassified model of Raudenbush and Bryk Raudenbush and Bryk (2002). Both models have the ======= that of the Tennessee Value Added Assessment System (TVAAS, Ballou et al., 2004) and the crossclassified model of Raudenbush and Bryk (2002). Both models have the »»»> 1.2 property that the effects of teachers experienced by a student over time are allowed to "layer" additively, so that past teachers affect all future outcomes.

The TVAAS layered model is the most ambitious VAM effort to date. The full model simultaneously examines outcomes on multiple subjects, from multiple cohorts, across five or more years of testing. We consider a simplified version of that model for three years of testing on one subject for one cohort. The model for the outcomes of student k is as follows:

$$\begin{aligned} Y_{k0} &= \mu_0 + \theta_{j0(k)} + \epsilon_{k0} \\ Y_{k1} &= \mu_1 + \theta_{j0(k)} + \theta_{j1(k)} + \epsilon_{k1} \\ Y_{k2} &= \mu_2 + \theta_{j0(k)} + \theta_{j1(k)} + \theta_{j2(k)} + \epsilon_{k2} \end{aligned} \quad (5)$$

The random effects for teachers in year t, θ_{jt}, are assumed to be independent and normally distributed with mean 0 and variance $\sigma^2_{\theta t}$. The within student residual terms $\boldsymbol{\epsilon}_k = (\epsilon_{k0}, \epsilon_{k1}, \epsilon_{k2})$ are assumed to be normally distributed with mean $\mathbf{0}$ and unstructured positive definite covariance matrix $\boldsymbol{\Sigma}_\epsilon$. The residuals are independent across students and are independent of the teacher effects.

Because the model deals directly with the crossclassified structure of the data, special syntax is required to fit the model with `lme`. The main challenges are building the teacher links and specifying the distribution of the teacher random effects. The first step is to augment the long dataframe with binary variables for each teacher with the property that for each record, the new variable takes on the value 1 if the teacher's random effect is part of that score under the model and zero otherwise. That is, if there are n students in the data set, and if there are J_t teachers in year t with $J = J_0 + J_1 + J_2$, then one must augment the long dataframe with a $(3n \times J)$ matrix $\boldsymbol{Z}$ with elements of 1 where linkages occur and 0 otherwise. Note that in most settings, the matrix is large with primarily entries of 0, which is why sparse matrix techniques are likely to be a beneficial line of computational development for these models.

There are several ways to construct this matrix but one straightforward method is to build it from simple assignment matrices of size $(n \times J_t)$ using the teacher identifier fields `y0.tchid`, `y1.tchid` and `y2.tchid` from the wide format dataframe. The following code carries this out for the hypothetical completely observed dataset; modifications are necessary in the presence of missing data:

```
it<-rep(1:n,each=3)
z0<-model.matrix(~y0.tchid-1,data=wdat)[it,]
z1<-model.matrix(~y1.tchid-1,data=wdat)[it,]
z2<-model.matrix(~y2.tchid-1,data=wdat)[it,]
i0<-seq(from=1,to=(3*n-2),by=3)
z1[i0,]<-0
z2[c(i0,i0+1),]<-0
ldat<-data.frame(ldat,cbind(z0,z1,z2))
```

The calls to `model.matrix` result in $(n \times J_t)$ assignment matrices, which are then expanded to duplicated rows via the index `it`. Then, the appropriate rows of `z1` and `z2` are set to zero to indicate that later teachers are not associated with prior scores. The final call augments the long dataframe with the new variables.

The next step is to specify the distribution of the J teacher random effects. Recall that the teacher effects are assumed to be independent and normally distributed, with different variances in each year. That is, the covariance matrix of the joint distribution of all teacher random effects (where teachers are blocked by year) is

$$\mathrm{Var}(\boldsymbol{\theta}) = \begin{bmatrix} \sigma^2_{\theta 0}\boldsymbol{I}_{J_0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \sigma^2_{\theta 1}\boldsymbol{I}_{J_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sigma^2_{\theta 2}\boldsymbol{I}_{J_2} \end{bmatrix} \quad (6)$$

Communicating this structure to `lme` requires a relatively sophisticated `random` argument. To construct this argument it is necessary to make use of **nlme**'s `pdMat` classes, which allow specification of a flexible class of positive definite covariance structures for random effects. The specific classes that are required are `pdIdent`, which refers to matrices of the form $\sigma^2\boldsymbol{I}$, and `pdBlocked`, which creates a block diagonal matrix from its `list` argument. To build the `pdIdent` blocks it is necessary to construct a formula of the form

```
~tchid1 + tchid2 + ... + tchidJ -1
```

for each year; this is used to allow a separate random effect for each teacher. A common way to build such a formula from a vector of character strings is

```
fmla0<-as.formula(paste("~",
       paste(colnames(z0),collapse="+"),"-1"))
```

Letting `fmla1` and `fmla2` refer to the analogous formulas for years 1 and 2, the covariance structure in Equation (6) is specified as:

```
mat<-pdBlocked(list(pdIdent(fmla0),
     pdIdent(fmla1),pdIdent(fmla2)))
```

This covariance structure is then passed to `lme` in the call to fit the layered model:

```
lme(fixed=yt~factor(year)-1,data=ldat,
    random=list(dumid=mat),
    weights=varIdent(form=~1|year),
    correlation=corSymm(form = ~1|dumid/stuid))
```

Because the layered model allows a separate mean for each year, it is necessary to treat `year` as a factor; including the `-1` in the fixed effects specification removes the intercept so that the yearly means are estimated directly. The `random` argument makes use of a trick that is necessary to force `lme` to deal with crossclassified data when it is designed for nested data.

In brief, lme assumes that units are nested in hierarchical levels, and it allows a vector of random coefficients for each higher-level unit that affects the outcomes of all lower-level units nested in that higher-level unit. Because students are not nested in teachers, and thus in principle the random effects for any teacher are eligible to contribute to the responses for any student, it is necessary to impart an artificial nesting structure on the data. This is the role of the dumid variable which takes on the same value for all records in the dataset. Students are trivially nested in dumid, and teachers are assumed to be attributes of dumid that have random coefficients, thus allowing any teacher to affect any student. Finally, as in the previous section, the weights and correlation arguments allow an unstructured residual covariance matrix within students; note that in the correlation argument it is necessary to respect the artificial nesting of students in dumid.

The cross-classified specification of the model affects the form of the model output in one notable way: a summary of the model object will report a separate estimated standard deviation component for each teacher. However, those values will be common across all teachers in the same year, as required by the pdIdent specifications.

The Raudenbush and Bryk cross-classified model uses the same layering of the teacher effects as the TVAAS layered model, but specifies an explicit model for student growth rather than an unstructured student-level covariance matrix. Specifically, the model for the outcomes of student k is

$$\begin{aligned} Y_{k0} &= (\mu+\mu_k)+\theta_{j0(k)}+\epsilon_{k0} \\ Y_{k1} &= (\mu+\mu_k)+(\beta+\beta_k)+\theta_{j0(k)}+\theta_{j1(k)}+\epsilon_{k1} \\ Y_{k2} &= (\mu+\mu_k)+2(\beta+\beta_k) \\ &+ \theta_{j0(k)}+\theta_{j1(k)}+\theta_{j2(k)}+\epsilon_{k2} \end{aligned} \tag{7}$$

Here the residuals ϵ_{kt} are assumed to be independent and homoskedastic both within and across students. The student random intercepts and slopes (μ_k, β_k) have a bivariate normal distribution with mean zero, variances σ^2_μ and σ^2_β and covariance $\sigma_{\mu\beta}$. The distributions of teacher random effects are the same as those assumed by the TVAAS layered model.

In order to specify this model in lme one needs to perform the same steps regarding the teacher random effects as were necessary with the layered model, culminating in the creation of the object mat above. The resulting call to lme is:

```
lme(fixed=yt~year,data=ldat,
    random=list(dumid=mat,stuid=~year))
```

Note that here year is treated as numeric, with the model estimating a marginal linear trend in time. The random statement needs to specify the random effects structure at each of the two levels of nesting: the teacher effects at the degenerate highest-level of nesting have covariance matrix mat, and the random intercepts and slopes for the students have an unrestricted covariance matrix.

Practical Considerations for Cross-Classified Models: Although this article has shown how lme can be used to estimate the most prominent cross-classified models in educational achievement modeling, it is important to bear in mind that it is neither designed nor optimized for such models. Sometimes, particularly with larger cross-classified models, lme will be exceedingly memory intensive and may not converge. To increase the likelihood of convergence in practical settings, particularly with unstructured student-level correlation, it is useful to start the algorithm at the empirical correlation of the data via the value argument of the corClasses constructor functions. Also in some circumstances we have found that replacing the unstructured student correlation matrix with compound symmetry (via corCompSymm) greatly enhances the stability of the estimation. In our experience with actual student achievement data, compound symmetry is often a reasonable assumption and the impact of this restriction on model inferences is generally minimal. Finally, in our experience with lme, the TVAAS layered model is more likely to converge than the Raudenbush and Bryk cross-classified model.

Another practical constraint is that to our knowledge, lme does not allow there to be more than 200 random effects associated with any one unit at any one level of nesting. This implies, for example, that the layered or cross-classified cannot be fit as specified above when the total number of teachers exceeds 200, because teachers are assumed to be attributes of dumid each with a separate random coefficient. A clever way to sidestep this boundary is to make use of additional, "higher-level" degenerate nesting variables. Although all such variables take on only a single value in the dataset, they are still trivially nested, and it is technically possible to specify up to 200 random effects at each of these levels. With three years of cross-classified data in (for example) the TVAAS layered model, this can be used to expand the capabilities of lme from 200 total teachers to 600 via the following syntax, where the additional degenerate nesting variables dumid2 and dumid3 have been appended to ldat:

```
lme(fixed=Y~factor(year)-1,data=ldat,
    random=list(dumid=pdIdent(fmla1),
                dumid2=pdIdent(fmla2),
                dumid3=pdIdent(fmla3)),
    weights=varIdent(form=~1|year),
    correlation=corSymm(form =
    ~1|dumid/dumid2/dumid3/stuid))
```

Although such structuring in principle increases the capabilities lme, our experience is that models with significantly more than 200 teachers often do not convergence.

 ISSN 1609-3631

Extensions and other issues

Incomplete data

The hypothetical examples assumed fully observed data: all student outcomes and all appropriate links of units to higher level units are known. In practical applications this will seldom be true. Students may be missing scores, links to teachers, or both. For the nested univariate models, it is common (though potentially not advisable) to use only complete cases; this is achieved by passing `na.action=na.omit` as an additional argument to `lme`. The multivariate models can use incomplete records, though the specific assumptions and models necessary to do so require careful substantive consideration. Incomplete data also make some of the preprocessing operations required to fit crossclassified models somewhat more tedious; in particular, adjustments to the $\boldsymbol{Z}$ matrix to account for missing scores and/or missing teacher links are required.

