R News

The Newsletter of the R Project
Volume 5/2, November 2005

Editorial

by Douglas Bates

One of the strengths of the R Project is its package system and its network of archives of packages. All useRs will be (or at least should be) familiar with CRAN, the Comprehensive R Archive Network, which provides access to more than 600 contributed packages. This wealth of contributed software is a tribute to the useRs and developeRs who have written and contributed those packages and to the CRAN maintainers who have devoted untold hours of effort to establishing, maintaining and continuously improving CRAN.

Kurt Hornik and Fritz Leisch created CRAN and continue to do the lion's share of the work in maintaining and improving it. They were also the founding editors of *R News*. Part of their vision for *R News* was to provide a forum in which to highlight some of the packages available on CRAN, to explain their purpose and to provide an illustration of their use.

This issue of *R News* brings that vision to fruition in that almost all our articles are written about CRAN packages by the authors of those packages.

We begin with an article by Raftery, Painter and Volinsky about the BMA package that uses Bayesian Model Averaging to deal with uncertainty in model selection.

Next Pebesma and Bivand discuss the sp package in the context of providing the underlying classes and methods for working with spatial data in R. Early on they mention the **task view** for spatial statistical analysis that is being maintained on CRAN. Task views are a new feature on CRAN designed to make it easier for those who are interested in a particular topic to navigate their way through the multitude of packages that are available and to find the ones targeted to their particular interests.

To provide the exception that proves the rule, Xie describes the use of condor, a batch process scheduling system, and its DAG (directed acylic graph) capability to coordinate remote execution of long-running R processes. This article is an exception in that it doesn't deal with an R package but it certainly should be of interest to anyone using R for running simulations or for generating MCMC samples, etc. and for anyone who works with others who have such applications. I can tell you from experience that the combination of condor and R, as described by Xie, can help enormously in maintaining civility in a large and active group of statistics researchers who share access to computers.

Continuing in the general area of distributed statistical computing, L'Ecuyer and Leydold describe their rstream package for maintaining multiple reproducible streams of pseudo-random numbers, possibly across simulations that are being run in parallel.

This is followed by Benner's description of his mfp package for multivariable fractional polynomials and Sailer's description of his crossdes package

Contents of this issue:

Editorial 1
BMA: An R package for Bayesian Model Averaging 2
Classes and methods for spatial data in R . . . 9
Running Long R Jobs with Condor DAG 13
rstream: Streams of Random Numbers for Stochastic Simulation 16
mfp 20
crossdes 24
R Help Desk 27
Changes in R 29
Changes on CRAN 36
Forthcoming Events: useR! 2006 43

for design and randomization in crossover studies.

We round out the issue with regular features that include the R Help Desk, in which Duncan Murdoch joins Uwe Ligges to explain the arcane art of building R packages under Windows, descriptions of changes in the 2.2.0 release of R and recent changes on CRAN and an announcement of an upcoming event, useR!2006. Be sure to read all the way through to the useR! announcement. If this meeting is anything like the previous useR! meetings – and it will be – it will be great! Please do consider attending.

This is my last issue on the *R News* editorial board and I would like to take this opportunity to extend my heartfelt thanks to Paul Murrell and Torsten Hothorn, my associate editors. They have again stepped up and shouldered the majority of the work of preparing this issue while I was distracted by other concerns. Their attitude exemplifies the spirit of volunteerism and helpfulness that makes it so rewarding (and so much fun) to work on the R Project.

Douglas Bates
University of Wisconsin – Madison, U.S.A.
bates@R-project.org

BMA: An R package for Bayesian Model Averaging

by Adrian E. Raftery, Ian S. Painter and Christopher T. Volinsky

Bayesian model averaging (BMA) is a way of taking account of uncertainty about model form or assumptions and propagating it through to inferences about an unknown quantity of interest such as a population parameter, a future observation, or the future payoff or cost of a course of action. The BMA posterior distribution of the quantity of interest is a weighted average of its posterior distributions under each of the models considered, where a model's weight is equal to the posterior probability that it is correct, given that one of the models considered is correct.

Model uncertainty can be large when observational data are modeled using regression, or its extensions such as generalized linear models or survival (or event history) analysis. There are often many modeling choices that are secondary to the main questions of interest but can still have an important effect on conclusions. These can include which potential confounding variables to control for, how to transform or recode variables whose effects may be nonlinear, and which data points to identify as outliers and exclude. Each combination of choices represents a statistical model, and the number of possible models can be enormous.

The R package `BMA` provides ways of carrying out BMA for linear regression, generalized linear models, and survival or event history analysis using Cox proportional hazards models. The functions `bicreg`, `bic.glm` and `bic.surv`, account for uncertainty about the variables to be included in the model, using the simple BIC (Bayesian Information Criterion) approximation to the posterior model probabilities. They do an exhaustive search over the model space using the fast leaps and bounds algorithm. The function `glib` allows one to specify one's own prior distribution. The function `MC3.REG` does BMA for linear regression using Markov chain Monte Carlo model composition (MC^3), and allows one to specify a prior distribution and to make inference about the variables to be included and about possible outliers at the same time.

Basic ideas

Suppose we want to make inference about an unknown quantity of interest Δ, and we have data D. We are considering several possible statistical models for doing this, $M_1, \ldots, M_K$. The number of models could be quite large. For example, if we consider only regression models but are unsure about which of p possible predictors to include, there could be as many as 2^p models considered. In sociology and epidemiology, for example, values of p on the order of 30 are not uncommon, and this could correspond to around 2^{30}, or one billion models.

Bayesian statistics expresses all uncertainties in terms of probability, and make all inferences by applying the basic rules of probability calculus. BMA is no more than basic Bayesian statistics in the presence of model uncertainty. By a simple application of the law of total probability, the BMA posterior distribution of Δ is

$$p(\Delta|D) = \sum_{k=1}^{K} p(\Delta|D, M_k) p(M_k|D), \qquad (1)$$

where $p(\Delta|D, M_k)$ is the posterior distribution of Δ given the model M_k, and $p(M_k|D)$ is the posterior probability that M_k is the correct model, given that one of the models considered is correct. Thus the BMA posterior distribution of Δ is a weighted average of the posterior distributions of Δ under each of the models, weighted by their posterior model probabilities. In (1), $p(\Delta|D)$ and $p(\Delta|D, M_k)$ can be probability density functions, probability mass functions, or cumulative distribution functions.

The posterior model probability of M_k is given by

$$p(M_k|D) = \frac{p(D|M_k)p(M_k)}{\sum_{\ell=1}^{K} p(D|M_\ell)p(M_\ell)}. \qquad (2)$$

In equation (2), $p(D|M_k)$ is the *integrated likelihood* of model M_k, obtained by integrating (not maximizing) over the unknown parameters:

$$\begin{aligned} p(D|M_k) &= \int p(D|\theta_k, M_k)p(\theta_k|M_k)d\theta_k \qquad (3)\\ &= \int (\text{ likelihood } \times \text{ prior })d\theta_k, \end{aligned}$$

where θ_k is the parameter of model M_k and $p(D|\theta_k, M_k)$ is the likelihood of θ_k under model M_k. The prior model probabilities are often taken to be equal, so they cancel in (2), but the BMA package allows other formulations also.

The integrated likelihood $p(D|M_k)$ is a high dimensional integral that can be hard to calculate analytically, and several of the functions in the BMA package use the simple and surprisingly accurate BIC approximation

$$2 \log p(D|M_k) \approx 2 \log p(D|\hat{\theta}_k) - d_k \log(n) = -\,\text{BIC}_k, \qquad (4)$$

where $d_k = \dim(\theta_k)$ is the number of independent parameters in M_k, and $\hat{\theta}_k$ is the maximum likelihood estimator (equal to the ordinary least squares estimator for linear regression coefficients). For linear regression, BIC has the simple form

$$\text{BIC}_k = n \log(1 - R_k^2) + p_k \log n \qquad (5)$$

up to an additive constant, where R_k^2 is the value of R^2 and p_k is the number of regressors for the kth regression model. By (5), $\text{BIC}_k = 0$ for the null model with no regressors.

When interest focuses on a model parameter, say a regression parameter such as β_1, (1) can be applied with $\Delta = \beta_1$. The BMA posterior mean of β_1 is just a weighted average of the posterior means of β_1 under each of the models:

$$E[\beta_1|D] = \sum \tilde{\beta}_1^{(k)} p(M_k|D), \qquad (6)$$

which can be viewed as a model-averaged Bayesian point estimator. In (6), $\tilde{\beta}_1^{(k)}$ is the posterior mean of β_1 under model M_k, and this can be approximated by the corresponding maximum likelihood estimator, $\hat{\beta}_1^{(k)}$ (Raftery, 1995). A similar expression is available for the BMA posterior standard deviation, which can be viewed as a model-averaged Bayesian standard error. For a survey and literature review of Bayesian model averaging, see Hoeting et al. (1999).

Computation

Implementing BMA involves two hard computational problems: computing the integrated likelihoods for all the models (3), and averaging over all the models, whose number can be huge, as in (1) and (6). In the functions `bicreg` (for variable selection in linear regression), `bic.glm` (for variable selection in generalized linear models), and `bic.surv` (for variable selection in survival or event history analysis), the integrated likelihood is approximated by the BIC approximation (4). The sum over all the models is approximated by finding the best models using the fast leaps and bounds algorithm. The leaps and bounds algorithm was introduced for all subsets regression by Furnival and Wilson (1974), and extended to BMA for linear regression and generalized linear models by Raftery (1995), and to BMA for survival analysis by Volinsky et al. (1997). Finally, models that are much less likely *a posteriori* than the best model are excluded. This is an exhaustive search and finds the globally optimal model.

If the number of variables is large, however, the leap and bounds algorithm can be substantially slowed down; we have found that it can slow down substantially once the number of variables goes beyond about 40–45. One way around this is via specification of the argument `maxCol`. If the number of variables is greater than `maxCol` (whose current default value is 31), the number of variables is reduced to `maxCol` by backwards stepwise elimination before applying the leaps and bounds algorithm.

In the function `glib` (model selection for generalized linear models with prior information), the integrated likelihood is approximated by the Laplace method (Raftery, 1996). An example of the use of `glib` to analyze epidemiological case-control studies is given by Viallefont et al. (2001). In the function `MC3.REG` (variable selection and outlier detection for linear regression), conjugate priors and exact integrated likelihoods are used, and the sum is approximated using Markov chain Monte Carlo (Hoeting et al., 1996; Raftery et al., 1997).

Example 1: Linear Regression

To illustrate how BMA takes account of model uncertainty about the variables to be included in linear regression, we use the `UScrime` dataset on crime rates in 47 U.S. states in 1960 (Ehrlich, 1973); this is available in the MASS library. There are 15 potential independent variables, all associated with crime rates in the literature. The last two, probability of imprisonment and average time spent in state prisons, were the predictor variables of interest in the original study, while the other 13 were control variables. All variables for which it makes sense were logarithmically transformed. The commands are:

```
library(BMA)
library(MASS)
data(UScrime)
x.crime<- UScrime[,-16]
y.crime<- log(UScrime[,16])
x.crime[,-2]<- log(x.crime[,-2])
crime.bicreg <- bicreg(x.crime, y.crime)
summary (crime.bicreg, digits=2)
```

This summary yields Figure 1. (The default number of digits is 4, but this may be more than needed.) The column headed "`p!=0`" shows the posterior probability that the variable is in the model (in %). The column headed "`EV`" shows the BMA posterior mean, and the column headed "`SD`" shows the BMA posterior standard deviation for each variable. The following five columns show the parameter estimates for the best five models found, together with the numbers of variables they include, their R^2 values, their BIC values and their posterior model probabilities.

The `plot` command shows the BMA posterior distribution of each of the regression parameters, given by (1) with $\Delta = \beta_k$. For example, the BMA posterior distribution of the coefficient of the average time in prison variable is shown in Figure 2. The spike at 0 shows the probability that the variable is not in the model, in this case 0.445. The curve shows the posterior density given that the variable is in the model, which is approximated by a finite mixture of normal densities and scaled so that the height of the density curve is equal to the posterior probability that the variable is in the model. The BMA posterior distributions of all the parameters are produced by the command
`> plot (crime.bicreg,mfrow=c(4,4))`
and shown in Figure 3.

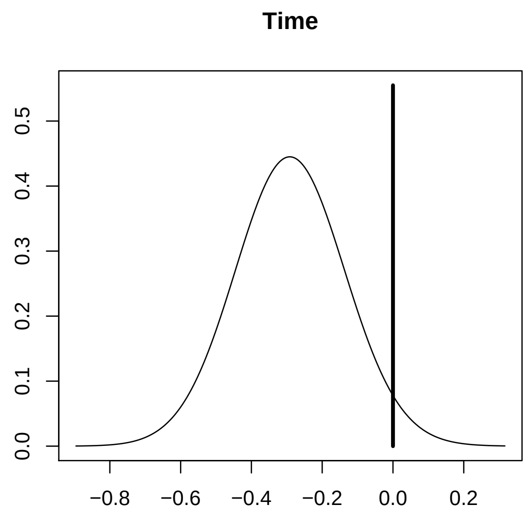

Figure 2: BMA posterior distribution of the coefficient of the average time in prison variable in the crime dataset. The spike at 0 shows the posterior probability that the variable is not in the model. The curve is the model-averaged posterior density of the coefficient given that the variable is in the model, approximated by a finite mixture of normal distributions, one for each model that includes the variable. The density is scaled so that its maximum height is equal to the probability of the variable being in the model.