Multivariate annual outcomes

In some cases it might be of interest to model simultaneously the outcomes on multiple assessments in a given year, either on the same or different subjects. Generally this would be done only in the context of the multivariate models, where then one must consider appropriate mean and covariance structures for outcomes both contemporaneously and across time. It is beyond the scope of this article to explore the details of such specifications, but for some data and model structures (particularly with nested rather than crossclassified data) it is possible to specify and estimate such models with `lme`.

Covariates

Although the bulk of this article focused on the specifications of the random effects structures of the models, which can be used to estimate student, teacher or school effects, in practical settings it is often of interest to augment the models with covariates at one or more levels of the data. The relationships of such variables to outcomes may be of substantive interest, they may be used to explain variation in random effects, or they may be used to make more normalized comparisons among estimated unitspecific effects. In most applications, such variables are modeled with fixed effects, and thus are simply specified by passing standard R model formulae as the `fixed` argument to `lme` for any of the models specified above. The syntax seamlessly handles variables at any level of the multilevel data structure.

Conclusion

`lme` is an excellent tool for the many applications of longitudinal student achievement modeling in education research. In many research problems involving moderately sized data sets, it is possible to estimate all of the models that are commonly used to make inferences about teacher and school effects as well as the relationships of outcomes with educator practices and characteristics. The current primary limitation is the size of the dataset, particularly with crossclassified models. Although large crossclassified models are outside the current scope of the **nlme** package, theoretical and practical developments are underway to broaden the size and scope of such models that can be estimated (Bates and DebRoy, 2003; Browne et al., 2001).

The examples presented previously can be generalized to handle some additional complications of the data such as team teaching and multiple student cohorts. More complex models involving studentbyteacher interactions and unknown persistence of teacher effects (such as that described in McCaffrey et al. (2004)) are currently beyond the reach of `lme` as well as other standard packages.

Bibliography

D. Ballou, W. Sanders, and P. Wright. Controlling for student background in valueadded assessment of teachers. *Journal of Educational and Behavioral Statistics*, 2004. to appear. 20

D. Bates and S. DebRoy. Linear mixed models and penalized least squares. *Journal of Multivariate Analysis*, 2003. Submitted for publication. 17, 23

W. J. Browne, H. Goldstein, and J. Rasbash. Multiple membership multiple classification (MMMC) models. *Statistical Modelling: An International Journal*, 1(2):103–124, 2001. 17, 23

D. McCaffrey, J. Lockwood, K. D., T. Louis, and L. Hamilton. Models for valueadded modeling of teacher effects. *Journal of Educational and Behavioral Statistics*, 2004. to appear. 17, 19, 23

J. C. Pinheiro and D. M. Bates. *MixedEffects Models in S and S-PLUS*. Statistics and Computing. Springer, 2000. 17, 19, 20

S. Raudenbush and A. Bryk. *Hierarchical Linear Models: Applications and Data Analysis Methods*. Sage Publications, Newbury Park, CA, second edition, 2002. 17, 20, 21

J.R. Lockwood, Daniel F. McCaffrey
The RAND Corporation
lockwood@rand.org, danielm@rand.org

Harold C. Doran
New American Schools
hdoran@nasdc.org

lmeSplines

An R package for fitting smoothing spline terms in LME models

by Rod Ball

Smoothing splines can be formulated in terms of linear mixed models. Corresponding to any smoothing spline term there is a mixed model with a set of random effects, a covariance structure, and a Z-matrix linking the random effects back to observations in the users dataframe. For a smoothing spline model a single variance component is estimated which is inversely proportional to the smoothing parameter for the smoothing spline. The `lmeSplines` package provides functions for generating and manipulating the necessary Z-matrices corresponding to sets of random effects. Using the Choleski decomposition of the covariance matrix, the random effects are transformed to an independent set of random effects, enabling the model to be fitted in `lme` with existing `pdMat` classes. Model predictions can be obtained at the data points and, by interpolation in the Z-matrices, at alternate points using `predict.lme`.

Smoothing splines

Smoothing splines are a parsimonious representation (only one variance parameter is fitted) of a non-parametric curve. Compared with non-linear models, smoothing splines offer a number of advantages:

- Avoiding the need to find a parametric model.
- Avoiding strong model assumptions about the form of the curve.
- Avoiding problems e.g. in longitudinal data where some individuals don't follow the standard curves.
- Can be used to test for smooth departures from a given model.
- A non-linear mixed model can be replaced with a linear model giving more rapid convergence of model fits, and hence enabling more complex and realistic variance structures to be fitted.

Smoothing splines as mixed models

Our package is based on the formulation given in Verbyla *et al* (1999), and extends the capabilities of the NLME package for fitting linear and non-linear mixed models. Readers interested in trying NLME are recommended to read the excellent book (Pinheiro and Bates 2000). The NLME package is substantial, with comprehensive on-line documentation, however it helps to have worked through the examples, and have an overview understanding of the system to fully appreciate its potential.

In the one dimensional case, the smoothing spline for fitting a function of the form $y = g(t) + \epsilon$, is found by maximising the penalised likelihood:

$$\text{penalised likelihood} = \text{log likelihood} - \lambda \int g''(t)^2 dx \tag{1}$$

Restricting to the observed data points, the choice of g from an infinite dimensional space reduces to a finite dimensional problem, which can be expressed as a linear mixed model. The mixed model has the form:

$$y = X_s\beta_s + Z_s u_s + \epsilon \tag{2}$$

where β are fixed effects with intercept and slope coefficients, u_s is a set of random effects with $u_s \sim N(0, G_s\sigma_s^2)$, and $\epsilon \sim N(0, \sigma^2)$. The matrices Q and G_s depend only on the the spacings between the time points (see Verbyla *et al* p. 278), where Q is denoted by Δ), Z_s is given by

$$Z_s = Q(Q^t Q)^{-1}, \tag{3}$$

and the smoothing spline parameter λ is related to the mixed model parameters by

$$\lambda = \frac{\sigma^2}{\sigma_s^2}. \tag{4}$$

We transform to a set of independent random effects u_s' with

$$u_s = Lu_s', \; Z_s' = Z_s L, \tag{5}$$

where L is the Choleski decomposition of G_s, $LL' = G_s$, giving

$$y = X_s\beta_s + Z_s' u_s' + \epsilon \tag{6}$$

with $u_s' \sim N(0, I\sigma_s^2)$.

For further information on the historical context of representing smoothing splines as mixed models and related approaches to smoothing see the discussion (Verbyla *et al* p. 276), and the other references.

Fitting smoothing spline models

The package provides 3 functions: the function `smspline()` calculates the matrix Z_s' in (6), (henceforth referred to as the Z-matrix, or simply Z), with columns corresponding to the unique values of the 'time' covariate, except for the two endpoints. The function `smspline.v()` returns a list containing the matrices X, Q, Z, G_s which can be used for further calculations in the smoothing spline framework. A third function approx.Z is used for transforming the spline basis (columns of the Z-matrix).

The function smspline() is used to calculate the Z-matrix. A smoothing spline is fitted by including a term (or block) of the form pdIdent(~ Z - 1) in the LME random effects structure.

In the first example a smoothing spline is fit to a curve with measurements made on 100 time points. This takes less than a second to fit on a 2 Ghz Linux PC.

Example 1.

```
> library{lmeSplines}
> data(smSplineEx1)
> # variable 'all' for top level grouping
> smSplineEx1$all <- rep(1,nrow(smSplineEx1))
> # setup spline Z-matrix
> smSplineEx1$Zt <- smspline(~ time,
+    data=smSplineEx1)
> fit1s <- lme(y ~ time, data=smSplineEx1,
+    random=list(all=pdIdent(~Zt - 1)))
```

Note:

1. LME has several methods for specifying the covariance structure for random effects which can be confusing to new users. We will use only one: consisting of a named list with each element corresponding to a level of grouping, specified by a formula, such as ~ time, or a 'pdMat' object such as pdIdent(~Zt - 1), or a list of such objects wrapped up with pdBlocked e.g. pdBlocked(list(~time,pdIdent(~Zt - 1))).

2. Where there is no grouping structure, or we wish to use the 'top-level' grouping, consisting of the whole dataset, we introduce the variable all consisting of a vector of ones.

3. The structure smSplineEx1 is a dataframe with 100 rows. We have added the matrix Zt which is a matrix with 100 rows.

```
> str(smSplineEx1)
'data.frame':   100 obs. of  5 variables:
 $ time   : num  1 2 3 4 5 6 7 8 9 10 ...
 $ y      : num  5.80 5.47 4.57 3.65 ...
 $ y.true: num  4.24 4.46 4.68 4.89 ...
 $ all    : num  1 1 1 1 1 1 1 1 1 1 ...
 $ Zt     : num [1:100, 1:98] 1.169 ...
```

Fortunately the R dataframes and model formulae can contain matrix terms. Using ~ Zt in a model formula is equivalent to specifying every column of Zt as a term.

4. The ' -1' in pdIdent(~Zt - 1) stops R from adding an intercept term. Thus there are 98 random effects specified, one for every column of Zt, i.e. one for every unique time point except the ends.

The LME fitted model is summarised as:-

```
> summary(fit1s)
Linear mixed-effects model fit by REML
 Data: smSplineEx1
  AIC BIC logLik
  282 292   -137
Random effects:
 Formula: ~Zt - 1 | all
 Structure: Multiple of an Identity
           Zt1    Zt2    Zt3    Zt4
StdDev: 0.0163 0.0163 0.0163 0.0163
. . .
         Zt97   Zt98 Residual
StdDev: 0.0163 0.0163     0.86

Fixed effects: y ~ time
            Value Std.Error DF t-value
(Intercept)  6.50     0.173 98    37.5
time         0.04     0.003 98    13.0
            p-value
(Intercept)  <.0001
time         <.0001
 Correlation:
     (Intr)
time -0.868

Standardized Within-Group Residuals:
   Min     Q1    Med     Q3    Max
-2.889 -0.581  0.116  0.659  1.784

Number of Observations: 100
Number of Groups: 1
```

We read off the estimate of $\hat{\sigma}_s = 0.0163$ (standard deviation parameters for Zt1, Zt2,...), and the estimate of $\hat{\sigma} = 0.86$ (residual standard deviation), and hence the smoothing parameter is estimated as $\hat{\lambda} = (\hat{\sigma}/\hat{\sigma}_s)^2 = (0.0163/0.86)^2 = 3.6 \times 10^{-4}$ representing quite a smooth curve. (Cf Figure 1).

Comparison with Bsplines and Flexi

We compare the smoothing spline fit to Bspline models with 3 and 5 knots (fit1bs3, fit1bs5 respectively) viz:-

```
> fit1bs3 <- update(fit1s, random=list(all=
+    ~bs(time,3)-1))
> fit1bs5 <- update(fit1s, random=list(all=
+    ~bs(time,5)-1))
> anova(ex1.fit1s,ex1.fit1bs3,ex1.fit1bs5)
        Model df AIC BIC logLik L.Rat Pval
fit1s       1  4 282 292   -137
fit1bs3     2 13 298 332   -136  1.86 0.99
fit1bs5     3 24 316 378   -134  4.54 0.95
```

A plot of the smoothing spline fit is shown in Figure 1, with Bspline fits with 3 and 5 knots are shown for comparison. Output from Flexi, a Bayesian smoother (Upsdell 1994,1996; Wheeler and Upsdell 1997) is also shown. The Flexi and smoothing spline curves appear similar with Flexi giving slightly less of a 'hump' near the end. The Bspline models used many more degrees of freedom, took longer to fit (2 sec and 10 sec for 3 and 5 knot points respectively) and gave poorer fits by either AIC of BIC criteria—note the polynomial type wobbles.

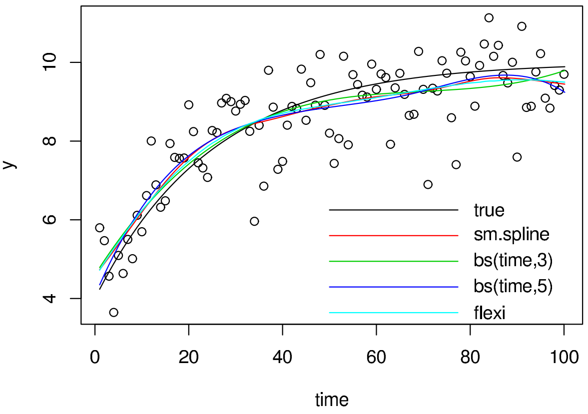

Figure 1: Spline fits. Curves fitted include: the 'true' curve, from which deviations were simulated, a smoothing spline, Bsplines with 3,5 knots, and Flexi, a Bayesian smoother.

Aside: Discussion of mixed model formulation in LME, GENSTAT and SAS

By using the variable 'all' we see that models with no grouping structure are a special case of models with a grouping structure. This makes it possible to fit any models, such as commonly fitted in GENSTAT or SAS, where there is no grouping structure (unless specifically requested using a 'repeated' or 'random' statement in SAS), albeit without taking advantage of the efficiency gains possible from LME's treatment of grouping. Model formulae in LME are often more cumbersome than the corresponding GENSTAT REML or SAS PROC MIXED formulae, because LME assumes a general positive definite symmetric covariance matrix for all effects in a formula, while GENSTAT and SAS assume independent sets of random effects for each term in a formula. Suppose that a,b are factors, and we want random terms for a, b, and the interaction, with different variances for different levels of b. In GENSTAT this would be fitted with:

```
vcomp random = a + b + a.b
vstruct[term=b]factor=b; model=diag
vstruct[term=a.b]factor=a,b; model=id,diag
```

In LME, using the levels of a as groups, this would become

```
random=list(
  all=pdBlocked(list(
    pdIdent(~1),pdIdent(~a-1),pdDiag(~b-1))),
  a = pdDiag(~b-1))
```

LME is still lacking some 'pdMat' classes corresponding to the separable (i.e. tensor product) structures in GENSTAT. In many cases, one of the covariance matrices in a tensor product is the identity, and can be modelled using the factor with the identity structure as a grouping factor, as above. If, however, a in the GENSTAT model had a general symmetric covariance matrix V_a and b had a diagonal matrix D_b, the tensor product covariance matrix $V_a \otimes D_b$ for the interaction between a and b would be fitted in GENSTAT with:

```
vcomp random = a + b + a.b
vstruct a; symm
vstruct b; diag
vstruct a.b; symm,diag
```

which currently has no equivalent in LME.

Of course LME has the major advantage that anyone can write a new pdMat class.

Fitting with alternate sets of time points

The fitted curve in Fig. 1 was obtained using fitted values from the NLME fitted model object viz:-

```
lines(smSplineEx1$time,fitted(fit1s),col=2)
```

If the observed data are irregular or more points are needed, predictions at other points can be obtained in principle from the spline model using the matrices X_s, Q, G_s. We have not implemented these calculations, rather, we use a third function approx.Z() for transforming the spline basis to alternate sets of points (e.g. finer or coarser grids) for modelling and/or prediction. For example to fit the model on a coarser grid:-

```
# fit model on coarser grid
times20 <- seq(1,100,length=20)
Zt20 <- smspline(times20)
smSplineEx1$Zt20 <- approx.Z(Zt20,times20,
    smSplineEx1$time)
fit1s20 <- lme(y ~ time, data=smSplineEx1,
    random=list(all=pdIdent(~Zt20 - 1)))
```

Predictions with alternate sets of time points

For model predictions we need to generate a newdata argument for predict.lme() which contains all the terms in the model including the Z-matrix (here Zt20). This means we need a replacement for

Zt20 with values for each time point in the prediction dataset. This is obtained by calling the function approx.Z which uses linear interpolation in the columns of the Z-matrix.

```
# get model predictions on a finer grid
times200 <- seq(1,100,by=0.5)
pred.df <- data.frame(all=rep(1,
    length(times200)),time=times200)
pred.df$Zt20 <- approx.Z(Zt20,times20,
    times200)
yp20.200 <- predict(fit1s20,newdata=pred.df)
```

Note: Linear interpolation is a good approximation here because the spline basis functions (columns of the Z-matrix, i.e. the matrix Z_s' after the transformation (5)) are approximately piecewise linear.

Models with multiple levels of grouping

More general models can contain multiple smoothing spline terms at different levels of grouping and/or with different time covariates.

This is illustrated in the second example. The Spruce dataset contains size measurements taken over time on 13 different trees from each of 4 different plots.

Example 2.

```
data(Spruce)
Spruce$Zday <- smspline(~days, data=Spruce)
spruce.fit2 <- lme(logSize ~ days,
  data=Spruce, random=list(
    all=pdIdent(~Zday - 1),
    plot=pdBlocked(list(
            ~ days, pdIdent(~Zday - 1))),
    Tree = ~1))
```

Fixed or random? Note the inclusion of intercept and slope terms as *random* effects at the plot level. The smoothing spline model (6) specifies *fixed* effects for the slope and intercept. We recommend that the choice of random or fixed effects should be made prior to consideration of fitting a smoothing spline term—whether these should be fixed or random follows the same logic regardless of whether a smoothing spline term is added. We specify random effects since plot effects come from a population of plots, which gives rise to a population of spline curves.

The pdBlocked construction at the plot level gives a variance structure in two mutually independent blocks. The first block contains two random effects (intercept and slope) for each plot with a general symmetric 2×2 covariance matrix, and the second contains 11 independent random effects for the smoothing spline.

Alternate possible models include adding linear terms at the tree level, or linear and spline terms. However, for these data a single overall spline term gave the best fit.

Much more complex models can be fitted. Recently we have fitted models for the spatial distribution of wood density in tree stems. There were measurements made for each annual ring of each of a number of discs taken from log ends (at approximately 5m intervals) on 2 trees per clone for each of 10 clones, for a total sample size of around 2650. There were four levels of grouping: all/clone/tree/disc. The model contained linear terms, smooth trends represented by pdIdent(~Zr-1), pdIdent(~Zh-1), in ring number and height, and deviations represented by e.g. pdIdent(~factor(ring)-1), at each level, as appropriate, additionally with an AR(1) correlation structure for within group errors. The LME model fitted was:

Example 3.

```
lme(density ~ ring + height, random=list(
  all = pdBlocked(list(
          pdIdent(~Zr-1),
          pdIdent(~factor(ring) -1),
          pdIdent(~Zh-1),
          pdIdent(~factor(height)-1))),
  clone = pdBlocked(list(
          ~ ring + height,
          pdIdent(~factor(ring) -1),
          pdIdent(~Zh-1),
          pdIdent(~factor(height)-1))),
  tree  = pdBlocked(list(
          ~ ring + height,
          pdIdent(~factor(ring) -1),
          pdIdent(~Zh-1),
          pdIdent(~factor(height)-1))),
  disc = pdBlocked(~ ring, pdIdent(~Zr -1)),
  cor=corAR1(form= ~ring))
```

The smoothing spline models fit relatively quickly, but with additional within group correlation structures the model fits take a number of hours. Further details will be reported elsewhere.

References

Ball, R. D. (2003) Experiences with LME, smoothing splines, and application to Pinus radiata wood density. New Zealand Statistical Association Annual Conference, Massey University, Palmerston North.

Green, P. J. and Silverman, B. W. (1994) *Nonparametric Regression and Generalized Linear Models*. Chapman and Hall, London.

Guo, W. (2002) Inference in smoothing spline analysis of variance. *J. Royal Stat. Soc. B* 64(4) 887–898.

Kimeldorf, G.S. and Wahba, G. (1970) A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Ann. Math. Statist.*, 41, 495–502.

Pinheiro, J. C. and Bates, D. M. (2000) *Mixed-Effects Models in S and S-PLUS* Springer-Verlag, New York.

Silverman, B. W. (1985) Some aspects of the spline smoothing approach to non parametric regression curve fitting (with discussion). *J.R. Statist. Soc. B*, 47, 1–52.

Speed, T. P. (1991) Discussion of "That BLUP is a good thing: the estimation of random effects" by G. K. Robinson. *Statist. Sci.*, 6, 42–44.

Upsdell, M. P. (1994) Bayesian smoothers as an extension of nonlinear regression. The New Zealand Statistician, 29, pp.66–81

Upsdell, M. P. (1996) Choosing an Appropriate Covariance Function in Bayesian Smoothing, *Bayesian Statistics 5*, pp.747-756, Oxford University Press, Oxford.

Verbyla, A., Cullis, B. R., Kenward, M. G., and Welham, S. J. (1999) The analysis of designed experiments and longitudinal data by using smoothing splines. Appl. Statist. 48(3) 269–311.

Wahba, G. (1978) Improper priors, spline smoothing and the problem of guarding against model errors in regression. *J.R. Statist. Soc. B*, 40, 364–372.

Wahba, G. (1983) Bayesian "confidence intervals" for the cross-validated smoothing spline. *J.R. Statist. Soc.B* 45, 2133–150.

Wecker, W. P. and Ansley, C. F. (1983) The signal extraction approach to nonlinear regression and spline smoothing. *J. Am. Statist. Ass.*, 78, 81–89.