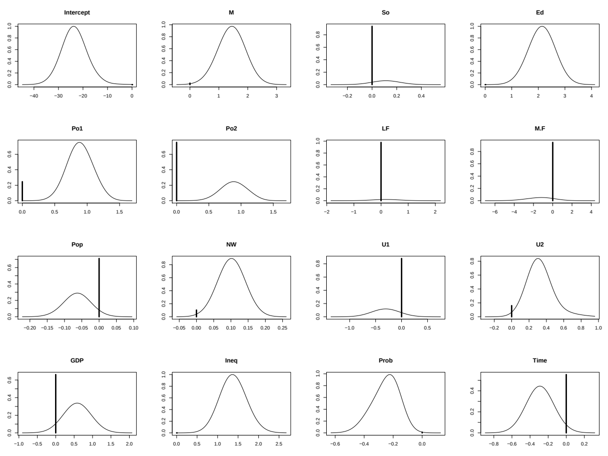

Figure 3: BMA posterior distribution of all the coefficients for the crime dataset, produced by the command `plot(crime.bicreg,mfrow=c(4,4))`

A visual summary of the BMA output is produced by the `imageplot.bma` command, as shown in Figure 4. Each row corresponds to a variable, and each column corresponds to a model; the corresponding rectangle is red if the variable is in the model and white otherwise. The width of the column is proportional to the model's posterior proba-

```
Call: bicreg(x = x.crime, y = y.crime)
  51  models were selected
 Best  5  models (cumulative posterior probability =  0.29 ):

           p!=0    EV       SD     model 1  model 2  model 3  model 4  model 5
Intercept  100.0  -23.4778  5.463  -22.637  -24.384  -25.946  -22.806  -24.505
M           97.5    1.4017  0.531    1.478    1.514    1.605    1.268    1.461
So           6.3    0.0069  0.039      .        .        .        .        .
Ed         100.0    2.1282  0.513    2.221    2.389    2.000    2.178    2.399
Po1         75.4    0.6707  0.423    0.852    0.910    0.736    0.986      .
Po2         24.6    0.2186  0.396      .        .        .        .      0.907
LF           2.1    0.0037  0.080      .        .        .        .        .
M.F          5.2   -0.0691  0.446      .        .        .        .        .
Pop         28.9   -0.0183  0.036      .        .        .     -0.057      .
NW          89.6    0.0911  0.050    0.109    0.085    0.112    0.097    0.085
U1          11.9   -0.0364  0.136      .        .        .        .        .
U2          83.9    0.2740  0.191    0.289    0.322    0.274    0.281    0.330
GDP         33.8    0.1971  0.354      .        .      0.541      .        .
Ineq       100.0    1.3810  0.333    1.238    1.231    1.419    1.322    1.294
Prob        98.8   -0.2483  0.100   -0.310   -0.191   -0.300   -0.216   -0.206
Time        44.5   -0.1289  0.178   -0.287      .     -0.297      .        .

nVar                                    8        7        9        8        7
r2                                    0.842    0.826    0.851    0.838    0.823
BIC                                 -55.912  -55.365  -54.692  -54.604  -54.408
post prob                             0.089    0.067    0.048    0.046    0.042
```

Figure 1: Summary of the output of `bicreg` for the crime data

bility. The basic idea of the image plot was proposed by Clyde (1999); in her version all the columns had the same width, while in the `imageplot.bma` output they depend on the models' posterior probabilities.

It is clear that M (percent male), Education, NW (percent nonwhite), Inequality and Prob (probability of imprisonment) have high posterior probabilities of being in the model, while most other variables, such as So (whether the state is in the South), have low posterior probabilities. The Time variable (average time spent in state prisons) does appear in the best model, but there is nevertheless a great deal of uncertainty about whether it should be included. The two variables Po1 (police spending in 1959) and Po2 (police spending in 1960) are highly correlated, and the models favored by BMA include one or the other, but not both.

Example 2: Logistic Regression

We illustrate BMA for logistic regression using the low birthweight data set of Hosmer and Lemeshow (1989), available in the MASS library. The dataset consists of 189 babies, and the dependent variable measures whether their weight was low at birth. There are 8 potential independent variables of which two are categorical with more than two categories: race and the number of previous premature labors (`ptl`). Figure 5 shows the output of the commands

```
library(MASS)
data(birthwt)
birthwt$race <- as.factor (birthwt$race)
birthwt$ptl <- as.factor (birthwt$ptl)
bwt.bic.glm <- bic.glm (low ~ age + lwt
 + race + smoke + ptl + ht + ui + ftv,
 data=birthwt, glm.family="binomial")
summary (bwt.bic.glm,conditional=T,digits=2)
```

The function `bic.glm` can accept a formula, as here, instead of a design matrix and dependent variable (the same is true of `bic.surv` but not of `bicreg`). By default, the levels of a categorical factor such as race are constrained to be either all in the model or all out of it. However, by specifying `factor.type=F` in `bic.glm`, one can allow the individual dummy variables specifying a factor to be in or out of the model.

The posterior distributions of the model parameters are shown in Figure 6, and the image plot is shown in Figure 7. Note that each dummy variable making up the race factor has its own plot in Figure 6, but the race factor has only one row in Figure 7.

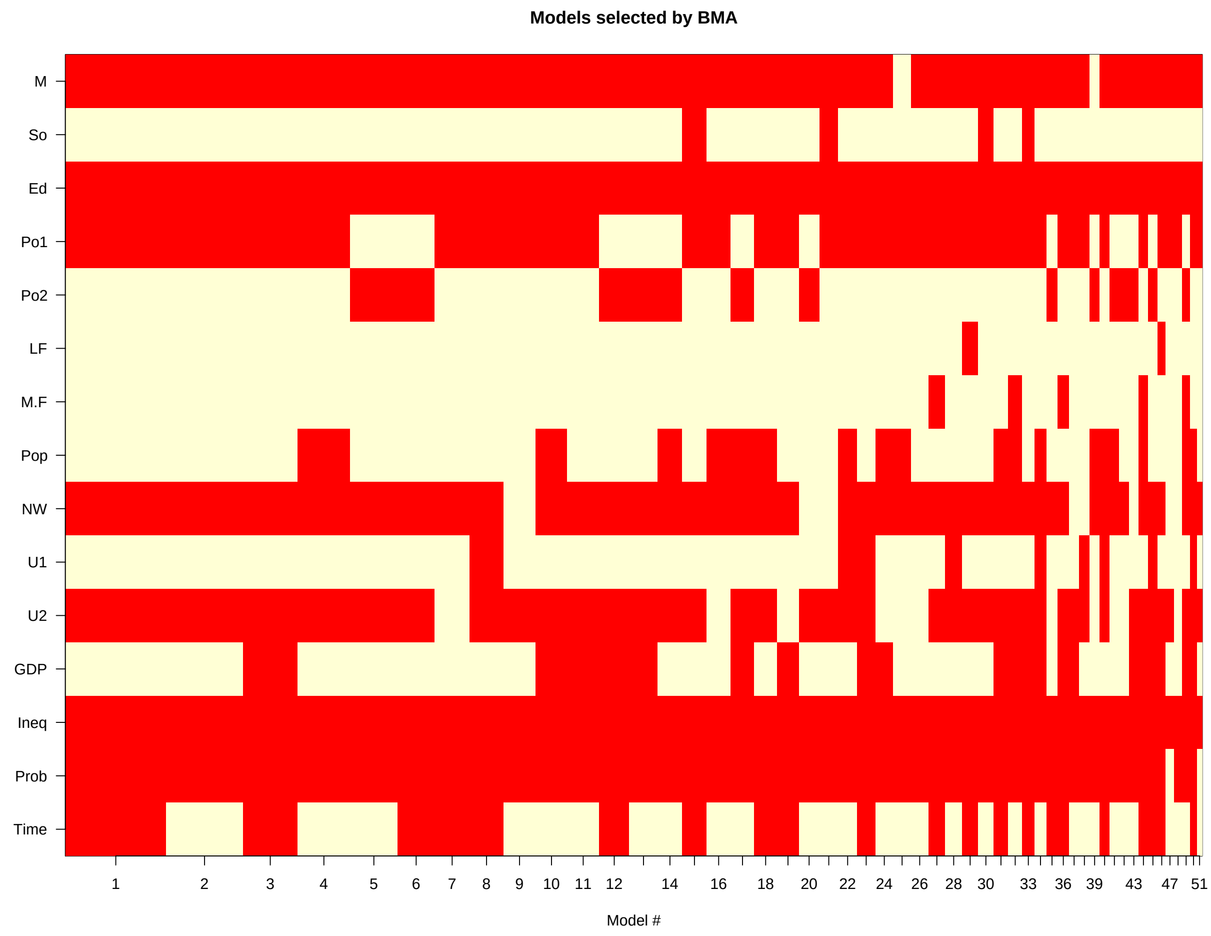

Figure 4: Image plot for the crime data produced by the command `imageplot.bma(crime.bicreg)`

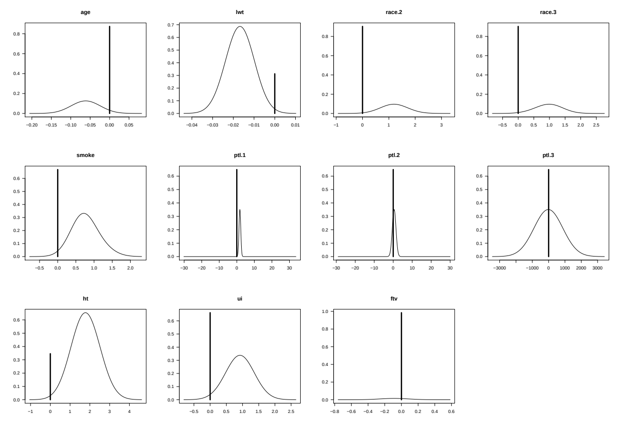

Figure 6: BMA posterior distributions of the parameters of the logistic regression model for the low birthweight data. Note that there is a separate plot for each of the dummy variables for the two factors (race and `ptl`)

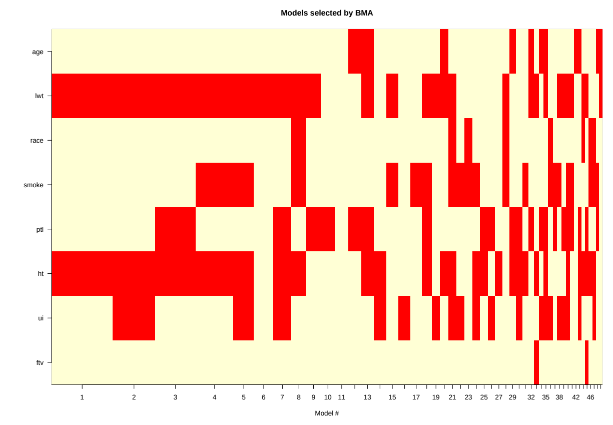

Figure 7: Image plot summarizing the BMA analysis for logistic regression of the low birthweight data

Example 3: Survival Analysis

BMA can account for model uncertainty in survival analysis using the Cox proportional hazards model. We illustrate this using the well-known Primary Biliary Cirrhosis (PBC) data set of Fleming and Harrington (1991). This consists of 312 randomized pa-

```
Call: bic.glm.formula(f = low ~ age+lwt+race+smoke+ptl+ht+ui+ftv, data = birthwt, glm.family = "binomial")
  49  models were selected
 Best  5  models (cumulative posterior probability =  0.37 ):

           p!=0   EV      SD       cond EV  cond SD  model 1    model 2    model 3    model 4    model 5
Intercept  100    0.4716  1.3e+00    0.472    1.309     1.451      1.068      1.207      1.084      0.722
age        12.6  -0.0078  2.4e-02   -0.062    0.037       .          .          .          .          .
lwt        68.7  -0.0116  9.7e-03   -0.017    0.007    -0.019     -0.017     -0.019     -0.018     -0.016
race        9.6
    .2            0.1153  3.9e-01    1.201    0.547       .          .          .          .          .
    .3            0.0927  3.2e-01    0.966    0.461       .          .          .          .          .
smoke      33.2   0.2554  4.3e-01    0.768    0.397       .          .          .        0.684      0.653
ptl        35.1
   .1             0.6174  8.9e-01    1.758    8.211       .          .        1.743        .          .
   .2             0.1686  6.1e-01    0.480    7.592       .          .        0.501        .          .
   .3            -4.9110  5.2e+02  -13.988  882.840       .          .      -13.986        .          .
ht         65.4   1.1669  1.0e+00    1.785    0.729     1.856      1.962      1.924      1.822      1.922
ui         33.8   0.3105  5.1e-01    0.918    0.445       .        0.930        .          .        0.896
ftv         1.5  -0.0013  2.4e-02   -0.087    0.173       .          .          .          .          .

nVar                                                      2          3          3          3          4
BIC                                                    -753.823   -753.110   -752.998   -752.865   -751.656
post prob                                                 0.111      0.078      0.073      0.069      0.037
```

Figure 5: Summary of the `bic.glm` output for the low birthweight data. For factors, either all levels are in or all levels are out of the model. With the option `conditional=T`, the posterior means and standard deviations of the parameters conditionally on the variable being in the model are shown in addition to the unconditional ones.

tients with the disease, and the dependent variable was survival time; 187 of the records were censored. There were 15 potential predictors. The data set is available in the `survival` library, which is loaded when `BMA` is loaded.