Wheeler, D. M. and Upsdell, M. P (1997) *Flexi 2.4 reference manual*. New Zealand Pastoral Agriculture Research Institute Ltd, Ruakura, Hamilton, New Zealand.
martin.upsdell@agresearch.co.nz
www.agresearch.co.nz

Rod Ball,
Forest Research, Rotorua, New Zealand
Rod.Ball@forestresearch.co.nz

Debugging Without (Too Many) Tears

New packages: debug and mvbutils

by Mark Bravington

"Man is born unto trouble, as surely as the sparks fly upward"[1]; and writing functions leads surely to bugs, even in R. Although R has a useful debugging facility in `trace` and `browser`, it lacks some of the sophisticated features of debuggers for languages like C. The new package `debug` offers:

- a visible code window with line-numbered code and highlighted execution point;
- the ability to set (conditional) breakpoints in advance, at any line number;
- the opportunity to keep going after errors;
- multiple debugging windows open at once (when one debuggee calls another, or itself);
- full debugging of `on.exit` code;
- the ability to move the execution point around without executing intervening statements;
- direct interpretation of typed-in statements, as if they were in the function itself.

I often use the debugger on other people's code as well as my own: sometimes to see exactly what went wrong after typing something, and sometimes merely to see how another function actually works. And sometimes— just sometimes— because there's a bug!

Although this article is mostly about `debug`, it makes sense to briefly mention another new package, `mvbutils`, which is required by `debug`. Besides its many miscellaneous utilities, `mvbutils` supports:

- hiearchical, searchable project organization, with workspaces switchable inside a single R session;
- function editing (interface to text editors), with multiple simultaneous edits, an "unfrozen" R prompt, and automatic backup;
- function code and plain-text documentation stored in the same R object, and editable in the same file;
- informal plain-text documentation via `help`, and conversion to Rd format;
- nested `sourceing`, and interspersal of R code and data in the same file;
- macro-like functions, executing inside their caller's environment;
- untangling and display of "what calls what" within groups of functions.

Once the packages are installed and loaded, detailed documentation can be obtained via `README.debug()` and `README.mvbutils()`. The code of `debug`— which is written entirely in R itself— reveals quite a bit about the workings of R, and will form the subject of a future Programmer's Niche article. The rest of this article is a basic introduction to usage.

Using the debugger

Consider this function, which is laboriously trying to calculate a factorial:

```
f <- function(i) {
  if (i<0)
    i <- NA
  else
    for(j in (i-1):2)
      i <- i * j
  i
}
```

This doesn't quite work: `f(3)` is rightly 6, but `f(2)` is 4 not 2. To prepare for debugging, type `mtrace(f)`, then `f(2)` to actually start. A window containing the line-numbered code of `f` will appear at the bottom of the screen, looking like a wider version of Figure 1. You may need to resize your R window to see it. In the R command window, you'll see a prompt[2] `D(1)>`. The debugger is awaiting your input.

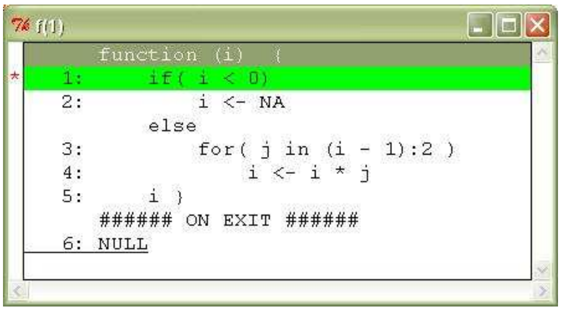

Figure 1: The code window at the start of debugging function `f`

In the R window, press `<ENTER>` to execute the bright green highlighted statement and print the result. The `FALSE` value comes from the conditional test in the `if` statement. The highlight will move

[1] Book of Job, 5:7

[2] The 1 is the current frame number, which matches the number in the title bar of the code window; see "The code window".

into the `else` block, to the `for` loop at line 3. Press <ENTER> again, and the values that the index will loop through are printed. The intention is obviously not to do the loop at all for `i<=2`, but instead you'll see `[1] 1 2` because the "`:`" is going the "wrong" way[3].

If you were really debugging this function, you would probably exit with `qqq()` at this point, and then edit `f`. For now, though, type `go(5)`. The debugger will begin executing statements in `f` continuously, without pausing for input. When line 5 is reached, it will be highlighted in the code window, and the `D(1)>` prompt will return in the R command window. At the prompt, type `i` to see its value. You can type any R statement (including assignments) at the prompt; it will be evaluated in the function's frame, and the result will be printed in the R command window. The pending statement in your function will stay pending, i.e. it won't be executed yet. To see all this, try hand-tweaking the result by typing `i<- 2`. Then type `go()` (without a line number this time) to run through the rest of the function and return the correct value at the normal R prompt.

If you type something at the `D(...)>` prompt that leads to an error, the error message will be printed but the `D(...)>` prompt will return, with the line that led to the error highlighted in the code window. You can carry on as if the error didn't happen; type some commands to patch the problem, or `mtrace` another function inside which the error occurred, or `skip` to another statement (see below), or type `qqq()` to finish debugging.

Special debugging functions

The "big five" are: `mtrace` to prepare or unprepare debugging, `bp` to manage breakpoints, `go` to execute statements continuously, `skip` to move the execution point without actually executing any code, and `qqq` to halt the debugging session and return you to the usual R prompt (with a "nonerror" message). The last three can only be called inside the debugger, the first two from inside or outside. `go` and `qqq` have already featured, and there isn't much more to say about them. As for the others:

- `mtrace(f)` does two things. First, it stores debugging information about `f` in `tracees$f`. The list `tracees` has one element for each function currently `mtraced`; it lives in the `mvb.session.info` environment, shown by `search()`. This environment is created when `mvbutils` is loaded, and lasts until the end of the R session. It is never written to disk, and is used to store session information, like "frame 0" in Splus.

  Second, `mtrace` overwrites `f` with a modified version that calls the debugger. This does not change `f`'s `source` attribute, so you will see no apparent change if you type `f` (but try `body(f)`). However, if you `save` an `mtraced` function and then `load` it into a new R session, you'll see an error when you invoke the function. Before saving, type `mtrace(f,F)` to untrace `f`, or `mtrace.off()` to untrace all `tracees`— or consult `?Save` for a shortcut.

  `mtrace`ing a function should not affect editing operations. However, after the function is read back into R, it will need to be `mtraced` again (unless you use `fixr` in `mvbutils`). All previous breakpoint will be cleared, and a new one will be set at line 1.

- To illustrate `skip`, type `f(-5)` to start debugging again, and press <ENTER>. The `if` test will be `TRUE`, so execution will move to line 2. At this point, you might decide that you want to run the loop anyway. Type `skip(3)` to move the execution point to the start of the `for` loop, and `i` to confirm that the value has not been altered. `skip(1)` will move you *back* to the start of the function, ready to start over. However, note that skipping backwards doesn't undo any changes.

- `bp(5)` will set a breakpoint at line 5 (if the debugger is already running), as shown by the red asterisk that appears to the left of line 5. Type `go()`; the highlight should reappear on line 5. Now clear the line 1 breakpoints by `bp(1,F)`— the upper asterisk will disappear. Press <ENTER> to execute line 5 and finish the function, then `f(2)` again to restart the debugger. This time, when the code window appears, the highlight will be on line 5. Because the line 1 breakpoint was cleared, execution has proceeded continuously until the breakpoint on line 5 triggers. If all breakpoints are cleared, the entire function will run without the code window ever appearing, unless an error occurs (try `f('oops')`). Clearing all breakpoints can be useful when a function will be invoked repeatedly.

  Conditional breakpoints can be set by e.g. `bp(4,i<=j)`. The second argument is an (unquoted) expression which will be evaluated in the function frame whenever line 4 is reached; if the expression evaluates to anything except `FALSE`, the breakpoint will trigger. Such breakpoints are mainly useful inside loops, or when a function is to be called repeatedly. Because everything in a breakpoint expression

[3] One solution to this common loop bug, is to replace the "`:`" with the `%downto%` operator from `mvbutils`: `for( j in (i-1) %downto% 2)`. A far better way to calculate vectorized factorials, though, is based on `cumprod(1 %upto% max(x))[x]`, and there is always `round(exp(lgamma(i+1)))`.

gets evaluated in the function frame, including assignments, conditional breakpoints can be used for "invisible mending" of code. Put the "patch" statement followed by `FALSE` inside braces, as in the following hack to get `f` working OK:

```
bp( 4, { if(i<3) break; FALSE})
```

The `FALSE` at the end prevents the breakpoint from triggering, so that execution continues without pause. `f(2)` and `f(1)` will now work OK, giving possibly the world's worst factorial function.

Breakpoints are usually set inside the debugger, after the code window has appeared. However, they can also be set in advance. Unless you are clearing/setting at line 1, you will need to check the line list, which can be seen in `tracees$f$line.list`.

Debugging `on.exit` code

Code set in `on.exit` statements can be debugged just like normal body code. At the end of the main body code, the code window displays a section marked `##### ON EXIT #####`. Until an `on.exit` statement gets executed, this section just contains a `NULL` statement. When `on.exit` is invoked (either in the function, or from the prompt), the code window will change to show the new exit code. You can set breakpoints in the exit code and step through it just as if it were body code. The exit code will normally be reached when there are no more statements to execute in the body code, or following a `return` statement. Without the debugger, the exit code will also be run after an error; however, with the debugger, errors switch the debugger into step mode, but do not move the execution point.

If you quit the debugger via `qqq()`, the exit code will *not* be executed. To avoid this, `skip` to the start of the exit code, `go` to the line number of the final `NULL`, and then type `qqq()`.

Every statement that is executed or typed in while in the body code, will update the return value (except debugspecific commands such as `go()`). Once the debugger has moved into the exit code, the return value does not change. You can find out the return value by typing `get.retval()`. To forcibly change it, first `skip` back to any line in the body code, then call `return` to set the value and return to the exit code block.

Note that if your function completes (i.e. the debugger leaves it without you calling `qqq()`), the return value will be passed on OK and execution will carry on as normal, even if there were error messages during the execution of your function.

Which statements can be traced?

The basic unit of execution in `debug` is the numbered line. Certain statements get individual lines, while other statements are grouped together in a single line, and therefore can't be stepped into. Grouped statements comprise:

- The test expressions in `if`, `while` and `switch` statements.
- The lists of items to be looped over in `for` statements.
- Calls to "normal" (nonflowcontrol) functions. You can usually `mtrace` the function in question if you want to see what's happening.
- Assignments. If you write code such as `x<- { for( i in 1:5) y<- i/y+y/i; y}`, you're on your own!

The code window

In the present version of `debug`, all you can do in the code window is look at your code. You can scroll up or down and select lines, but this has no effect on debugging. Several code windows can be open at once, not necessarily for different functions (try calling `f` from within `f`), and can be distinguished by the function name and frame number in the title. The window whose number matches the `D(...)>` prompt is currently "active". Some aspects of window position and appearance can be controlled via `options`; see the package documentation.

Strategies for debugging

There are no rules for how to go about debugging, but in most cases I begin by following one of two routes.

When faced with an actual error message, I first call `traceback()` to see where the error occurred; often, this is not in my own functions, even if one of them is ultimately responsible. I then call `mtrace` either on my function, or on the highestnumbered function from `traceback` for which the cause of error is mysterious. Next, I simply retype whatever gave the error, type `go()` when the code window appears, and wait for the crash. The debugger highlights the error line, and I can inspect variables. If it's still not obvious why the crash occurred, I `mtrace` whichever function is invoked by the error line— say `glm`.. Then I press `<ENTER>` to take me into `glm`, and either examine the arguments, or type `go()` again to take me to the error line in `glm`, etc. etc.

If there's no error but `f` is giving strange results, then I `mtrace` and invoke `f`, and step through the code with `<ENTER>`, watching intermediate results and printing any variables I'm curious about. If there is an unproblematic block of code at the start of f, then it's faster to use go(N) (or bp(N) followed by go()) to get to line number N after the block; sometimes N will be the first line of the exit code. Once I understand roughly where the problem is—as shown by a variable taking on an unexpected value—I will either qqq() or patch things up, by setting variables directly from the keyboard and/or using skip, just to check that the next few statements *after* the bug are OK.

The fancier features, such as conditional breakpoints and bp(1,F), are used less, but can be invaluable timesavers where loops are involved.

Speed issues

Functions do run slower when mtraced, though this is only likely to be a problem if a lengthy loop has to execute before reaching the interesting code. You can speed things up by wrapping innocuous code near the start in an evalq({...}) call. For instance, if you mtrace and run this function

```
slowcrash <- function( x) {
  for( i in 1:(10^5)) x <- x+1
  crash <- x + "oops!"
}
```

then you'll be waiting a long time before go arrives at the accident scene. Instead, try mtraceing and running this:

```
fastcrash <- function( x) {
  evalq({ for( i in 1:(10^5)) x <- x+1 })
  crash <- x + "oops!"
}
```

If an error occurs in a complex statement inside a loop, but only after a large number of loop iterations have completed, you can try a different trick. Put the loop contents into an mlocal "macro" function (see mvbutils) called e.g. loopee; replace the guts of the loop with a call to loopee(); make sure loopee is *untraced*; invoke f; and type go(). When the debugger stops at the offending iteration, on the loopee() line, call mtrace(loopee) and press <ENTER> to step into loopee.

Omissions and limitations

1. try(...) statements work fine, but can't yet be stepped into (it's on the to-do list). Fancy error handling via tryCatch etc. isn't explicitly handled, and I don't know how the debugger will respond.

2. The debugger endeavours to invisibly replace certain commands with "debug-friendly" equivalents, both in function code and in keyboard input. It is possible to defeat this with really strange code such as eval(parse(text='next')). If you write that sort of code, you'll get what you deserve...

3. A few key base functions (and most functions in the debugger) can't be debugged directly. A workaround is to copy them to a new variable and mtrace that.

4. There is no "watch window" and no "global expression-triggered breakpoint" facility. Both could be added (the Splus version of debug did have a watch window at one point), but would take me some time.

5. Loss of breakpoints after editing is annoying; the Splus version tries harder to keep them, and I might implement the same for R.

6. The code window is very unsophisticated, reflecting my TCL/TK abilities; mark-and-copy would be nice, for example.

7. S4 methods aren't handled (though S3 methods are, even hidden ones); I have no plans to write my own S4 methods, and hope not to have to debug anyone else's!

Feedback on debug and mvbutils is welcome. I will fix bugs, and will try to add feasible enhancements as time permits.

Mark Bravington
CSIRO Mathematics and Information Sciences
Hobart, Tasmania, Australia
mark.bravington@csiro.au

The R2HTML Package

Formatting HTML output on the fly or by using a template scheme

By Eric Lecoutre

Statistics is not only theory and methodology, but also computing and communication. Applied statisticians are aware that they have to pay particular attention to the last step of an analysis: the report. A very elegant way to handle the final report with R is to use the wonderful Sweave system (Leisch, 2002a) in package **tools**: not only does it allow professional quality reports by using LaTeX, but it also stores the code that is used for the report within the document, which is very useful when returning to the analysis later on. This solution, however, is not always applicable, as the user may not be familiar with LaTeX or may need another format to communicate with a client, who, in many cases, will expect a report that he can edit or to which he can add some details. RTF format is ideal for this type of communication, as it can be opened on many platforms and allows some formatting enhancements (bold, tables, . . .). Nevertheless, it is not easy to produce and does not enable the user to embed graphs. Another universal format which can achieve our goal is HTML: it is lightweight, readable on almost all platforms, editable, and allows graphs. Moreover, it can easily be exported to other formats.

This document describes the **R2HTML** package which provides some support for writing formatted HTML output. Although some knowledge of HTML is desirable for those who wish to customize the output, the package can be used successfully without this background. We will create several web pages, which the reader can find at the following address: http://www.stat.ucl.ac.be/R2HTML/.

Introduction to HTML and the R2HTML package

According to the W3 Consortium, HTML is the lingua franca for publishing hypertext on the World Wide Web. It is a nonproprietary format based upon SGML, and can be created and processed by a wide range of tools, from simple plain text editors to sophisticated WYSIWYG authoring tools. HTML uses soealled tags to structure text into headings, paragraphs, lists, hypertext links and to apply a format. For example the tag is used to start bold text, and the tag to stop bold text. The tag with the slash (/) is known as the closing tag. Many opening tags need to be followed by a closing tag, but not all of them do.

Consequently, the only required knowledge in order to write basic HTML documents is the list of existing tags and their functionality. By way of illustration, here is the structure of a (rather basic) HTML document:

```
<html>
<h1>My first HTML page </h1>
<p>This is some basic text with a
   <b>bold</b> word.</p>
<p>It uses h1, and p tags which allows to create
   a title and to define a paragraph</p>
</html>
```

Now, we have a very easy way to create our first webpage from R: simply using the cat function to write text into an external file. In the following example, consider how we call the cat function severall times having set the append argument to TRUE in order to add information to the page.

```
> htmlfile = file.path(tempdir(),
+     "page1.html")
> cat("<html><h1>My first HTML page from R</h1>",
+     file = htmlfile)
> cat("\n<br>Hello Web World!",
+     append = TRUE, file = htmlfile)
> cat("\n</html>", append = TRUE,
+     file = htmlfile)
```

The package **R2HTML** contains a collection of wrapper fynctions that call the cat function to write HTML codes. The main functions are:

- HTML() Main generic function with methods for all classic classes (matrix, lm, summary.xxx, . . .).
- HTMLbr() Inserts a
 HTML tag, which requires a new line to be started.
- HTMLhr() Inserts an <hr> HTML tag, a horizontal rule to separate pieces of text.
- HTMLInsertGraph() Inserts an <img> HTML tag to add an existing graph to the report. The graph should have been created before in a suitable web format such as GIF, JPEG or PNG.

Basically, the **R2HTML** package contains a generic HTML() function that behaves like the internal cat function. Common arguments are append and file, the default file is set by the hidden variable .HTML.file:

```
> .HTML.file = file.path(tempdir(),
+     "page2.html")
> HTML(as.title("Title of my report"),
+     append = FALSE)
> HTMLhr()
> HTML("3 dimensions identity matrix")
> HTML(diag(3))
```

Generating HTML output on the fly

The first way to use the **R2HTML** package is to generate automatic HTML output during an interactive session. This is especially convenient for teaching, as students can then keep a log of the commands they asked for and the corresponding outputs, with graphs incorporated. For a dynamic session to be successful, two commands have to be processed:

- `HTMLStart()`
- `HTMLStop()`

Here is a sketch of the way those commands work. When calling `HTMLStart()`, several actions are performed:

- Three HTML files are written into the temporary directory of the session (or in the directory specified in an option). The main file 'index.html' is linked to the other two, by incorporating them within HTML frames. This makes it possible to have the commands on the left of the screen and the corresponding outputs on the right.
- A new environment, called `HTMLenv`, is created, where some internal variables are stored. Those variables make it possible to store the path of the output files, and to know which action has been performed with the latest submitted command.
- A new `fix` function is assigned to the global environment, masking the internal one. When calling the "new" `fix`, a boolean is set to `TRUE` in the `HTMLenv` environment, so that we know that the latest action was to edit a function.
- `addTaskCallback` is called, adding a task to each submitted command. This task, handled by the function `ToHTML` (not visible to the user) is the core of the process, as it exports the last manipulated object. This function also tests whether a stored boolean indicates that a function has been edited and, if so, exports the edited function. When doing so, a new file is created, so that at the end of the process, one can keep track of all the versions of the function at the different stages of the work.
- Finally, as a side effect, the prompt is changed to `HTML>` so as a signal that outputs are currently being redirected.

From this moment on, every command is treated twice: first it is evaluated and then the result goes through the `ToHTML` function which writes it into the HTML output.

As there is no convenient way to know when a graph has been performed (or modified, think of `lines`, `points`, `...`) and as it is not desirable to export every graph, the user has to explicitly ask for the insertion of the current active graph to the output, by calling the `HTMLplot()` function.

When necessary, a call to the `HTMLStop()` function stops the process and removes all the temporary variables created.

The following example only works in an interactive R session:

```
> HTMLStart(filename = "dynamic",
+     echo = TRUE)

 *** Output redirected to directory:  C:\tmp
 *** Use HTMLStop() to end redirection.[1] TRUE

HTML> sqrt(pi)
[1] 1.772454
HTML> x = rnorm(10)
HTML> x^2
 [1] 2.91248574 0.21033662
 [3] 0.16120327 1.56429808
 [5] 0.02863139 3.47605227
 [7] 1.36348399 0.30152315
 [9] 0.73402896 0.77886722
HTML> myfunction = function(x)
+  return(summary(x))
### try to fix the function
HTML> myfunction(x)
   Min. 1st Qu.  Median    Mean
-1.7070 -0.3017  0.6291  0.3878
3rd Qu.    Max.
 1.0960  1.8640
HTML> plot(x)
HTML> HTMLplot()
[1] TRUE
HTML> HTMLStop()
[1] "C:\\.../dynamic_main.html"
```

Creating personalized reports

A simple analysis

For anyone who knows the basics of HTML, the **R2HTML** package offers all the necessary material to develop fast routines in order to create one's own reports. But even users without knowledge of HTML can easily create such reports. What we propose here is a socalled template approach. Let us imagine that we have to perform a daily analysis whose output consists of some summary tables and graphs.