The analysis proceeded as follows:

```
data(pbc)
x.pbc<- pbc[1:312,]
surv.t<- x.pbc$time
cens<- x.pbc$status
x.pbc<- x.pbc[,-c(6,7,10,17,19)]
x.pbc$bili<- log(x.pbc$bili)
x.pbc$alb<- log(x.pbc$alb)
x.pbc$protime<- log(x.pbc$protime)
x.pbc$copper<- log(x.pbc$copper)
x.pbc$sgot<- log(x.pbc$sgot)
pbc.bic.surv <- bic.surv(x.pbc,surv.t,cens)
summary(pbc.bic.surv,digits=2)
plot(pbc.bic.surv,mfrow=c(4,4))
imageplot.bma(pbc.bic.surv)
```

The summary results, BMA posterior distributions, and image plot visualization of the results are shown in Figures 8, 9 and 10.

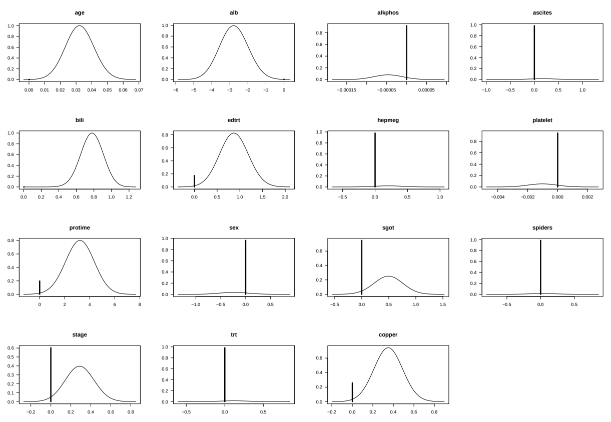

Figure 9: BMA posterior distributions of the parameters of the Cox proportional hazards model model for the PBC data

```
Call: bic.surv.data.frame(x = x, surv.t = surv.t, cens = cens)
  39  models were selected
 Best  5  models (cumulative posterior probability =  0.37 ):

          p!=0    EV        SD        model 1    model 2    model 3    model 4    model 5
age       100.0   3.2e-02  9.2e-03     0.032      0.029      0.031      0.031      0.036
alb        99.2  -2.7e+00  8.3e-01    -2.816     -2.446     -2.386     -3.268     -2.780
alkphos     8.1  -3.7e-06  1.6e-05      .          .          .          .          .
ascites     1.6   2.6e-03  4.3e-02      .          .          .          .          .
bili      100.0   7.8e-01  1.3e-01     0.761      0.743      0.786      0.801      0.684
edtrt      82.6   7.2e-01  4.4e-01     0.813      0.821      1.026       .         0.837
hepmeg      2.0   4.1e-03  4.3e-02      .          .          .          .          .
platelet    5.4  -5.5e-05  3.2e-04      .          .          .          .          .
protime    80.2   2.6e+00  1.6e+00     3.107      2.706       .         3.770      3.417
sex         3.4  -8.0e-03  6.8e-02      .          .          .          .          .
sgot       25.2   1.2e-01  2.5e-01      .          .          .          .         0.407
spiders     1.3   3.7e-04  2.5e-02      .          .          .          .          .
stage      39.7   1.1e-01  1.7e-01      .         0.244      0.318       .          .
trt         1.6   1.9e-03  2.8e-02      .          .          .          .          .
copper     74.2   2.6e-01  2.0e-01     0.358      0.347      0.348      0.357      0.311

nVar                                     6          7          6          5          7
BIC                                 -177.952   -176.442   -176.212   -175.850   -175.537
post prob                              0.147      0.069      0.062      0.051      0.044
```

Figure 8: Summary of the `bic.surv` output for the PBC data.

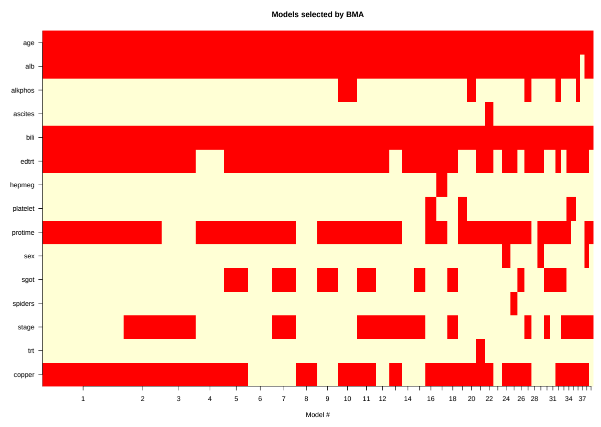

Figure 10: Image plot summarizing the BMA analysis for Cox proportional hazards modeling of the PBC data

Summary

The `BMA` package carries out Bayesian model averaging for linear regression, generalized linear models, and survival or event history analysis using Cox proportional hazards models. The library contains functions for plotting the BMA posterior distributions of the model parameters, as well as an image plot function that provides a way of visualizing the BMA output. The functions `bicreg`, `bic.glm` and `bic.surv` provide fast and automatic default ways of doing this for the model classes considered. Prior information can be incorporated explicitly using the `glib` and `MC3.REG` functions, and `MC3.REG` also takes account of uncertainty about outlier removal.

Acknowledgement

The `MC3.REG` function is based on an earlier Splus function written by Jennifer A. Hoeting.

Bibliography

M. A. Clyde. Bayesian model averaging and model search strategies (with Discussion). In *Bayesian Statistics 6* (edited by J. M. Bernardo et al.), pages 157–185. Oxford University Press, Oxford, U.K., 1999. 5

I. Ehrlich. Participation in illegitimate activities: a theoretical and empirical investigation. *Journal of Political Economy*, 81:521–565, 1973. 3

T. R. Fleming and D. P. Harrington. *Counting processes and survival analysis*. Wiley, New York, 1991. 6

G. M. Furnival and R. W. Wilson. Regression by leaps and bounds. *Technometrics*, 16:499–511, 1974. 3

J. A. Hoeting, A. E. Raftery, and D. Madigan. A method for simultaneous variable selection and outlier identification in linear regression. *Computational Statistics and Data Analysis*, 22:251–270, 1996. 3

J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. Bayesian model averaging: A tutorial (with discussion). *Statistical Science*, 14:382–417, 1999. 3

D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley, New York, 1989. 5

A. E. Raftery. Bayesian model selection in social research (with Discussion). In *Sociological Methodology 1995* (edited by P. V. Marsden), pages 111–163. Cambridge, Mass. : Blackwell Publishers, 1995. 3

A. E. Raftery. Approximate Bayes factors and accounting for model uncertainty in generalized linear models. *Biometrika*, 83:251–266, 1996. 3

A. E. Raftery, D. Madigan, and J. A. Hoeting. Model selection and accounting for model uncertainty in linear regression models. *Journal of the American Statistical Association*, 92:179–191, 1997. 3

V. Viallefont, A. E. Raftery, and S. Richardson. Variable selection and Bayesian model averaging in case-control studies. *Statistics in Medicine*, 20:3215–3230, 2001. 3

C. T. Volinsky, D. Madigan, A. E. Raftery, and R. A. Kronmal. Bayesian model averaging in proportional hazards models: Assessing the risk of a stroke. *Applied Statistics*, 46:433–448, 1997. 3

Classes and methods for spatial data in R

by Edzer J. Pebesma and Roger S. Bivand

R has been used for spatial statistical analysis for a long time: packages dedicated to several areas of spatial statistics have been reviewed in Bivand and Gebhardt (2000) and Ripley (2001), and more recent packages have been published (Ribeiro and Diggle, 2001; Pebesma, 2004; Baddeley and Turner, 2005). In addition, R links to GIS have been developed (Bivand, 2000, 2005; Gómez-Rubio and López-Quílez, 2005). A task view of spatial statistical analysis is being maintained on CRAN.

Many of the spatial packages use their own data structures (classes) for the spatial data they need or create, and provide methods, e.g. for plotting them. However, up to now R has been lacking a full-grown coherent set of classes and methods for the major spatial data types: points, lines, polygons, and grids. Package **sp**, available on CRAN since May 2005, has the ambition to fill this gap.

We believe there is a need for **sp** because

(i) with support for a single package for spatial data, it is much easier to go from one spatial statistics package to another. Its classes are either supported directly by the packages, reading and writing data in the new spatial classes, or indirectly e.g. by supplying data conversion between **sp** and the package in an interface package. This requires one-to-many links, which are easier to provide and maintain than many-to-many links,

(ii) the new package provides a well-tested set of methods (functions) for plotting, printing, subsetting and summarizing spatial objects, or combining (overlay) spatial data types,

(iii) packages with interfaces to GIS and geographic (re)projection code support the new classes,

(iv) the new package provides Lattice plots, conditioning plots, plot methods that combine points, lines, polygons and grids with map elements (reference grids, scale bars, north arrows), degree symbols (52°N) in axis labels, etc.

This article introduces the classes and methods provided by **sp**, discusses some of the implementation details, and the state and potential of linking **sp** to other packages.

Classes and implementation

The spatial classes in **sp** include points, lines, polygons and grids; each of these structures can have multiple attribute variables, describing what is actually registered (e.g. measured) for a certain location or area. Single entries in the attribute table may correspond to multiple lines or polygons; this is useful because e.g. administrative regions may consist of several polygons (mainland, islands). Polygons may further contain holes (e.g. a lake), polygons in holes (island in lake), holes in polygons in holes, etc. Each spatial object registers its coordinate reference system when this information is available. This allows transformations between latitude/longitude and/or various projections.

The **sp** package uses S4 classes, allowing for the validation of internal consistency. All `Spatial` classes in **sp** derive from an abstract class `Spatial`, which only holds a bounding box and the projection or coordinate reference system. All classes are S4 classes, so objects may be created by function `new`, but we typically envisage the use of helper functions to create instances. For instance, to create a `SpatialPointsDataFrame` object from the `meuse` data.frame, we might use:

```
> library(sp)
> data(meuse)
> coords = SpatialPoints(meuse[c("x", "y")])
```

data type	class	attributes	extends
points	SpatialPoints	none	Spatial*
points	SpatialPointsDataFrame	AttributeList	SpatialPoints*
pixels	SpatialPixels	none	SpatialPoints*
pixels	SpatialPixelsDataFrame	AttributeList	SpatialPixels*
			SpatialPointsDataFrame**
full grid	SpatialGrid	none	SpatialPixels*
full grid	SpatialGridDataFrame	AttributeList	SpatialGrid*
line	Line	none	
lines	Lines	none	Line list
lines	SpatialLines	none	Spatial*, Lines list
lines	SpatialLinesDataFrame	data.frame	SpatialLines*
polygon	Polygon	none	Line*
polygons	Polygons	none	Polygon list
polygons	SpatialPolygons	none	Spatial*, Polygons list
polygons	SpatialPolygonsDataFrame	data.frame	SpatialPolygons*

* by direct extension; ** by setIs() relationship;

Table 1: Overview of the spatial classes provided by **sp**. Classes with topology only are extended by classes with attributes.

```
> meuse = SpatialPointsDataFrame(coords, meuse)
> plot(meuse, pch=1, cex = .05*sqrt(meuse$zinc))
```

the plot of which is shown in figure 1.

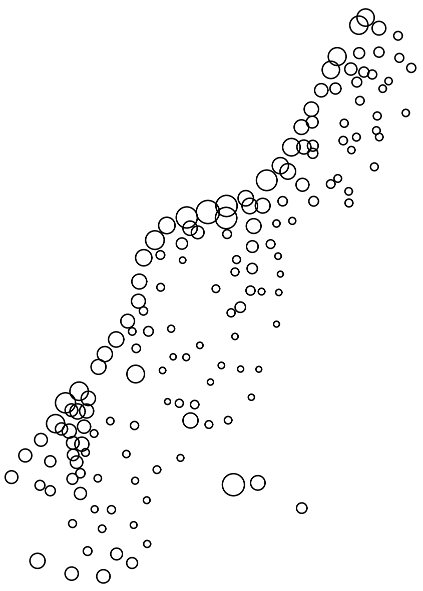

Figure 1: Bubble plot of top soil zinc concentration

The function `SpatialPoints()` creates a `SpatialPoints` object. `SpatialPointsDataFrame()` merges this object with an attribute table to creates an object of class `SpatialPointsDataFrame`.

An alternative to the calls to `SpatialPoints()` and `SpatialPointsDataFrame()` above is to use `coordinates<-`, as in

```
> coordinates(meuse) = c("x", "y")
```

Reading polygons from a shapefile (a commonly used format by GIS) can be done by the `readShapePoly` function in **maptools**, which depends on **sp**. It returns an object of class `SpatialPolygonsDataFrame`, for which the `plot` method is in **sp**. An example that uses a shapefile provided by package **maptools** is:

```
> library(maptools)
> fname = system.file("shapes/sids.shp",
+    package="maptools")
> p4s = CRS("+proj=longlat +datum=NAD27")
> nc = readShapePoly(fname, proj4string=p4s)
> plot(nc, axes = TRUE, col=grey(1-nc$SID79/57))
```

for which the plot is shown in figure 2. The function `CRS` defines the coordinate reference system.

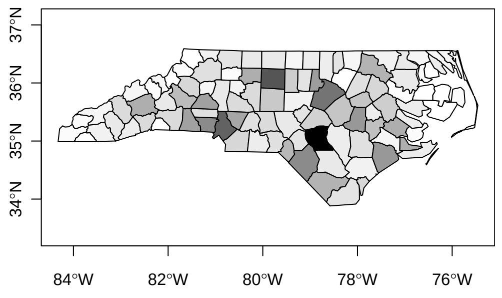

Figure 2: North Carolina sudden infant death (SID) cases in 1979 (white 0, black 57)

An overview of the classes for spatial data in **sp** is given in Table 1. It shows that the points, pixels and grid classes are highly related. We decided to implement two classes for gridded data: `SpatialPixels` for unordered points on a grid, `SpatialGrid` for a full ordered grid, e.g. needed by the `image` command. `SpatialPixels` store coordinates but may be efficient when a large proportion of the bounding box has missing values because it is outside the study area ('sparse' images).