First, we collect in a list all the objects necessary for writing the report. An easy way to do so is to create a user function `MyAnalysis` that returns such a list. In addition, we assign a userdefined class to this object.

```
MyAnalysis = function(data) {
  table1 = summary(data[,1])
  table2 = mean(data[, 2])
  dataforgraph1 = data[,1]
  output = list(tables =
```

```
    list(t1 = table1, t2 = table2),
    graphs = list(d1 = dataforgraph1))
  class(output) = "MyAnalysisClass"
  return(output)
}
```

We then provide a new HTML function, based on the structure of our output object and corresponding to its class:

```
HTML.MyAnalysisClass = function(x,
 file = "report.html", append = TRUE,
 directory = getwd(), ...) {
 file = file.path(directory, file)
 cat("\n", file = file,
     append = append)
 HTML.title("Table 1: summary for
     first variable",file = file)
 HTML(x$tables$t1, file = file)
 HTML.title("Second variable",
     file = file)
 HTML(paste("Mean for second",
  "variable is: ",
   round(x$tables$t2,3),
   sep = ""),file = file)
 HTMLhr(file = file)
 png(file.path(directory,
     "graph1.png"))
 hist(x$graphs$d1,
  main = "Histogram for 1st variable")
 dev.off()
 HTMLInsertGraph("graph1.png",
     Caption = "Graph 1 - Histogram",
     file = file)
 cat(paste("Report written: ",
     file, sep = ""))
}
```

If we want to write the report, we simply have to do the following:

```
> data = matrix(rnorm(100), ncol = 2)
> out = MyAnalysis(data)
> setwd(tempdir())
> HTML(out, file = "page3.html")
Report written: C:/.../page3.html
```

The advantage of this approach is that we store all the raw material of the analysis within an object, and that we dissociate it from the process that creates the report. If we keep all our objects, it is easy to modify the `HTML.MyAnalysisClass` function and generate all the reports again.

A template scheme to complete the report

What we wrote before is not a real HTML file, as it does not even contain standard headers `<html>...</html>`. As we see it, there are two differents ways to handle this, each with its pros and cons. For this personalization, it is indispensable to have some knowledge of HTML.

First, we could have a pure R approach, by adding two functions to our report, such as:

```
MyReportBegin = function(file = "report.html",
    title = "My Report Title") {
    cat(paste("<html><head><title>",
        title, "</title></head>",
        "<body bgcolor=#D0D0D0>",
        "<img=logo.gif>", sep = ""),
        file = file, append = FALSE)
}
MyReportEnd = function(file = "report.html") {
    cat("<hr size=1></body></html>",
        file = file, append = TRUE)
}
MyReport = function(x, file = "report.html") {
    MyReportBegin(file)
    HTML(x, file = file)
    MyReportEnd(file)
}
```

Then, instead of calling the HTML function directly, we would consider it as an internal function and, instead, call the `MyReport` function.

```
> out = MyAnalysis(data)
> MyReport(out, file = "page4.html")
Report written: C:/.../page4.html
```

The advantage is that we can even personalize the header and the footer of our report on the basis of some R variables such as the date, the name of the data or anything else.

If we do not need to go further than that and only need hard coded content, we can build the report on the basis of two existing files, `header.html` and `footer.html`, which can be modified to suit our needs. To work properly, the following piece of code supposes that those two files do exist in the working directory:

```
MyReport = function(x, file = "report.html",
    headerfile = "header.html",
    bottomfile = "footer.html") {
    header = readLines(headerfile)
    cat(paste(header, collapse = "\n"),
        file = file, append = FALSE)
    HTML(x, file = file, append = TRUE)
    bottom = readLines(bottomfile)
    cat(paste(bottom, collapse = "\n"),
        file = file, append = TRUE)
}
```

Going one step further with CSS

Cascading Style Sheets (CSS) compensate for some of the deficiencies of HTML language. CSS add to each standard HTML element its own style, which is defined in an external file. Thus, when the inhouse stylebook of the report is changed, one need only modify the definition of the classes in a single place to change the look of all the reports -past or future - that rely on the defined classes.

The use of cascading style sheets makes it possible to:

- give a homogeneous look to all generated reports
- change the look of a bunch of reports at one time
- produce lighter reports, as formatting instructions are kept separate
- download and view reports more rapidly

for details about CSS see http://www.w3.org/Style/CSS/.

All functions of package **R2HTML** rely on CSS and a given sample CSS file, R2HTML.CSS. This file is used by HTMLStart. In order to work properly, the CSS file must located in the same directory as the report and one simply has to add the following line to it <link rel=stylesheet type=text/css href=R2HTML.css>. This job is performed by the HTMLCSS() function. It is a good idea to systematically start a report with this function, as CSS files are very powerful. So, in its last version, our reporting function yields:

```
MyReport = function(x, file = "report.html",
    CSSfile = "R2HTML") {
    MyReportBegin(file)
    HTMLCSS(file = file, CSSfile = CSSfile)
    HTML(x, file = file)
    MyReportEnd(file)
}
```

Summary

The **R2HTML** package provides functions to export many base R objects to HTML. In this paper we have described a simple mechanism to use these functions in order to write HTML reports for statistical analyses performed with R. The mechanism is flexible and allows customizations in many ways, mainly by using a template approach (separating the body of the report from the wrapper, i.e., header and footer) and by using an external CSS file.

Bibliography

F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physika Verlag, Heidelberg, Germany, 2002a. ISBN 3-7908-1517-9.

33

Eric Lecoutre
Institut de statistique, UCL, Belgium
lecoutre@stat.ucl.ac.be

R Help Desk

Package Management

Uwe Ligges

Preface

A huge number of packages dedicated to wide range of applications is available for R. Writing this article, there are more than 300 packages: base and recommended packages, CRAN[1] packages, packages from the Omegahat[2] and BioConductor[3] projects, as well as a couple of other packages. This shows that package management is really important. A recent summary of existing packages is published in the R FAQ by Hornik (2003).

Convenient tools for developing, checking, installing, updating and removing packages are available in R. Therefore, users and developers can easily handle packages. This is one of the main reasons for R's rapid growth.

In this article, the procedures to install, update, and manage packages in (more than) one library are described. In particular, this article summarizes and extends the existing documentation on package management.

Documentation on package management

Package management is described in several documents, because handling packages is fundamental for users of R. The "R Installation and Administration" manual (R Core, 2003a) is the main resource for users who want to install packages. Among other topics it explains the terminology. In particular, it is important to distinguish between a *package* and a *library*:

> "A package is loaded from a library by the function `library()`. Thus a library is a directory containing installed packages; ..."

"The R FAQ" (Hornik, 2003) devotes a section to package management. The complementary "R for Windows FAQ" (Ripley, 2003) explains a couple of specific Windows issues related to package management. For users of the Macintosh, the "RAqua FAQ" (Iacus, 2003) might be extended by a corresponding section in a future versions.

The manual "Writing R Extensions" (R Core, 2003b) is devoted to the development of packages, hence of minor interest for the user who tries to install or update a package.

[1] http://cran.r-project.org
[2] http://www.omegahat.org
[3] http://www.bioconductor.org

Libraries

If you have installed a released version of R, the base and recommended packages can be found in the main library: `R_HOME/library`, where `R_HOME` denotes the path to your version of R, e.g. `/usr/local/lib/R` or `c:\Programs\rw1081`.

By default, further packages will be installed into `R_HOME/library` as well.

It may be sensible to have more than one library, depending on the purpose of the system R has been installed on.

Let's consider the following situation: R has been installed on a server, and the regular user does not have any write access to R's main library. There is no need to ask the administrator to install a package from CRAN. The user can easily install the package into another library to which he has write access, e.g.: `/home/user/myRstuff/library`. Moreover, it might be convenient to have a third library containing packages the user is maintaining himself, e.g.: `/home/user/myRstuff/mylibrary`.

In the example given above, Unix file paths have been used, but the idea is applicable to arbitrary operating systems. As an example, consider our department's Windows system: R is installed on a network share, say `s:\R` (no write access for the regular user). If a user has write access to, say, `d:\user`, then `d:\user\myRstuff\library` might be a good choice for a library containing private packages.

Since R does not know about any further libraries than the default one, the environment variable `R_LIBS` must be set appropriately. This can be done in the usual way for setting environment variables on your operating system, or even better by specifying it in one of the environment files R processes on its startup, see `?Startup` for details. For example, in order to tell R that a second library `/home/user/myRstuff/mylibrary` is available, you might want to add the line

```
R_LIBS=/home/user/myRstuff/mylibrary
```

to the file '.Renviron' (see `?Startup`). More libraries can be specified as a semicolon separated list.

During R's startup, the library search path is initialized by calling the function `.libPaths()` on the directories defined in `R_LIBS` (the main library is always included). If `.libPaths()` is called without any arguments, the current library search path is returned.

Source vs. binary packages

Packages may be distributed in source or binary form.

Source packages are platform independent, i.e. independent of hardware (CPU, ...) and operating system, as long as the package author does not implement platform specific details — and, of course, as long as R is available on the particular platform. Installing source packages requires a couple of tools to be installed (e.g. Perl, and depending on the package: C compiler, Fortran compiler, ...). In order to provide packages to a large group of users, developers must submit their packages in source form to CRAN[4]

For users working on Unix-like systems (including Linux, Solaris, ...), it is quite easy to install packages from source, because their systems generally have (almost) all of the required tools installed.

Binary packages are platform specific. They may also depend on the R version — in particular, if compiled code has been linked against R. To install a binary package, no special tools are required, because the shared object files (as known as DLL[5] under Windows), help pages (HTML, text, ...), etc. already have been precompiled in a binary package.

In order to make package installation as convenient as possible for the user, binary versions of CRAN packages are provided for some platforms (in particular these are currently MacOS X, SuSE Linux, and Windows) and the most recent versions of R. For example, binary versions of packages for Windows regularly appear within two days after the corresponding source package on CRAN — given there is no special dependence or need for manual configuration.

By convention, source packages regularly have the file extension '.tar.gz', binary packages for Windows '.zip', and those for Linux '.deb' or '.rpm'. Both extensions '.tar.gz' and '.zip' indicate archives files (an archive contains files in compressed form) packed by different tools. Hence, it's a fast but also unsafe way to decide by looking at file extensions: Omegahat source packages still end in '.zip', MacOS X binary packages end in '.tar.gz'. You can always check whether a package contains folders `help` and `html`. If it does, it is almost certainly a binary package.

Installing and updating source packages

If you have more than one library, you need to specify the library to/in/from which a package is to be installed, updated, removed, etc. The syntax to install source packages on Unix using the command line is

```
$ R CMD INSTALL -l /path/to/library package
```

The part used to specify the library, `-l /path/to/library`, can be omitted. In that case the first library in the environment variable `R_LIBS` is used if set, otherwise the main library. Note that file '.Renviron' is not read by `R CMD`. A Section describing the syntax for Windows can be found below.

For example, a source package `mypackage_0.0-1.tar.gz` can be installed to the library `/home/user/myRstuff/mylibrary` using the command

```
$ R CMD INSTALL -l /home/user/myRstuff/mylibrary
    mypackage_0.0-1.tar.gz
```

In most cases an alternative approach is much more convenient: Packages can be downloaded and installed from within R using the function `install.packages()`. In order to install a package **package** from CRAN into the library `/path/to/library` from an appropriate CRAN mirror[6] (which might not be the US mirror given below), the call looks like:

```
R> options(CRAN="http://cran.us.r-project.org/")
R> install.packages("package",
R+     lib = "/path/to/library")
```

Analogously to the command line version, the specification of a library may be omitted when the package is to be installed into the first library of the library search path. More specifically, `install.packages()` looks for available source packages on CRAN, downloads the latest version of the requested source package, and installs it via `R CMD INSTALL`.