A single `Polygons` object may consist of multiple `Polygon` elements, which could be islands, or holes, or islands in holes, etc. Similarly, a `Lines` entity may have multiple `Line` segments: think of contour lines as an example.

This table also shows that several classes store their attributes in objects of class `AttributeList`, instead of `data.frames`, as the class name would suggest. An `AttributeList` acts mostly like a `data.frame`, but stores no `row.names`, as row names take long to compute and use much memory to store. Consider the creation of a 1000 × 1000 grid:

```
> n = 1000
> df = data.frame(expand.grid(x=1:n, y=1:n),
+     z=rnorm(n * n))
> object.size(df)
[1] 56000556 # mostly row.names!
> library(sp)
> coordinates(df) = ~x+y
> object.size(df)
[1] 16002296 # class SpatialPointsDataFrame
> gridded(df) = TRUE
> object.size(df)
[1] 20003520 # class SpatialPixelsDataFrame
> fullgrid(df) = TRUE
> object.size(df)
[1] 8003468 # class SpatialGridDataFrame
```

Using `data.frame` as attribute tables for moderately sized grids (e.g. Europe on a 1 km × 1 km grid) became too resource intensive on a computer with 2 Gb RAM, and we decided to leave this path.

For classes `SpatialLinesDataFrame` and `SpatialPolygonsDataFrame` we expect that the spatial entities corresponding to each row in the attribute table dominate memory usage; for the other classes it is reversed. If row names are needed in a points or gridded data structure, they must be stored as a (character) attribute column.

`SpatialPolygons` and `SpatialLines` objects have IDs stored for each entity. For these classes, the ID is matched to attribute table IDs when a `Spatial*DataFrame` object is created. For points, IDs are matched when present upon creation. If the matrix of point coordinates used in creating a `SpatialPointsDataFrame` object has rownames, these are matched against the data frame row.names, and subsequently discarded. For points and both grid classes IDs are not stored because, for many points, this wastes processing time and memory. ID matching is an issue for lines and polygons when topology and attributes often come from different sources (objects, files). Points and grids are usually created from a single table, matrix, or array.

Methods

Beyond the standard `print`, `plot` and `summary` methods, methods that **sp** provides for each of the classes are shown in table 2.

Some of these methods are much for user convenience, to make the spatial data objects behave just like `data.frame` objects where possible; others are specific for spatial topology. For example, the overlay function can be used to retrieve the polygon information on point locations, or to retrieve the information for a collection of points falling inside each of the polygons. Alternatively, points and grid cells may be overlayed, resulting in a point-in-gridcell operation. Other overlay methods may be implemented when they seem useful. Method `summary` shows the data class, bounding box, projection information, number of spatial entities, and a summary of the attribute table if present.

Trellis maps

When plotting maps one usually needs to add spatial reference elements such as administrative boundaries, coastlines, cities, rivers, scale bars or north arrows. Traditional plots allow the user to add components to a plot, by using `lines`, `points`, `polygon` or `text` commands, or by using `add=TRUE` in `plot` or `image`. Package **sp** provides traditional plot methods for e.g. `points`, `lines` and `image`, but has in addition an `spplot` command which lets you plot Trellis plots (provided by the **lattice** package) to produce plots with multiple maps. Method `spplot` allows for addition of spatial elements and reference elements on all or a selection of the panels.

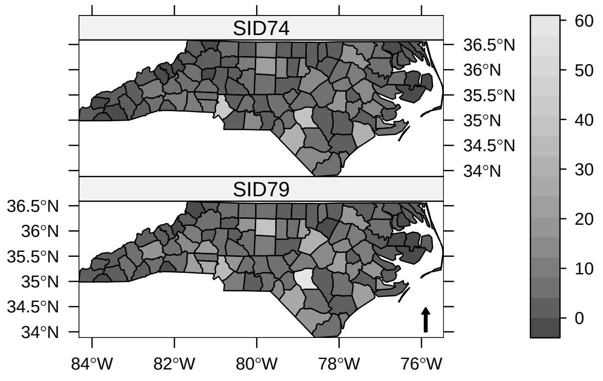

Figure 3: Trellis graph created by `spplot`

The example in figure 3 is created by:

```
> arrow = list("SpatialPolygonsRescale",
+   layout.north.arrow(2),
+   offset = c(-76,34), scale = 0.5, which=2)
> spplot(nc, c("SID74","SID79"),as.table=TRUE,
```

method	what it does
`[`	select spatial items (points, lines, polygons, or rows/cols from a grid) and/or attributes variables
`$`, `$<-`, `[[`, `[[<-`	retrieve, set or add attribute table columns
`spsample`	sample points from a set of polygons, on a set of lines or from a gridded area, using the simple sampling methods given in Ripley (1981)
`spplot`	lattice (Trellis) plots of spatial variables (figure 3; see text)
`bbox`	give the bounding box
`proj4string`	get or set the projection (coordinate reference system)
`coordinates`	set or retrieve coordinates
`polygons`	set or retrieve polygons
`gridded`	verify whether an object is a grid, or convert to a gridded format
`dimensions`	get the number of spatial dimensions
`coerce`	convert from one class to another
`transform`	(re-)project spatial coordinates (uses spproj)
`overlay`	combine two different spatial objects (see text)
`recenter`	shift or re-center geographical coordinates for a Pacific view

Table 2: Methods provided by package **sp**

```
+   scales=list(draw=T), sp.layout = arrow)
```

where the `arrow` object and `sp.layout` argument ensure the placement of a north arrow in the second panel only.

Connections to GIS

On the R-spatial web site (see below) a number of additional packages are available, linking **sp** to several external libraries or GIS:

spproj allows (re)projection of **sp** objects,

spgpc allows polygon clipping with **sp** objects, and can e.g. check whether polygons have holes,

spgrass6 read and write **sp** objects from and to a GRASS 6.0 data base,

spgdal read and write **sp** objects from gdal data sets: it provides a `"["` method that reads in (part of) a gdal data and returns an **sp** grid object.

It is not at present likely that a single package for interfacing external file and data source formats like the recommended package **foreign** will emerge, because of large and varying external software dependencies.

Connections to R packages

CRAN packages maintained by the **sp** authors do already use **sp** classes: package **maptools** has code to read shapefiles and Arc ASCII grids into **sp** class objects, and to write **sp** class objects to shapefiles or Arc ASCII grids. Package **gstat** can deal with points and grids, and will compute irregular block kriging estimates when a polygons object is provided as newdata (prediction 'regions').

The **splancs**, **DCluster**, and **spdep** packages are being provided with ways of handling **sp** objects for analysis, and **RArcInfo** for reading ArcInfo v. 7 GIS binary vector E00 files.

Interface packages present which convert to or from some of the data structures adopted by spatial statistics packages include **spPBS**, **spmaps**; packages under development are **spspatstat**, **spgeoR**, **spfields** and **spRandomFields**. These interface packages depend on both **sp** and the package they interface to, and provide methods to deal with **sp** objects in the target functions. Whether to introduce dependence on **sp** into packages is of course up to authors and maintainers. The interface framework, with easy installation from the R-spatial repository on SourceForge (see below), is intended to help users who need **sp** facilities in packages that do not use **sp** objects directly — for example creating an observation window for **spatstat** from polygons read from a shapefile.

Further information and future

The R-spatial home page is

`http://r-spatial.sourceforge.net/`

Announcements related to **sp** and interface packages are sent to the R-sig-geo mailing list.

A first point for obtaining more information on the classes and methods in **sp** is the package vignette. There is also a demo, obtained by

```
> library(sp)
> demo(gallery)
```

which gives the plots of the gallery also present at the R-spatial home page.

Development versions of **sp** and related packages are on cvs on sourceforge, as well as interface packages that are not (yet) on CRAN. An off-CRAN package repository with source packages and Windows binary packages is available from sourceforge as well, so that the package installation should be sufficiently convenient before draft packages reach CRAN:

```
> rSpatial = "http://r-spatial.sourceforge.net/R"
> install.packages("spproj", repos=rSpatial)
```

Acknowledgements

The authors want to thank Barry Rowlingson (whose ideas inspired many of the methods now in **sp**, see Rowlingson et al. (2003)), and Virgilio Gómez-Rubio.

Bibliography

A. Baddeley and R. Turner. Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005. URL: www.jstatsoft.org, ISSN: 1548-7660. 9

R. Bivand. Interfacing GRASS 6 and R. *GRASS Newsletter*, 3:11–16, June 2005. ISSN 1614-8746. 9

R. Bivand. Using the R statistical data analysis language on GRASS 5.0 GIS data base files. *Computers & Geosciences*, 26:1043–1052, 2000. 9

R. Bivand and A. Gebhardt. Implementing functions for spatial statistical analysis using the R language. *Journal of Geographical Systems*, 3(2):307–317, 2000. 9

V. Gómez-Rubio and A. López-Quílez. RArcInfo: using GIS data with R. *Computers & Geosciences*, 31: 1000–1006, 2005. 9

E. Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30:683–691, 2004. 9

P. Ribeiro and P. Diggle. geoR: A package for geostatistical analysis. *R-News*, 1(2), 2001. URL: www.r-project.org, ISSN: 1609-3631. 9

B. Ripley. Spatial statistics in R. *R-News*, 1(2), 2001. URL: www.r-project.org, ISSN: 1609-3631. 9

B. Ripley. *Spatial Statistics*. Wiley, New York, 1981. 12

B. Rowlingson, A. Baddeley, R. Turner, and P. Diggle. Rasp: A package for spatial statistics. In A. Z. K. Hornik, F. Leisch, editor, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20–22, Vienna, Austria.*, 2003. ISSN 1609-395X. 13

Running Long R Jobs with Condor DAG

by Xianhong Xie

For statistical computing, R and S-Plus have been around for quite some time. As our computational needs increase, the running time of our programs can get longer and longer. A simulation that runs for more than 12 hours is not uncommon these days. The situation gets worse with the need to run the program many times. If one has the access to a cluster of machines, he or she will definitely want to submit the programs to different machines, the management of the submitted jobs could prove to be quite challenging. Luckily, we have batch systems like Condor (http://www.cs.wisc.edu/condor), PBS (http://pbs.mrj.com/service.html), etc. available.

What is Condor?

Condor is a batch processing system for clusters of machines. It works not only on dedicated servers for running jobs, but also on non-dedicated machines, such as PCs. Some other popular batch processing systems include PBS, which has similar features as Condor. The results presented in this article should translate to PBS without much difficulties. But Condor provides some extra feature called checkpointing, which I will discuss later.

As a batch system, Condor can be divided into two parts: job management and resource management. The first part serves the users who have some jobs to run. And the second part serves the machines that have some computing power to offer. Condor matches the jobs and machines through the ClassAd mechanism which, as indicated by its name, works like the classified ads in the newspapers.

The machine in the Condor system could be a computer sitting on somebody else's desk. It serves not only the Condor jobs but also the primary user of the computer. Whenever someone begins using the keyboard or mouse, the Condor job running on the computer will get preempted or suspended depending on how the policy is set. For the first case, the memory image of the evicted job can be saved to some servers so that the job can resume from where it left off once a machine is available. Or the image is just discarded, and the job has to start all over again after a machine is available. The saving/resuming mechanism is called checkpointing, which is only available with the Condor Standard universe jobs. To be able to use this feature, one must have access to the objective codes of the jobs, and relink the objectives against the Condor library. There are some restrictions on what is allowed in one's source codes.

Another popular Condor universe is the Vanilla universe, under which most batch ready jobs should be able to run. One doesn't have to have access to the objective codes, nor is relinking involved. The restrictions on what one can do in one's programs are much less too. Anything that runs under the Standard universe should be able to run under the Vanilla universe, but one loses the nice feature of checkpointing.

Condor supports parallel computing applications with 2 universes called PVM and MPI. In addition, it has one universe for handling jobs with dependencies, one universe that extends Condor to grid computing, and one universe specifically for Java applications.

Condor in Action

The typical steps for running a Condor job are: preparing the programs to make them batch ready, creating Condor submit file(s), submitting the jobs to Condor, checking the results. An example is given in the following.

Suppose we have to run some simulations; each of them accepts one parameter `i` which controls how many iterations needs to be done. To distribute these simulations to different machines and to make the management of jobs easier, one could consider using Condor. The Condor submit file (named `Rjobs.submit`) for the jobs contains

```
Universe        = vanilla
Executable      = /path/to/R
Getenv          = true
Arguments       = --vanilla
Input           = foo.R
Error           = foo.err
Log             = foo.log

Environment     = i=20
Output          = foo_1.out
Queue

Environment     = i=25
Output          = foo_2.out
Queue

Environment     = i=30
Output          = foo_3.out
Queue
```

Note in the submit file, there are 2 occurrences of `vanilla`. The first one is the Condor universe under which the R jobs run. The second occurrence tells R not to load the initialization files and not to save the '.RData' file, among other things. We let all the jobs shared the same log file and error file to reduce the number of files generated by Condor. To pass different inputs to the R code, we used one environmental variables `i`, which can be retrieved in R with `i <- as.numeric(Sys.getenv("i"))`. To submit the jobs to Condor, one can do

```
% condor_submit Rjobs.submit
Submitting job(s)...
Logging submit event(s)...
3 job(s) submitted to cluster 6.
```

The output shows 3 jobs were submitted to one cluster, the submit events being logged in the given log file. To check the Condor queue, one can do

```
% condor_q

-- Submitter: gaia.stat.wisc.edu : <128.105.5.24:34459> : ...
 ID      OWNER    SUBMITTED    RUN_TIME ST PRI SIZE CMD
   6.0   xie     5/17 15:27  0+00:00:00 I  0   0.0  R --vanilla
   6.1   xie     5/17 15:27  0+00:00:00 I  0   0.0  R --vanilla
   6.2   xie     5/17 15:27  0+00:00:00 I  0   0.0  R --vanilla

3 jobs; 3 idle, 0 running, 0 held
```

The jobs are all idle at first. When they are finished, the Condor queue will become empty and one can check the results of the jobs.