The function `update.packages()` helps to keep packages on a system up to date. According to R Core (2003a), `update.packages()`

> "... downloads the list of available packages and their current versions, compares it with those installed and offers to fetch and install any that have later versions on CRAN."

Note that, instead of the current package management tools, `packageStatus()` (see its help page `?packageStatus` for details) is intended to become the default package manager for future versions of R. Both `packageStatus()` and the package **reposTools** from the BioConductor project are expected to become great tools for modern package management in R.

On Windows, `install.packages()` and `update.packages()` are used to install binary versions of packages rather than source packages (see below).

See R Core (2003a) for an explanation how to remove packages.

[4]Note that submissions of any binary packages to CRAN will not be accepted.
[5]dynamic link libraries
[6]For a list of mirrors see http://cran.r-project.org/mirrors.html

Package management on the Macintosh

Recent versions of R only support MacOS X. On this operating system's command line, R behaves as under Unix.

For users of RAqua, the GUI has menus for the installation and update of source and binary packages. Also, there are separate menus for CRAN packages and packages from the BioConductor project, as well as for packages from local directories.

Functions like `install.packages()` for managing source packages behave as on Unix, but there are also functions for managing binary packages available in RAqua. Instead of ending on `.packages`, names of these functions end on `.binaries`. Hence, `install.binaries()` installs binary packages, etc. A corresponding binary repository for RAqua is available on CRAN at `yourCRANmirror/bin/macosx/MajVer/` where MajVer is, e.g., '1.8' for R-1.8.1.

Package management on Windows

Package management on Windows is special in a way: On the typical Windows system many of the required tools to install packages from source are missing. Moreover, the operating system's command shell is quite different from typical Unix command shells.

Therefore, R's command line tools are slightly different on Windows. For example, the command to install the source package called **mypackage** in file 'mypackage_0.0-1.tar.gz' to the library path `c:\myRstuff\mylibrary` using the command line is:

```
c:\> Rcmd INSTALL -l c:\myRstuff\mylibrary
       mypackage_0.0-1.tar.gz
```

How to collect the required tools and how to set up the system in order to be able to install packages from source is described in the file `R_HOME/src/gnuwin32/readme.packages` within the R sources. It is highly recommended to read that file very carefully line by line, because almost every line is important to get `Rcmd INSTALL` working. Those who have already compiled their version of R from source also have set up their system and tools correctly to install source packages.

The R functions `install.packages()`, `update.packages()` and friends work differently on Windows than on Unix (with the same syntax, though). On Windows, these functions look at the list of *binary* packages (rather than of source packages) and download the latest version of the requested binary package.

The repository in which these functions are looking for binary packages is located at `yourCRANmirror/bin/windows/contrib/MajVer/` where MajVer is, e.g., '1.8' for R-1.8.1. The ReadMe files in these directories (and also the one in `yourCRANmirror/bin/windows/contrib/ReadMe`) contain important information on availability of packages and how special circumstances (e.g. what happens when packages do not pass the quality control, `Rcmd CHECK`) are handled.

If you are using R's GUI ('RGui.exe') on Windows, you will find a menu called "Packages" that provides a GUI interface to the functions `install.packages()`, `update.packages()` and `library()`. Besides the CRAN repository, the BioConductor repository is also accessible via the menu, as well as the installation of local files.

Note that it is not possible to install to another library than that one in the first place of the library search path (`.libPaths()[1]`) using the menu. Consider using `install.packages()` directly in order to install to a different library. For example, in order to install a binary version of the package **mypackage**, available as a local zip file `c:\myRstuff\mypackage_0.0-1.zip`, into the library `c:\myRstuff\mylibrary`, we can use

```
R> install.packages(
R+   "c:/myRstuff/mypackage_0.0-1.zip",
R+   lib = "c:/myRstuff/mylibrary", CRAN = NULL)
```

Note that you have to escape the backslash, or use normal slashes as above, to specify a path within R for Windows.

Bibliography

Hornik, K. (2003): *The R FAQ*, Version 1.8-28. http://www.ci.tuwien.ac.at/~hornik/R/ ISBN 3-900051-01-1. 37

Iacus, S. (2003): *RAqua FAQ/DOC*, Version for R 1.8.x. http://cran.r-project.org/bin/macosx/RAqua-FAQ.html. 37

R Development Core Team (2003a): *R Installation and Administration*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-900051-02-X. 37, 38

R Development Core Team (2003b): *Writing R Extensions*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-900051-04-6. 37

Ripley, B.D. (2003): *R for Windows FAQ*, Version for rw1081. http://cran.r-project.org/bin/windows/rw-FAQ.html. 37

Uwe Ligges
Fachbereich Statistik, Universität Dortmund, Germany
ligges@statistik.uni-dortmund.de

Programmer's Niche

Little Bits of String

by Thomas Lumley

Character strings occupy an unusual position in R. They are treated as atomic types — little bits of string — by all the subscripting functions. On the other hand, in most cases where the programmer really is treating strings as atomic objects it would be better to use factors instead.

The decision not to treat strings as vectors of characters makes sense when you consider what sorts of operations we do on strings. The vectorised operations in R mostly deal with operating on the same entry in multiple vectors. With two strings, the 'same character' is more likely to be defined relative to patterns in the string rather than by counting characters from the beginning. The most useful operations on strings are not elementwise operations on their characters, but pattern matching and replacement operations. In the Unix world these pattern matching operations are described by *regular expressions*. R provides two implementations of regular expressions, which allow quite complex text manipulation, though for efficient processing of very large quantities of text a specialised tool such as Perl is better. I will discuss only some features of one of the R implementations. A more complete description of regular expressions in R is given by `help(regex)` and the references it cites.

A regular expression is a small program that matches one or more strings. For example, the regular expression `a regular expression` matches the single string "a regular expression". R provides functions `grep`, `sub`, `gsub`, `regexpr`, and `strsplit` that take a regular expression and a string and look for any of the possible outputs of the regular expression in the string. This can be useful even with very simple regular expressions, for example, `apropos("file")` returns all the currently defined objects that have the string `"file"` in their names.

As the example shows, most characters in a regular expression just match themselves as output. The power of regular expressions comes from the special characters that do more than this. The first two are `^` and `$`, which represent the beginning and end of the string respectively. The regular expression `^print` matches "`print`" at the beginning of the string, which is found in the strings `"print.data.frame"` and `"printer.type"` but not in `"dev.print"`. In simpler times the function `methods`, which lists the methods available for a given generic function, could just use a regular expression of this sort to find all functions whose name began with the name of the generic; things are much more complicated now.

In addition to representing a single character, we can represent a class of characters. At the extreme, the character `.` matches any character, but more restricted patterns are often more useful. For example, the regular expression `[[:alpha:]]` matches any single uppercase or lowercase letter, `[[:digit:]]` matches any digit, and `[[:space:]]` matches any of the whitespace characters. Other square-bracket expressions specify other character classes. You may see `[A-Z]` for specifying ranges of characters (between A and Z in this example), but this should be used with care (for example, in Danish the letters Æ, Ø, and Å are not between A and Z).

The special characters present a problem when you need an actual `.` or `$`, say, in a string. To prevent their being interpreted as special they must be preceded by a backslash and, given the familiar problems with C strings, this means typing a double backslash. To match ".sgml" at the end of a string (identifying S help files) we need the regular expression `\\.sgml$` (as a convenience, the R functions that accept regular expressions mostly accept an argument `fixed=TRUE` to specify that all special characters should be interpreted as ordinary characters).

The real power of regular expressions comes from specifying repetitions of characters or sets of characters. The qualifiers `?`, `+`, and `*` specify that the previous character should be repeated at most once, at least once, or any number of times. For example, `^[[:space:]]*` matches any amount of empty space at the start of a string, so

```
sub("^[[:space:]]*", "", string)
```

will remove any leading whitespace characters from the strings (that is, replace it with nothing). Parentheses create groups that can be repeated together, so that `(".+",[[:space:]]*)*` matches any number of terms in quotation marks, separated by commas and optionally by space. Even trickier things are possible with parentheses:

```
([[:alpha:]]+)[[:space:]]\\1
```

matches repeated words. The backreference `\\1` means 'the string that the first set of parentheses matched'. We could remove repeated words from a string with

```
> gsub("([[:alpha:]]+)[[:space:]]*\\1", "\\1",
          "the the end is is nigh")
[1] "the end is nigh"
```

The Sweave tools (Leisch, 2002) provide a nice example of the integration of regular expressions into R programs. The underlying R code can process input files in multiple formats, with the necessary descriptions of the formats given by regular expressions (*e.g.* `tools::SweaveSyntaxNoweb`[1])

[1] in future versions of R this will be `utils::SweaveSyntaxNoweb`

Despite this power, the main use I make of regular expressions is in simple script processing. For example, in porting a package written for S-PLUS recently I wrote a function to find all the SGML documentation files in the package, strip the version control comments from the beginning of each file, and feed them to `R CMD Sd2Rd` to make Rd files. This could have been done in Perl or probably with a simple shell script, but it seemed easier just to use R.

An interesting exercise for the reader: Creating a namespace for an existing package that uses S3 methods requires registering the methods in the `NAMESPACE` file. That is, for a function such as `print.coxph`, the `NAMESPACE` file should contain `S3method(print, coxph)`. How would you write a function that looked for methods for a list of common generic functions and created the necessary `S3method` calls? You would probably need to use the functions either `strsplit` and `paste` or `regexpr` and `substr`, and would need to remember that the generic name need not be a single word (`is.na.Surv` springs to mind). Some hand editing would still be required, for example to determine whether `t.test.formula` was a method for `t` or `t.test` or neither.

References

Leisch, F. (2002) Sweave, Part I: Mixing R and LaTeX. *R News* 2(3): 28–31.

Thomas Lumley
Biostatistics, University of Washington
tlumley@u.washington.edu

Recent Events

R at the Statistics Canada Symposium

R was one of the software packages demonstrated at the 2003 Statistics Canada Symposium, an annual event organized by Statistics Canada and presented near Ottawa, Ontario. The event is attended largely by the methodologists (statisticians) who work here and at other national statistical agencies, such as the US Census Bureau. This year's symposium was the 20th edition of the conference. The four day event had two days of software demonstrations with different packages being shown each day. Out of the 10 different packages being demonstrated, 7 were developed at Statistics Canada for very specific tasks.

Although the software demonstrations were hidden in a room out of sight from the main proceedings, there were still several curious visitors to the R demo computer. We showed some basic demos of R at work in sampling and in simulation problems, emphasizing both the simplicity of the code and the results that can be produced. We also used the built-in graphics demo, which impressed many people. As most people working for Statistics Canada use SAS, many of our discussions were about why one would use R instead of SAS (besides the fact that R is free). We had many discussions on programming structure, creating plots, dealing with large amounts of data and general ease of use. All our visitors, when shown the software, seemed to like it and were not adverse to the idea of using it instead of SAS. In fact, they couldn't understand why we were not already using it more widely.