Why Use Condor DAG?

Due to the restrictions of Condor, there is no Condor Standard universe version of R. This means R can only be run under the Vanilla universe of Condor. In the scenario described before, long-running Vanilla jobs could be evicted many times without making any progress. But when we divide the big jobs into smaller ones (say 3 four-hour jobs instead of one 12-hour job), there is a much better chance that all the smaller jobs will finish. Since the smaller jobs are parts of the big job, there are usually some dependencies between them. The Condor Directed Acyclic Graph (DAG) system was created to handle Condor jobs with dependencies.

Brief Description of Condor DAG

In a Condor DAG, each node represents a Condor job, which is specified by a Condor submit file. The whole graph is specified by a DAG file, where the lists of the nodes, relationships between the nodes, macro variables for each node, and other things are described. When a DAG job is submitted to Condor, Condor deploys the parent jobs in the DAG first; a child job is not deployed until all its parents have been finished. Condor does this for all the jobs in the DAG file.

An Example of R and Condor DAG

Here is a fake example in which a big R job is divided into 3 parts. We suppose they can only be run sequentially. And we want to replicate the sequence 100 times. The Condor DAG file will look like this

```
Job  A1  Rjob_step1.submit
Job  B1  Rjob_step2.submit
Job  C1  Rjob_step3.submit
Job  A2  Rjob_step1.submit
Job  B2  Rjob_step2.submit
Job  C2  Rjob_step3.submit
...
Parent  A1  Child  B1
Parent  B1  Child  C1
...
Vars  A1  i="1"
Vars  B1  i="1"
Vars  C1  i="1"
...
```

Notice in the DAG file, we let all the R jobs for the same step share one submit file. And we pass a macro variable `i` to each job to identify which replicate the job belongs to. The submit file for step one of the replicate looks like this

```
Universe     = vanilla
Executable   = /path/to/R
Arguments    = --vanilla
Input        = foo_test_step1.R
Getenv       = true
Environment  = i=$(i)
Error        = foo_test.err
Log          = foo_test.log
Notification = never
Queue
```

In the submit file, we used the macro substitution facility `$(var)` provided by Condor. The environment variable `i` can be retrieved in R as before. This variable is used in generating the names of the files for input and output within the R code. Step 2 of one replicate will read the data generated by step 1 of the same replicate. So is the case with step 3 and step 2 of the same replicate. We set `Notification` to be `never` in the submit file to disable excessive notification from Condor, because we have 100 replicates.

To run the DAG job we just created, we can use the following command

```
condor_submit_dag foo_test.dag
```

say, where 'foo_test.dag' is the name of the DAG file.

To check how the DAG jobs are doing, we use the command `condor_q -dag`. If one wants to remove all the DAG jobs for some reason, he or she can do the following `condor_rm` *dagmanid*, where *dagmanid* is the id of the DAG manager job that controls all the DAG jobs. Condor starts one such job for each DAG job submitted via `condor_submit_dag`.

Generating DAG File with R Script

Notice the DAG file given before has a regular structure. To save the repetitive typing, a little R scripting is in order here. The R code snippet for generating the DAG file is given in the following.

```
con <- file("foo_test.dag", "wt")
for (i in 1:100) {
  cat("Job\tA", i, "\tRjob_step1.submit\n",
      sep="", file=con)
  cat("Job\tB", i, "\tRjob_step2.submit\n",
      sep="", file=con)
  cat("Job\tC", i, "\tRjob_step3.submit\n",
      sep="", file=con)
}
...
close(con)
```

Other Ways of Parallel Computing

In the article, we discussed a way of dividing big R jobs into smaller pieces and distributing independent jobs to different machines. This in principle is parallel computing. Some other popular mechanisms for doing this are PVM and MPI. There are R packages called Rmpi and rpvm, which provide interfaces for MPI and PVM respectively in R. It is unwise to argue which way is better, Condor DAG or Rmpi/rpvm. Our way of doing parallel computing should be considered as an alternative to the existing methods. Condor has 2 universes for PVM jobs and MPI jobs too. But they require the code be written in C or C++.

Conclusion

Condor is a powerful tool for batch processing. DAG is a very nice feature of Condor (especially DAG's potential for parallel computing). To run the statistical computing jobs in R that take very long to finish, we can divide the R jobs into smaller ones and make use of the DAG capability of Condor extensively. The combination of Condor DAG and R makes the managing of R jobs easier. And we can get more long running R jobs done under Condor in a reasonable amount of time.

Thanks

The author would like to thank Dr. Douglas Bates for suggesting that I write this article.

Xianhong Xie
University of Wisconsin-Madison, USA
xie@stat.wisc.edu

rstream: Streams of Random Numbers for Stochastic Simulation

by Pierre L'Ecuyer & Josef Leydold

Requirements for random number generators

Simulation modeling is a very important tool for solving complex real world problems (Law and Kelton, 2000). Crucial steps in a simulation study are (Banks, 1998):

1. problem formulation and model conceptualization;

2. input data analysis;

3. run simulation using streams of (pseudo)random numbers;

4. output data analysis;

5. validation of the model.

R is an excellent tool for Steps 2 and 4 and it would be quite nice if it would provide better support for Step 3 as well. This is of particular interest for investigating complex models that go beyond the standard ones that are implemented as templates in simulation software. One requirement for this purpose is to have good sources of (pseudo)random numbers available in the form of multiple streams that satisfy the following conditions:

- The underlying uniform random number generator must have excellent structural and statistical properties, see e.g. Knuth (1998) or L'Ecuyer (2004) for details.

- The streams must be reproducible. This is required for variance reduction techniques like common variables and antithetic variates, and for testing software.

- The state of a stream should be storable at any time.

- There should be the possibility to get different streams for different runs of the simulation ("seeding the random streams").

- The streams must be "statistically independent" to avoid unintended correlations between different parts of the model, or in parallel computing when each node runs its own random number generator.

- There should be substreams to keep simulations with common random numbers synchronized.

- It should be possible to rerun the entire simulation study with different sources of random numbers. This is necessary to detect possible (although extremely rare) incorrect results caused by interferences between the model and the chosen (kind of) random number generator. These different random number generators should share a common programming interface.

In R (like in most other simulation software or libraries for scientific computing) there is no concept of independent random streams. Instead, one global source of random numbers controlled by the global variables *.Random.seed*, which can be changed via `set.seed` and `RNGkind`, are available. The package **setRNG** tries to simplify the setting of this global random number generator. The functionalities listed above can only be mimicked by an appropriate sequence of `set.seed` calls together with proper management of state variables by the user. This is cumbersome.

Multiple streams can be created by jumping ahead by large numbers of steps in the sequence of a particular random number generator to start each new stream. Advancing the state by a large number of steps requires expertise (and even some investigation of the source code) that a user of a simulation software usually does not have. Just running the generator (without using the output) to jump ahead is too time consuming. Choosing random seeds for the different streams is too dangerous as there is some (although small) probability of overlap or strong correlation between the resulting streams.

A unified object-oriented interface for random streams

It is appropriate to treat random number streams as objects and to have methods to handle these streams and draw random samples. An example of this approach is **RngStreams** by L'Ecuyer et al. (2002). This library provides multiple independent streams and substreams, as well as antithetic variates. It is based on a combined multiple-recursive generator as the underlying "backbone" generator, whose very long period is split into many long independent streams viewed as objects These streams have substreams that can be accessed by a `nextsubstream` method.

We have designed a new interface to random number generators in R by means of S4 classes that treat random streams as objects. The interface is strongly influenced by **RngStreams**. It is implemented in the package **rstream**, available from CRAN. It consists of the following parts:

- Create an instance of an *rstream* object (*seed* is optional):[1]

  ```
  s <- new("rstream.mrg32k3a",
           seed=rep(12345,6))
  ```

  Consecutive calls of the constructor give "statistically independent" random streams. The destructor is called implicitly by the garbage collector.

- Draw a random sample (of size n):

  ```
  x <- rstream.sample(s)
  y <- rstream.sample(s,n=100)
  ```

- Reset the random stream;
  skip to the next substream:

  ```
  rstream.reset(s)
  rstream.nextsubstream(s)
  rstream.resetsubstream(s)
  ```

- Switch to antithetic numbers;
  use increased precision[2];
  read status of flags:

  ```
  rstream.antithetic(s) <- TRUE
  rstream.antithetic(s)
  rstream.incprecision(s) <- TRUE
  rstream.incprecision(s)
  ```

Notice that there is no need for seeding a *particular* random stream. There is a package seed for all streams that are instances of *rstream.mrg32k3a* objects. There is method `rstream.reset` for reseting a random stream. The state of the stream is stored in the object and setting seeds to obtain different streams is extremely difficult or dangerous (when the seeds are chosen at random).

Rstream objects store pointers to the state of the random stream. Thus, simply using the `<-` assignment creates two variables that point to the same stream (like copying an environment results in two variables that point to the same environment). Thus in parallel computing, *rstream* objects cannot be simply copied to a particular node of the computer cluster. Furthermore, *rstream* objects cannot be saved between R sessions. Thus we need additional methods to make independent copies (clones) of the same object and methods to save (pack) and restore (unpack) objects.

- Make an independent copy of the random stream (clone):

  ```
  sc <- rstream.clone(s)
  ```

- Save and restore a stream object (the packed rstream object can be stored and handled like any other R object):

  ```
  rstream.packed(s) <- TRUE
  rstream.packed(s) <- FALSE
  ```

Package **rstream** is designed to handle random number generators in a unified manner. Thus there is a class that deals with the R built-in generators:

- Create *rstream* object for built-in generator:

  ```
  s <- new("rstream.runif")
  ```

Additionally, it is easy to integrate other sources of random number generators (e.g. the generators from the GSL (`http://www.gnu.org/software/gsl/`), SPRNG (see also package **rsprng** on CRAN), or SSJ (`http://www.iro.umontreal.ca/~simardr/ssj/`)) into this framework which then can be used by the same methods. However, not all methods work for all types of generators. A method that fails responds with an error message.

The **rstream** package also interacts with the R random number generator, that is, the active global generator can be transformed into and handled as an *rstream* object and vice versa, every *rstream* object can be set as the global generator in R.

[1]'`rstream.mrg32k3a`' is the name of a particular class of random streams named after its underlying backbone generator. The library **RngStreams** by L'Ecuyer et al. (2002) uses the MRG32k3a multiple recursive generator. For other classes see the manual page of the package.

[2]By default the underlying random number generator used a resolution of 2^{-32} like the R built-in RNGs. This can be increased to 2^{-53} (the precision of the IEEE double format) by combining two consecutive random numbers. Notice that this is done implicitly in R when normal random variates are created via inversion (`RNGkind(normal.kind="Inversion")`), but not for other generation methods. However, it is more transparent when this behavior can be controlled by the user.

- Use stream s as global R uniform RNG:

  ```
  rstream.RNG(s)
  ```

- Store the status of the global R uniform RNG as *rstream* object:

  ```
  gs <- rstream.RNG()
  ```

Simple examples

We give elementary examples that illustrate how to use package **rstream**. The model considered in this section is quite simple and the estimated quantity could be computed with other methods (more accurately).

A discrete-time inventory system

Consider a simple inventory system where the demands for a given product on successive days are independent Poisson random variables with mean λ. If X_j is the stock level at the beginning of day j and D_j is the demand on that day, then there are $\min(D_j, X_j)$ sales, $\max(0, D_j - X_j)$ lost sales, and the stock at the end of the day is $Y_j = \max(0, X_j - D_j)$. There is a revenue c for each sale and a cost h for each unsold item at the end of the day. The inventory is controlled using a (s, S) policy: If $Y_j < s$, order $S - Y_j$ items, otherwise do not order. When an order is made in the evening, with probability p it arrives during the night and can be used for the next day, and with probability $1 - p$ it never arrives (in which case a new order will have to be made the next evening). When the order arrives, there is a fixed cost K plus a marginal cost of k per item. The stock at the beginning of the first day is $X_0 = S$.

We want to simulate this system for m days, for a given set of parameters and a given control policy (s, S), and replicate this simulation n times independently to estimate the expected profit per day over a time horizon of m days. Eventually, we might want to optimize the values of the decision parameters (s, S) via simulation, but we do not do that here. (In practice, this is usually done for more complicated models.)

We implement this model with the following R code. (The implementation is kept as simple as possible. It is not optimized, uses global variables, and we have neglected any error handling.)

```
## load library
library(rstream)

## parameter for inventory system
lambda <- 100  # mean demand size
c      <- 2    # sales price
h      <- 0.1  # inventory cost per item
K      <- 10   # fixed ordering cost
k      <- 1    # marginal ordering cost per item
p      <- 0.95 # probability that order arrives
m      <- 200  # number of days
```

We use two independent sources of randomness for the simulation of the daily demand and for the success of ordering the product.

```
## initialize streams of random numbers
gendemand <- new("rstream.mrg32k3a")
genorder  <- new("rstream.mrg32k3a")
```

Notice that the following R code need not be changed at all if we replace `rstream.mrg32k3a` by a different source of random numbers that provides the required functionality.

For the simulation of daily sales we need a generator for (truncated) Poisson distributions. Currently a system for invoking generators for nonuniform distributions that use particular random number generators similar to the **rstream** package does not exist. Developing such a package is the next step. Meanwhile we use the following simple (and slow!) generator which is based on the inversion method (Devroye, 1986, §X.3.3).

```
## simple generator for Poisson distribution that
## uses uniform random numbers from stream 'rs'
randpoisson <- function (lambda,rs) {
   X <- 0
   sum <- exp(-lambda)
   prod <- exp(-lambda)
   U <- rstream.sample(rs,1)
   while (U > sum) {
      X <- X + 1
      prod <- prod * lambda / X
      sum <- sum + prod
   }
   return (X)
}
```

The last brickstone is a routine that computes a realization of the the average profit for a given control policy (s, S).

```
simulateOneRun <- function (s,S) {
   X <- S; profit <- 0
   for (j in 1:m) {
      sales <- randpoisson(lambda,gendemand)
      Y <- max(0,X-sales)
      profit <- 0
      if (Y < s && rstream.sample(genorder,1)<p) {
         profit <- profit - (K + k * (S-Y))
         X <- S }
      else {
         X <- Y }
   }
   return (profit/m)
}
```

Now we can perform 100 independent simulation runs for control policy $(s, S) = (80, 200)$ and compute the 95% confidence interval for the average daily sales.

```
result <- replicate(100,simulateOneRun(80,200))
t.test(result,conf.level=0.95)$conf.int

## confidence interval:
[1] 85.02457 85.43686
attr(,"conf.level")
[1] 0.95
```

Common random numbers

In the second example we want to compare the control policy $(s_0, S_0) = (80, 200)$ with policy $(s_1, S_1) = (80, 198)$ in our simple inventory model.

```
resultdiff <- replicate(100,
   simulateOneRun(80,200)-simulateOneRun(80,198) )
t.test(resultdiff,conf.level=0.95)$conf.int

## confidence interval:
[1] -0.3352277  0.2723377
attr(,"conf.level")
[1] 0.95
```

The variance of this estimator can be reduced by using the technique of common random numbers. We have to make sure that for both simulations the same sequence of random numbers is used in the same way. Thus we have to keep the random streams used for demand and ordering synchronized by means of reset calls.

```
## reset streams
rstream.reset(gendemand)
rstream.reset(genorder)

resultdiffCRN <- replicate(100,
     {
      ## skip to beginning of next substream
      rstream.nextsubstream(gendemand);
      rstream.nextsubstream(genorder);
      simulateOneRun(80,200)}
   - {
      ## reset to beginning of current substream
      rstream.resetsubstream(gendemand);
      rstream.resetsubstream(genorder);
      simulateOneRun(80,198)} )

t.test(resultdiffCRN,conf.level=0.95)$conf.int

## confidence interval:
[1] 0.2354436 0.3717864
attr(,"conf.level")
[1] 0.95
```

The confidence interval is much narrower now than with independent random numbers.

Parallel computing

When stochastic simulations are run in parallel, it is crucial that the random numbers generated on each of the nodes in the computing environment are independent. The class `rstream.mrg32k3a` is perfectly suited for this task in a master/slave design.

```
## Master node
## create independent streams for each node
stream1 <- new("rstream.mrg32k3a")
rstream.packed(stream1) <- TRUE
stream2 <- new("rstream.mrg32k3a")
rstream.packed(stream2) <- TRUE

## Slave node
rstream.packed(stream1) <- FALSE
X <- rstream.sample(stream1,1);
```

Remark

Notice, that this "negative functionality" of un/packing the *rstream* object should be hidden in some user interface for parallel computing, e.g., **snow** (Simple Network of Workstations). This un/packing mechanism could be avoided when the state of the random stream is always stored inside R (instead of using external pointers). Package **rlecuyer** (version 0.1) implements access to the **RngStreams** library in such a way. However, we found out that this approach is rather slow. On our computer (Intel Pentium 4, 2.6 GHz, Linux 2.6.12, R version 2.1.1) the marginal generation time for a single random number is 10 times slower than with our approach (and 250 times slower for generating a random vector of size 100).

Adding uniform random number generators in R

In our opinion the proposed interface can serve as a brickstone of simulation packages based on the R environment. With this design, uniform random number generators can be used as arguments to subroutines. Thus random number generators can be easily exchanged or used in parallel. An extension for nonuniform random numbers is the next step towards a flexible and easy-to-use simulation environment in R.

There is one drawback in the implementation. R uses a static table to implement access to different kinds of random number generators. Only one slot (`RNGkind(kind="user-supplied")`) has been provided for users who want to add her own generator. This slot relies on a pointer to a function called `user_unif_rand`. Thus **rstream** has to abuse this slot to add additional random number generators. However, there are other packages on CRAN that use the same trick (**randaes**, **rlecuyer**, **rsprng**, **SuppDists**) or load one of these packages (**Rmpi**, **rpvm**, **rpart**, **snow**, **varSelRF**). Thus if a user loads two of these packages or adds her own uniform random number generator, the behavior of at least one of the packages is broken. This problem could be fixed by some changes in the R core system, either by adding a *package* variable to `RNGkind(kind="user-supplied")` similar to the `.Call` function, or by replacing the static table by a dynamic one.

Acknowledgments

This work was supported by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung*, Proj.No. P16767-N12 and the Natural Sciences and Engineering Research Council of Canada, Grant Number ODGP0110050.

Bibliography

J. Banks, editor. *Handbook of Simulation*. Wiley, New York, 1998. 16

L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New-York, 1986. 18

D. E. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1998. 16

A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 3rd edition, 2000. 16

P. L'Ecuyer. Random number generation. In J. E. Gentle, W. Haerdle, and Y. Mori, editors, *Handbook of Computational Statistics*, chapter II.2, pages 35–70. Springer-Verlag, Berrlin, 2004. 16

P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075, 2002. 17

mfp

Multivariable Fractional Polynomials

by Axel Benner

Introduction

The `mfp` package is targeted at the use of multivariable fractional polynomials for modelling the influence of continuous, categorical and binary covariates on the outcome in regression models, as introduced by Royston and Altman (1994) and modified by Sauerbrei and Royston (1999). It combines backward elimination with a systematic search for a 'suitable' transformation to represent the influence of each continuous covariate on the outcome. The stability of the models selected was investigated in Royston and Sauerbrei (2003). Briefly, fractional polynomials models are useful when one wishes to preserve the continuous nature of the covariates in a regression model, but suspects that some or all of the relationships may be non-linear. At each step of a 'backfitting' algorithm `mfp` constructs a fractional polynomial transformation for each continuous covariate while fixing the current functional forms of the other covariates. The algorithm terminates when no covariate has to be eliminated and the functional forms of the continuous covariates do not change anymore.

In regression models often decisions on the selection and the functional form of covariates must be taken. Although other choices are possible, we propose to use P-values to select models. With many covariates in the model, a large nominal P-value will lead to substantial overfitting. Most often a nominal P-value of 5% is used for variable selection.

The choice of a 'selection' P-value depends on the number of covariates and whether hypothesis generation is an important aim of the study. The `mfp` procedure therefore allows to distinguish between covariates of main interest and confounders by specifying different selection levels for different covariates.

Fractional Polynomials

Suppose that we have an outcome variable, a single continuous covariate, Z, and a suitable regression model relating them. Our starting point is the straight line model, $\beta_1 Z$ (for easier notation we ignore the intercept, β_0). Often this assumption is an adequate description of the functional relationship, but other functional forms must be investigated for possible improvements in model fit. A simple extension of the straight line is a power transformation model, $\beta_1 Z^p$. This model has often been used by practitioners in an *ad hoc* way, utilising different choices of p. We formalise the model slightly by calling it a first-degree fractional polynomial or FP1 function where the powers p are chosen from a restricted set, S (Royston and Altman, 1994). For pragmatic reasons, Royston and Altman (1994) suggested $S = \{-2, -1, -0.5, 0, 0.5, 1, 2, 3\}$, where Z^0 denotes $\log(Z)$. No subsequent changes to S have been proposed since then. The power transformation set includes the 'identiy' transformation $(p = 1)$, as well as the reciprocal, logarithmic, square root and square transformations. To fit an FP1 model each of the eight values of p is tried. The best-fitting model is defined as the one with the maximum likelihood among these models.

Extension from one-term FP1 functions to the more complex and flexible two-term FP2 functions follows immediately. FP2 functions with powers (p_1, p_2) include the formulations $\beta_1 Z^{p_1} + \beta_2 Z^{p_2}$ and $\beta_1 Z^{p_1} + \beta_2 Z^{p_1} \log(Z)$ if $p_2 = p_1$, the latter being the so-called repeated-powers functions. The best fit among the 36 (28 with $p_2 \neq p_1$ and 8 with $p_2 = p_1$) combinations of powers from S is defined as that with the highest likelihood.

An FPm model of degree m, $m = 1, 2$, is considered to have $2m$ degrees of freedom (d.f.), 1 d.f. for each parameter and 1 d.f. for each power. Because a restricted number of powers (only those from the set S) are used, the value $2m$ is a slight over-estimate of the effective number of d.f. (Royston and Altman, 1994). Defining the 'deviance' of a FP model as minus twice the maximised log likelihood, statistical significance between FP models is assessed by using the χ^2 distribution.

Model selection

For choosing among FP models at a given significance level of α, two procedures have been proposed in the literature, a sequential selection procedure and a closed testing selection procedure.

For the sequential selection procedure first a 2 d.f. test at level α of the best-fitting second-degree FP, FP2, against the best-fitting first-degree FP, FP1, is performed. If the test is significant, the final model is the best-fitting FP2. If the test is not significant, a 1 d.f. test at the α level of the best fitting FP1 against a straight line follows. If the test is significant, the final model is the best-fitting FP1. Otherwise, a 1 d.f. test at the α level of a straight line against the model omitting the covariate is performed. If the test is significant, the final model is a straight line, otherwise the covariate is omitted.

The closed testing selection procedure (denoted `RA2`) was originally described in Ambler and Royston (2001). It maintains approximately the correct Type I error rate for each covariate tested.

For a specific covariate X the `RA2` algorithm works as follows:

1. Perform a 4 d.f. test at the α level of the best-fitting FP2 against the null model. If the test is not significant, drop X and stop, otherwise continue.

2. Perform a 3 df test at the α level of the best-fitting FP2 against a straight line. If the test is not significant, stop (the final model is a straight line), otherwise continue.

3. Perform a 2 df test at the α level of the best-fitting FP2 against the best-fitting FP1. If the test is significant, the final model is the best-fitting FP2, otherwise the best-fitting FP1.

For the sequential selection procedure the actual Type I error rate may exceed the nominal value when the true relationship is a straight line. The procedure tends to favour more complex models over simple ones. For this reason the R implementation provides only the `RA2` procedure.

Multivariable Fractional Polynomials

Modelling the joint effect of several covariates one may wish to simplify the model, either by dropping non-significant variables and/or by reducing the complexity of FP functions fitted to continuous covariates. A solution to finding a model including FP terms was proposed by Royston and Altman (1994) and refined by Sauerbrei and Royston (1999). Instead of an exhaustive search an iterative algorithm was recommended and termed the `MFP` procedure. It combines backward elimination of variables with a search for the best FP functions of continuous covariates.

The order of covariates for the variable selection procedure is determined by fitting a linear model including all variables and dropping each variable singly according to the first step of a conventional backward elimination procedure. The values of the corresponding test statistics are then used to order the variables, from the most to the least 'significant'. Having imposed an order, each covariate is considered in turn. If a covariate is categorical or binary, a standard likelihood ratio test is performed to determine whether it should be dropped or not. If a covariate is continuous, the FP model selection procedure `RA2` is applied to determine the best FP or to eliminate the covariate. The procedure cycles through the variables in the predetermined order until the selected variables and FP functions do not change any more.

Usage

A typical `mfp` model formula has the form `response` $\sim$ `terms` where `response` is the (numeric) response vector and `terms` is a series of terms, usually separated by + operators, which specifies a linear predictor for `response` and provided by the `formula` argument of the function call. Fractional polynomial terms are indicated by `fp`.

For `binomial` models the response can also be specified as a `factor`. If a Cox proportional hazards model is required then the outcome need to be a survival object specified using the `Surv()` notation.

The argument `family` describes the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. Actually linear, logistic, Poisson and Cox regression models have been implemented.

The global argument `alpha` (default: 0.05) sets the FP selection level to be used for all covariates. The global variable selection level is set by argument `select` (default: 1, corresponding to 'no variable selection'). These values can be changed for individual covariates by using the `fp` function in the formula.

The `fp` function in the `formula` description of the model defines a fractional polynomial object for a single input variable using arguments `df`, `select`, `alpha` and `scale` for FP model selection. The `df` argument determines the maximum degree of the FP model to be tested for the corresponding covariate by its d.f. (default: 4, corresponding to an FP2 model).

The `scale` argument (default: TRUE) of the `fp` function denotes the use of pre-transformation scaling to avoid possible numerical problems.