Krisztina Filep
Social Survey Methods Division
Statistics Canada
Ottawa, Ontario, Canada
Krisztina.Filep@statcan.ca

Upcoming Events

useR! 2004

The first international R user conference 'useR! 2004' will take place at the Technische Universität Wien in Vienna, Austria from 2004-05-20 to 2004-05-22. This R user conference will

- give an overview of the new features of the rapidly evolving R project and
- provide a platform for R users to discuss and exchange ideas how R can be used to do statistical computations, data analysis, visualization, teaching and exciting applications in various fields.

Keynote lectures

A huge amount of exciting features and innovations came up in the last releases of R. How these features can be used efficiently will be presented in the following keynote lectures given by R core team members:

- **R Key Features & Innovations**
  - **grid** Graphics and Programming, by Paul Murrell
  - Namespaces & Byte Compilation, by Luke Tierney
  - S4 Classes and Methods, by Friedrich Leisch
- **Users Become Developers**
  - Good Programming Practice, by Martin Mächler
  - Language Interfaces (.Call & .External), by Peter Dalgaard
  - Packaging, Documentation, Testing, by Kurt Hornik
- **Topics of Current Interest**
  - Datamining: Large Databases and Methods, by Brian D. Ripley
  - Multilevel Models in R: Present & Future, by Douglas Bates

User-contributed presentations

A major goal of the useR! conference is to bring users from various fields together and provide a platform for discussion and exchange of ideas: both in the official presentations as well as in Vienna's wine and beer pubs, cafes and restaurants. The formal part of the conference will consist of both oral and poster user-contributed presentations reflecting the wide range of fields in which R is used to analyze data such as: biostatistics & medical statistics; visualization & graphics; spatial statistics; graphical models; time series, econometrics & finance; R on Windows and MacOS; office integration & user interfaces; teaching with R and many, many more...

Further information on the program, submission of abstracts and registration details are available online on the conference web page at http://www.ci.tuwien.ac.at/Conferences/useR-2004/.

See you in Vienna!

For the organizing committee:
Torsten Hothorn, Erlangen
Achim Zeileis and David Meyer, Wien
useR-org@ci.tuwien.ac.at

COMPSTAT 2004

COMPSTAT 2004, the 16th Symposium of the International Association for Statistical Computing (IASC, http://www.cbs.nl/isi/iasc.htm), will take place at Charles University in Prague, Czech Republic, from 2004-08-23 to 2004-08-27.

As at previous COMPSTATs, the scientific program will cover all aspects of the link between statistical theory and applied statistics provided by statistical computing, from the development and implementation of new statistical ideas through to user experiences and software evaluation. The program should appeal to anyone working in statistics and using computers, whether in universities, industrial companies, government agencies, research institutes or as software developers. On Aug 22, I will deliver a half-day pre-conference tutorial on "R: The Next Generation", introducing the key innovations in the R 2.0 release scheduled for 2004. There are no keynote lectures or invited organized sessions specifically dedicated to R.

The conference web site (http://compstat2004.cuni.cz/) has more details.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Kurt.Hornik@R-project.org

Changes in R 1.8.1

by the R Core Team

New features

- There is now a "Complex" S3 group generic (a sideeffect of fixing up the corresponding S4 group generic).
- `help("regex")` now gives a description of the regular expressions used in R.
- The startup message now shows the R Foundation as copyright holder, and includes the R ISBN number and a pointer to the new `citation()` function.
- The `solve()` function now uses the 'tol' argument for all noncomplex cases. The default tolerance for LINPACK is 1e7, as before. For LAPACK it currently is `.Machine$double.eps` but may be changed in later versions of R.
- `help.search()` now defaults to `agrep = FALSE` when `keyword=` is specified, since no one wants fuzzy matching of categories.
- Function `texi2dvi()` in package tools can be used to compile latex files from within R.
- Objects with formal S4 classes saved in pre1.8 versions and loaded into the current version have incompatible class attributes (no package information). A new function, `fixPre1.8()` in package methods, will fix the class attributes. See the help for this function.
- `heatmap()` allows `Rowv/Colv = NA`, suppressing the corresponding dendrogram.
- An "antifeature": Tcl 8.0 is now officially unsupported. In 1.8.0 it just didn't work. This very old version lacks several features that are needed for the new version of the tcltk package. R will still build the tcltk package against Tcl 8.0 but the resulting package will not load.

The above lists only new features, see the 'NEWS' file in the R distribution or on the R homepage for a list of bug fixes.

Changes on CRAN

by Kurt Hornik and Friedrich Leisch

New contributed packages

CDNmoney Components of Canadian Monetary Aggregates with continuity adjustments. By Paul Gilbert.

HI Simulation from distributions supported by nested hyperplanes, using the algorithm described in Petris & Tardella, "A geometric approach to transdimensional Markov chain Monte Carlo", Canadian Journal of Statistics, v.31, n.4, (2003). Also random direction multivariate Adaptive Rejection Metropolis Sampling. By Giovanni Petris and Luca Tardella.

asypow A set of routines written in the S language that calculate power and related quantities utilizing asymptotic likelihood ratio methods. S original by Barry W. Brown, James Lovato and Kathy Russel. R port by Kjetil Halvorsen.

concord Measures of concordance and reliability. By Jim Lemon.

covRobust The `cov.nnve()` function for robust covariance estimation by the nearest neighbor variance estimation (NNVE) method of Wang and Raftery (2002,JASA). By Naisyin Wang and Adrian Raftery with contributions from Chris Fraley.

digest The digest package provides two functions for the creation of 'hash' digests of arbitrary R objects using the md5 and sha1 algorithms permitting easy comparison of R language objects. The md5 algorithm by Ron Rivest is specified in RFC 1321. The SHA1 algorithm is specified in FIPS1801. This packages uses two small standalone C implementations (that were provided by by Christophe Devine) of the md5 and sha1 algorithms. Please note that this package is not meant to be used for cryptographic purposes for which more comprehensive (and widely tested) libraries such as OpenSSL should be used. By Dirk Eddelbuettel.

fda Analysis of functional data, that is data where the basic observation is a function of some sort—an area involving the generalization of the techniques of multivariate data analysis to functional data. By Jim Ramsay.

fork This library provides a simple wrapper around the Unix process management API calls: `fork`, `wait`, `waitpid`, `kill`, and `_exit`. These commands allow construction of programs that utilize multiple concurrent processes. By Gregory R Warnes.

labstatR This package contains sets of functions defined in "Laboratorio di Statistica con R", Iacus & Masarotto, MacGraw-Hill Italia, 2003. Function names and docs are in italian as well. By Stefano M.Iacus and Guido Masarotto.

lazy By combining constant, linear, and quadratic local models, lazy estimates the value of an unknown multivariate function on the basis of a set of possibly noisy samples of the function itself. This implementation of lazy learning automatically adjusts the bandwidth on a query-by-query basis through a leave-one-out cross-validation. By Mauro Birattari and Gianluca Bontempi.

ldDesign R package for design of experiments for association studies for detection of linkage disequilibrium. Uses an existing deterministic power calculation for detection of linkage disequilibrium between a bi-allelic QTL and a bi-allelic marker, together with the Spiegelhalter and Smith Bayes factor to generate designs with power to detect effects with a given Bayes factor. By Rod Ball.

lmm Some improved procedures for linear mixed models. S original by Joseph L. Schafer, R port by Jing hua Zhao.

magic A variety of methods for creating magic squares of any order greater than 2, and various magic hypercubes. Includes sundry magic square amusements and is intended as a rebuttal to the often-heard comment "I thought R was just for statistics". By Robin K. S. Hankin.

mapproj Converts latitude/longitude into projected coordinates. By Doug McIlroy. Packaged for R by Ray Brownrigg and Thomas P Minka.

pan Multiple imputation for multivariate panel or clustered data. S original by Joseph L. Schafer, R port by Jing hua Zhao.

pheno Provides some easy-to-use functions for time series analyses of (plant-) phenological data sets. These functions mainly deal with the estimation of combined phenological time series and are usually wrappers for functions that are already implemented in other R packages adapted to the special structure of phenological data and the needs of phenologists. By Joerg Schaber.

phyloarray Software to process data from phylogenetic or identification microarrays. At present state, it is rather limited and focuss was on a fast and easy way for calculating background values by interpolation and plotting melting curves. The functions for reading the data are similar to those used in package **sma** (statistical microarray analysis). By Kurt Sys.

rgdal Provides bindings to Frank Warmerdam's Geospatial Data Abstraction Library (GDAL) (>= 1.1.8) - this version with new-style classes. The GDAL library is external to the package, and must be correctly installed first. By Timothy H. Keitt and Roger Bivand.

Other changes

- Package **normix** was renamed to **nor1mix**.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Kurt.Hornik@R-project.org

Friedrich Leisch
Technische Universität Wien, Austria
Friedrich.Leisch@R-project.org

R Foundation News

by Bettina Grün

Donations

- Brian Caffo (USA)
- Department of Economics, University of Milano (Italy)
- Hort Research Institute (New Zealand)
- Biostatistics and Research Decision Sciences, Merck Research Laboratories (USA)
- Minato Nakazawa (Japan)
- Bill Pikounis (USA)
- James Robison-Cox (USA)
- Boris Vaillant (Germany)

New supporting institutions

- Department of Biostatistics, Johns Hopkins University, Maryland, USA
- Loyalty Matrix Inc., California, USA

New supporting members

Adelchi Azzalini (Italy)
Paul J. Brewer (Hong Kong)
Olaf Bürger (Germany)
James Curran (New Zealand)
Davide Fiaschi (Italy)
John Field (Australia)
John R. Gleason (USA)
Darlene Goldstein (Switzerland)
Eugène Horber (Switzerland)
Graham Jones (UK)
Christian Keller (Switzerland)
Shigeru Mase (Japan)
Daniel Mathews (USA)
Roger Peng (USA)
Bill Pikounis (USA)
Michael Tiefelsdorf (USA)
Harald Weedon-Fekjær (Norway)
Bernd Weiß (Germany)
Brandon Whitcher (UK)

Bettina Grün
Technische Universität Wien, Austria
Bettina.Gruen@ci.tuwien.ac.at

R News is a publication of the R Foundation for Statistical Computing, communications regarding this publication should be addressed to the editors. All articles are copyrighted by the respective authors. Please send submissions to regular columns to the respective column editor, all other submissions to the editor-in-chief or another member of the editorial board (more detailed submission instructions can be found on the R homepage).

R Project Homepage:
http://www.R-project.org/

This newsletter is available online at
http://CRAN.R-project.org/doc/Rnews/