Example

As an example we use the data from a clinical trial for patients with node positive breast cancer from the German Breast Cancer Study Group (GBSG). A total of 7 covariates were investigated. Complete data on covariates and the survival times is available for 686 patients of the study. During a median follow-up of about 5 years 299 events were observed for recurrence free survival time (`rfst`). For more details see Sauerbrei and Royston (1999) or references given there. The results are computed using the `mfp` package, version 1.3.1.

Cox proportional hazards model

The dataset `GBSG` which contains the data for the set of 686 patients with node-positive breast cancer is provided as example data set in the `mfp` package.

```
> data(GBSG)
```

The response variable is recurrence free survival time (`Surv(rfst, cens)`). In the example data set seven possible prognostic factors were given, of which 5 are continuous, age of the patients in years (`age`), tumor size in mm (`tumsize`), number of positive lymph nodes (`posnodal`), progesterone receptor in fmol (`prm`), estrogen receptor in fmol (`esm`). One more covariate is binary, menopausal status (`menostat`), and one is ordered categorical with three levels, tumor grade (`tumgrad`). Finally, the model has to be adjusted for hormonal therapy (`htreat`).

According to Sauerbrei and Royston (1999) a pre-transformation of the number of positive nodes was applied,

```
> GBSG$nodetrans <- exp(-0.12 * GBSG$posnodal)
```

In addition a value of 1 was added to `prm` and `esm` in the example data set GBSG to obtain positive values for these variables. To be compatible with other applications of mfp on this data set (cp. Sauerbrei et al., 2005) the data for age were divided by 50 and tumor grade levels 2 and 3 were combined.

```
> GBSG$age50 <- GBSG$age/50
> levels(GBSG$tumgrad) <-
    list("1"=1,"2 or 3"=c(2,3))
```

A Cox proportional hazards regression is the standard approach to model the hazard of tumour recurrence. Therefore the `mfp` model of recurrence free survival on the initial set of 7 covariates, stratified for hormone therapy, is fitted according to a Cox regression model by a call to `mfp()` using family argument `cox`. The global FP selection level `alpha` as well as the global variable selection level `select` are set to 5%.

```
 > f <- mfp(Surv(rfst, cens) ~  fp(age50)
     +fp(nodetrans)+fp(prm)+fp(esm)+fp(tumsize)
     +menostat+tumgrad+strata(htreat),
     family = cox, method="breslow", alpha=0.05,
     select=0.05, data = GBSG)
```

To be compatible with other implementations of multivariable fractional polynomials in SAS and Stata we use the Breslow method for tie handling.

Pre-transformation scaling was used for `esm`, `prm` and `tumsize`.

```
 > f$scale

              shift scale
nodetrans         0     1
prm               0   100
tumgrad2 or 3     0     1
tumsize           0    10
menostat2         0     1
age50             0     1
esm               0   100
```

The final model is given as

```
 > f

Call:
mfp(formula = Surv(rfst, cens) ~ fp(age50) + fp(nodetrans) +
    + fp(prm) + fp(esm) + fp(tumsize) + menostat + tumgrad +
    strata(htreat), data = GBSG,
    family = cox, method = "breslow",
    alpha = 0.05, select = 0.05)


Fractional polynomials:
              df.initial select alpha df.final power1 power2
nodetrans              4   0.05  0.05        1      1      .
prm                    4   0.05  0.05        2    0.5      .
tumgrad2 or 3          1   0.05  0.05        1      1      .
tumsize                4   0.05  0.05        0      .      .
menostat2              1   0.05  0.05        0      .      .
age50                  4   0.05  0.05        4     -2     -1
esm                    4   0.05  0.05        0      .      .


                   coef exp(coef) se(coef)     z       p
nodetrans.1     -1.9777   0.13839   0.2272 -8.70 0.0e+00
prm.1           -0.0572   0.94442   0.0111 -5.16 2.5e-07
tumgrad2 or 3.1  0.5134   1.67092   0.2495  2.06 4.0e-02
age50.1          2.4230  11.27964   0.4753  5.10 3.4e-07
age50.2         -5.3060   0.00496   1.1807 -4.49 7.0e-06

Likelihood ratio test=142  on 5 df, p=0 n= 686
```

Of the possible prognostic covariates `esm`, `menostat` and `tumsize` were excluded from the model (`df.final=0`). For covariates `age` and `prm` nonlinear transformations were chosen (`df.final>1`). Because of the pre-transformation the function used for the number of positive lymph nodes is also non-linear. For `prm` an FP1 was selected with $p = 0.5$ corresponding to a square-root transformation. For `age` an FP2 model was found with $p_1 = -2$, providing a new artificial variable `age.1`, and $p_2 = -1$, resulting in the new artificial variable `age.2`.

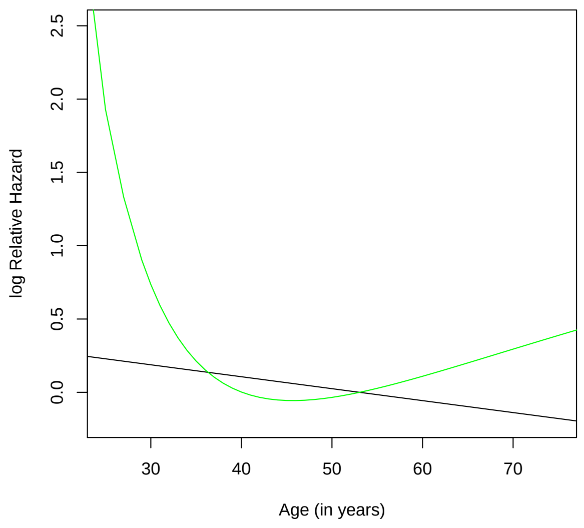

Figure 1: Estimated functional relationship between age and the log relative hazard of recurrence-free survival. The linear function is derived from a standard Cox regression model and is shown as solid black line. The green line shows the functional form as postulated by the `mfp` model. For both functions a constant was subtracted from the estimated function before plotting so that the median value of age yields 'y=0' in both cases.

If one had used a standard Cox regression model with linear functional forms for the continuous covariates the number of positive lymph nodes (`nodetrans`), progesterone receptor (`prm`) and tumor grade (`tumgrad`) would have a statistical significant effect at the 5% level.

The multivariable fractional polynomial Cox regression model additionally selected the FP2 transformations of variable age50 (`age50.1` and `age50.2`). Instead of the untransformed (`prm`) variable an FP1 of `prm` was chosen and three variables were deleted.

In Figure 1 the estimated functional forms for age resulting from the standard Cox regression model and the multivariable fractional polynomial regression model are compared.

The result of the R implementation of `mfp` differs from the one using Stata and SAS by selection of an FP2 transformation with powers -2 and -1 instead of -2 and -0.5 (Sauerbrei et al. (2005)). Using the actual implementation of the Cox regression model in R (function 'coxph' in package "survival", version 2.18) to compare the results of the models including the two different FP2 versions for age50 of R and Stata resulted in nearly the same computed log-likelihoods. The log-likelihood of the model including age50 with powers -2 and -1 is -1528.086 as compared to -1528.107 for the model including age50 with powers -2 and -0.5.

The value `fit` of the resulting mfp object can be used for survival curve estimation of the final model fit (2).

```
pf <- survfit(f$fit)
plot(pf, col=c("red","green"),
  xlab="Time (years)",
  ylab="Recurrence free survival rate",
  xscale=365.25)
```

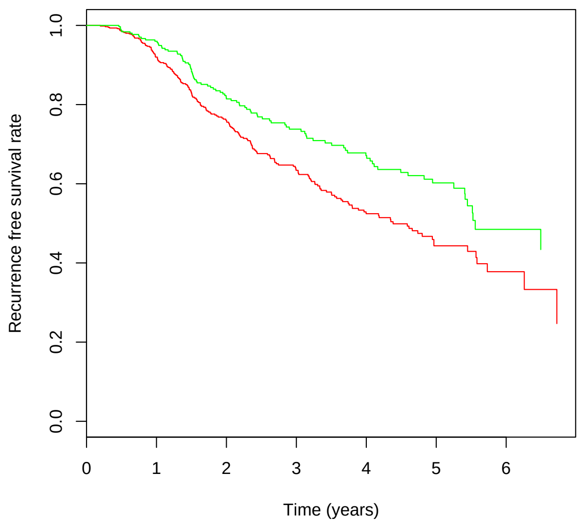

Figure 2: Predicted survival curves of the final mfp model for the two strata defined by hormonal treatment (red line: without hormonal treatment, green line: with hormonal treatment).

Bibliography

G. Ambler and P. Royston. Fractional polynomial model selection procedures: investigation of Type I error rate. *Journal of Statistical Simulation and Computation*, 69:89–108, 2001. 21

P. Royston and D. G. Altman. Regression using fractional polynomials of continuous covariates: parsimonious parametric modelling (with discussion). *Applied Statistics*, 43(3):429–467, 1994. 20, 21

P. Royston and W. Sauerbrei. Stability of multivariable fractional polynomial models with selection of variables and transformations: a bootstrap investigation. *Statistics in Medicine*, 22:639–659, 2003. 20

W. Sauerbrei and P. Royston. Building multivariable prognostic and diagnostic models: transformation of the predictors by using fractional polynomials. *Journal of the Royal Statistical Society (Series A)*, 162: 71–94, 1999. 20, 21, 22

W. Sauerbrei, C. Meier-Hirmer, A. Benner, and P. Royston. Multivariable regression model building by using fractional polynomials: Description of SAS, Stata and R programs. *Computational Statistics and Data Analysis*. Article in press. 22, 23

Axel Benner
German Cancer Research Center - Heidelberg, Germany.
benner@dkfz.de

crossdes

A Package for Design and Randomization in Crossover Studies

by Oliver Sailer

Introduction

Design of experiments for crossover studies requires dealing with possible order and carryover effects that may bias the results of the analysis. One approach to deal with those effects is to use designs balanced for the respective effects. Almost always there are effects unaccounted for in the statistical model that may or may not be of practical importance. An important way of addressing those effects is randomization.

crossdes constructs and randomizes balanced block designs for multiple treatments that are also balanced for carryover effects. Jones and Kenward (1989), Ch. 5 review different crossover designs and cite optimality results. Wakeling and MacFie (1995) promoted the use of balanced designs for sensory studies. Five construction methods described in this paper are implemented here. They include Williams designs (Williams, 1949) and designs based on complete sets of mutually orthogonal latin squares. If one is just interested in getting balanced incomplete block designs that are not balanced for carryover, the package **AlgDesign** may be used (Wheeler, 2004). **crossdes** also contains functions to check given designs for balance.

crossdes is available on CRAN. It requires three packages also available on CRAN: **AlgDesign**, **gtools**, located in package bundle **gregmisc**, and **MASS**, available in bundle **VR**.

Simple Crossover Studies for Multiple Treatments

When comparing the effects of different treatments on experimental units, economical and ethical considerations often limit the number of units to be included in the study. In crossover studies, this restraint is addressed by administering multiple treatments to each subject.

However, the design setup leads to a number of possible problems in such an analysis. In addition to the treatment effects, we have to consider subject effects, order effects and carryover effects. In case we are explicitly interested in estimating those effects, we may fit corresponding mixed effects models. If, however, we are only interested in differences between treatments, we may use balanced designs that average out the nuisance parameters as far as possible.

By far the most common crossover design is the AB/BA design with just two treatments applied in two periods, see e.g. Jones and Kenward (1989) or Senn (1993). Here we restrict our attention to balanced designs for the comparison of three or more treatments and to cases, where each unit receives each treatment at most once. In general these designs are no longer balanced for treatment or carryover effects if some observations are missing. Simulation studies suggest that these designs are fairly robust against small numbers of dropouts, see e.g. Kunert and Sailer (2005). In the next section we explain how to actually get balanced designs in R using **crossdes**.

Design Construction

There are three important parameters that we need to define before we can construct a design: The number of treatments t to compare, the number of periods $p \leq t$, i.e. the number of treatments each subject gets and the (maximum) number of subjects or experimental units n available for the study.

To ways to get balanced designs are implemented. The first approach is to specify one of the five construction methods described below. Unless there is no design for the specified parameters t, p and n, a matrix representing the experimental plan is given. Rows represent subjects and columns represent periods. The design should then be randomized. The function `random.RT` randomizes row and treatment labels.

The function `all.combin` constructs balanced block designs that are balanced for all orders of carryover effects up to $p-1$ based on all possible permutations of treatment orders. The user specifies the number of treatments and periods, t and p. While this approach works for any $t \geq 2$ and $p \leq t$, the fact that every treatment combination occurs leads to very large numbers of subjects required for the study as long as t and p aren't very small, see e.g. Patterson (1952).

As long as t is a prime power, there is a possibility to drastically reduce the number of subjects required and still retain the same properties as described above. The function `des.MOLS` constructs designs based on complete sets of mutually orthogonal latin squares (MOLS) where $t \leq 100$ has to be a prime power and $p \leq t$, see e.g. Williams (1949). The function `MOLS` gives a complete set of $t-1$ MOLS based on Galois Fields that is of use in other applications of combinatorics in the design of experiments as well (Street and Street, 1987). The necessary arguments are r and s, where r is prime and s is a positive integer such that $r^s = t$.

If the number of subjects is still too large or t is not a prime power, one may consider designs that are only balanced for first order carryover effects. If each subject gets each treatment once, Williams designs only use t or $2t$ subjects (Williams, 1949). `williams` gives carryover balanced latin squares for $t \geq 2$.

If the number of treatments is large, it may not be possible for the subjects to receive all treatments. Two construction methods for incomplete designs are implemented in the package. One is `williams.BIB`, which combines balanced incomplete block designs (BIBD) and Williams designs (Patterson, 1951). The user needs to specify a balanced incomplete block design. Such designs may be found using the package **AlgDesign** (Wheeler, 2004). The function `find.BIB` provides a convenient way to use that package to get a BIBD. The last construction function considered here is `balmin.RMD`, a function that constructs the balanced minimal repeated measurements designs of Afsarinejad (1983). These designs are balanced for first order carryover effects but in general, they are not balanced for subject and order effects.

A more convenient way is to use the menu driven function `get.plan` documented below. The user specifies t, p and the maximum number of subjects available. The function checks which of the above functions may work for the given parameters. If there is no design that fits to the parameters, the design parameters may be adapted and a new choice of methods is presented.

For example, we may want to get a design for the comparison of 7 products. Assume that a maximum of 100 test persons are available and they may test all products within one session, i.e. $t = p = 7$ and $n = 100$. We have

```
> design <- get.plan(7,7,100)
```

```
Possible constructions and minimum
numbers of subjects:
          1        2
Method:  williams des.MOLS
Number:  14       42

Please choose one of the following
constructions
1:williams
2:des.MOLS
3:Exit
Selection:
```

`get.plan` suggests to use either a Williams design (which requires 14 subjects only) or to use a set of MOLS, which requires 42 subjects. Assume that we are interested in keeping the number of subjects as low as possible. We choose a williams design and type in the corresponding number:

```
> Selection: 1
williams selected. How many 'replicates'
do you wish (1 - 7 )?
Selection:
```

We choose not to replicate the design since that would mean additional subjects and type in `1`. If we wanted to get closer to the maximum number of subjects available we could have selected a design based on MOLS (for higher order balance) and two replicates of the design. That would have meant 84 subjects.

```
> Selection: 1
1 replicate(s) chosen
Row and treatment labels have been
randomized.
Rows represent subjects, columns
represent periods.
```

As indicated, the design is already randomized. A possible realization of the treatment sequences for the subjects is as follows:

```
> design
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
 [1,]    1    3    4    5    7    6    2
 [2,]    3    5    1    6    4    2    7
 [3,]    5    6    3    2    1    7    4
 [4,]    7    2    4    6    1    5    3
 [5,]    3    1    5    4    6    7    2
 [6,]    1    4    3    7    5    2    6
 [7,]    7    4    2    1    6    3    5
 [8,]    5    3    6    1    2    4    7
 [9,]    2    7    6    4    5    1    3
[10,]    6    2    5    7    3    4    1
[11,]    6    5    2    3    7    1    4
[12,]    4    1    7    3    2    5    6
[13,]    2    6    7    5    4    3    1
[14,]    4    7    1    2    3    6    5
```

Checking for Balance

Experimental plans that fit into the row-column scheme may be checked for balance. The function `isGYD` checks whether a given design is a balanced block design with respect to rows, columns or both rows and columns. The user specifies the matrix that represents the design. The design is then checked for balance. If the argument `tables` is set to `TRUE`, matrices giving the number of occurences of each treatment in each row and column as well as other tables describing the design are given. Continuing the example of the previous section we have

```
>  isGYD(design, tables=TRUE)

[1] The design is a generalized latin
square.

$"Number of occurrences of treatments
in d"
 1  2  3  4  5  6  7
14 14 14 14 14 14 14

$"Row incidence matrix of d"
  1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 1 1 1 1 1 1 1 1 1  1  1  1  1  1

...

7 1 1 1 1 1 1 1 1 1  1  1  1  1  1

$"Column incidence matrix of d"
  1 2 3 4 5 6 7
1 2 2 2 2 2 2 2

...

7 2 2 2 2 2 2 2

$"Concurrence w.r.t. rows"
   1  2  3  4  5  6  7
1 14 14 14 14 14 14 14

...

7 14 14 14 14 14 14 14

$"Concurrence w.r.t. columns"
   1  2  3  4  5  6  7
1 28 28 28 28 28 28 28

...

7 28 28 28 28 28 28 28
```

The design is a generalized latin square which means that it's balanced for subject and period effects. We see for example that each product occurs 14 times within `design`. Each subject gets each product exactly once and each product appears in each period exactly twice.

Applying the function `isCbalanced` we see that our design is also balanced for first order carryover effects. Each product is preceded by every other product exactly twice but never by itself.

```
> isCbalanced(design)

The design is (first order) carry-over
balanced.

Left neighbour incidence matrix M_ij
(i is left neighbour of j)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0    2    2    2    2    2    2
[2,]    2    0    2    2    2    2    2
[3,]    2    2    0    2    2    2    2
[4,]    2    2    2    0    2    2    2
[5,]    2    2    2    2    0    2    2
[6,]    2    2    2    2    2    0    2
[7,]    2    2    2    2    2    2    0
```

Summary

The package **crossdes** implements methods for the construction of balanced crossover designs. Block designs may be randomized and checked for balance. For details on the functions described above the user is referred to the package manual.

Acknowledgements

Support by the Deutsche Forschungsgemeischaft (SFB 475, Komplexitätsreduktion in multivariaten Datenstrukturen) is gratefully acknowledged.

Bibliography

Afsarinejad, K. (1983) Balanced repeated measurements designs. *Biometrika* **70**, 199–204. 25

Jones, B. and Kenward, M.G. (1989) *Design and Analysis of Cross-Over Trials.* Chapman and Hall, London. 24

Kunert, J. and Sailer, O. (2005) On Nearly Balanced Designs for Sensory Trials. To appear in *Food Quality and Preference*. 24

Patterson, H.D. (1951) Change-over trials. *Journal of the Royal Statistical Society B* **13**, 256–271. 25

Patterson, H.D. (1952) The construction of balanced designs for experiments involving sequences of treatments. *Biometrika* **39**, 32–48. 25

Senn, S. (1993) *Cross-over Trials in Clinical Research.* Wiley, New York. 24

Street, A.P. and Street, D.J. (1987) *Combinatorics of experimental design.* Oxford University Press, Oxford. 25

Wakeling, I. N. and MacFie, H. J. H. (1995) Designing consumer trials for first and higher orders of carry-over effects when only a subset of k samples from p may be tested. *Food Quality and Preference* **6**, 299–308. 24

Wheeler, R.E. (2004). `optBlock`. **AlgDesign**. The R project for statistical computing http://www.r-project.org/ 24, 25

Williams, E. J. (1949) Experimental designs balanced for the estimation of residual effects of treatments. *Australian Journal of Scientific Research, Ser. A* **2**, 149–168.
24, 25

Oliver Sailer
Fachbereich Statistik, Universität Dortmund, Germany
sailer@statistik.uni-dortmund.de

R Help Desk

Make '`R CMD`' Work under Windows – an Example

Uwe Ligges and Duncan Murdoch

Introduction

During the last couple of months there have been a number of questions on both mailing lists 'R-help' and 'R-devel' on how to install R packages from source under Windows. Even though the procedure is documented in detail in the "R Installation and Administration" (R Development Core Team, 2005a) manual, people still seem to be uncertain about some details. This is perhaps partly caused by some confusing online material published by others who tried to help, but there is also the fact that requirements change over time, and the fact that the procedure is quite different from what most Windows users are used to doing.

To install packages from source, you will need to use the '`R CMD`' commands at a command shell (which Windows users sometimes call a "DOS prompt", even though DOS is long gone from modern versions of Windows). A well set up environment is required. For recent versions of R, all you need to do is described in the "R Installation and Administration" manual (R Development Core Team, 2005a), Appendix "The Windows toolset". It is immensely important to follow the instructions given therein exactly. While some variations will work, you will cause yourself a lot of grief if you happen on one that does not.

We will give *an example* and describe what we did on a Windows XP machine to get a working environment. We give the URLs of downloads in the footnotes. These URLs work at the time of this writing (November, 2005), but may change over time. Check the Rtools web page[1] for updates. All the software will be installed into the directory 'c:\devel', but the user can choose arbitrary directories. It is highly recommended to avoid special characters and blanks in the directory names, though.

First, we have installed a recent (binary) version of R from a local CRAN mirror[2] into 'c:\devel\R-2.2.0' containing the executable in its 'bin' directory. Of course, it is also possible to install/compile R from sources itself, but this is a bit more complicated than just installing packages. Now follow the sections of the mentioned Appendix:

1. "The command line tools" are downloaded[3] and unzipped so that the executables are located in 'c:\devel\tools\bin'.

2. "Perl" is downloaded[4] and installed so that the executables are located in 'c:\devel\perl\bin'.

3. The current release of the "The MinGW compilers" is downloaded[5] and installed so that the executables are located in 'c:\devel\MinGW\bin'.

   Alternatively, one may download the recent versions of the components separately from http://www.mingw.org: 'gcc-core-VERSION.tar.gz', 'gcc-g++-VERSION.tar.gz', 'gcc-g77-VERSION.tar.gz', 'binutils-VERSION.tar.gz', 'mingw-runtime-VERSION.tar.gz', and 'w32api-VERSION.tar.gz' (replace 'VERSION' with the current version numbers) and unpack them manually after the rest of the setup. More details below.

[1] http://www.murdoch-sutherland.com/Rtools
[2] A list of CRAN mirrors is given at http://CRAN.R-Project.org/mirrors.html.
[3] http://www.murdoch-sutherland.com/Rtools/tools.zip
[4] http://www.activestate.com/Products/ActivePerl/Download.html
[5] 'MinGW-5.0.0.exe' from http://sourceforge.net/projects/mingw/

4. "The Microsoft HTML Help Workshop" is downloaded[6] and installed to 'c:\devel\HtmlHelp\'.

5. "LaTeX": We choose the 'MiKTeX' distribution[7] and install it to 'c:\devel\texmf\' and 'c:\devel\localtexmf\', respectively. Therefore, the executables are in 'c:\devel\texmf\miktex\bin'.

The paths of 'Perl' and 'MiKTeX' are set automatically by their installers. However, you need to put the other tools on your path yourself. You can do this in the Windows Control Panel under "System | Advanced | Environment variables", but this will affect all programs on your system, and it may end up being modified when you install other software in the future. A better way is to create a small file called 'c:\devel\Rpaths.bat' containing the single line (without any blanks and only using ';' as the separator):

```
PATH=.;c:\devel\tools\bin;c:\devel\MinGW\bin;
c:\devel\R-2.2.0\bin;c:\devel\HtmlHelp;
c:\devel\Perl\bin;c:\devel\texmf\miktex\bin;
AllTheOtherStuff
```

where 'AllTheOtherStuff' stands for all the other stuff that was already set in your path. Most users can simply replace 'AllTheOtherStuff' by '%PATH%', but do not include any 'cygwin' path. If you have 'cygwin' in your path before, you can also replace 'AllTheOtherStuff' by a minimal configuration that is typically '%SystemRoot%\system32;%SystemRoot%'.

You also need to add an environment variable 'TMPDIR' pointing to an existing directory to be used for temporary files as described in Section "Installing packages" of R Development Core Team (2005a). This can also go into 'Rpath.bat'; a sample version of this file is available for download.[8].

To make sure the path is set each time you open a command shell, right click on the icon for your shell, choose "Properties | Shortcut" and change the "Target" to

```
C:\WINDOWS\system32\cmd.exe
/k c:\devel\Rpath.bat
```

(again, all on one line). Finally, edit the file 'c:\devel\R-2.2.0\src\gnuwin32\MkRules' and specify the directory of "The Microsoft HTML Help Workshop" at the appropriate place: `HHWDIR=c:/devel/HtmlHelp`. (Be careful here: you need to use an editor that leaves TAB characters unchanged. This file, like all of the R sources, uses Unix-style line terminators. This means that Notepad is *not* a good editor to use.)

After the steps above (which only need to be done once for each version of R), installation of a package is straightforward. For example, obtain the source (e.g. 'abind_1.1-0.tar.gz' [9]), and place it in 'c:\devel'. Open the command shell, change to this directory, and type `R CMD INSTALL abind_1.1-0.tar.gz`.

Some more information and examples on package management can be found in previous Newsletters (Ligges, 2003; Ripley, 2005).

To extract the source and look into the package's structure, type `tar -xfz abind_1.1-0.tar.gz`. This will create a subdirectory named 'abind' containing all of the source for the package. (You would use the same command to extract each of the MinGW components, if you had a problem above with 'MinGW-5.0.0.exe'. Do this in the 'c:\devel\MinGW' directory.) By extracting the sources we have prepared an example directory ('abind') that can be used to (re-)build the package's source tarball and install the package in the same way as you would do for a self written package as described in the following paragraph.

To develop your own package (e.g. 'mypackage'), create a subdirectory 'c:\devel\mypackage', and follow the instructions in the "Writing R Extensions" (R Development Core Team, 2005b) manual. The function `package.skeleton()` is especially helpful when getting started. Once you have all the files in place, go into 'c:\devel' in the command shell and type `R CMD build mypackage`. If that works, the next step is to try to install the package as described above: `R CMD INSTALL mypackage_version.tar.gz`. Finally, try `R CMD check mypackage_version.tar.gz` for extensive tests of your code and documentation. Installing or checking directly on the source tree of a package may pollute it (e.g. with object files, '.chm' files etc.), hence it is recommended to work on the already *built* package.

Bibliography

Ligges, U. (2003): R Help Desk: Package Management. *R News*, 3 (3), 37–39. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/. 28

R Development Core Team (2005a): R Installation and Administration. ISBN 3-900051-02-X. R Foundation for Statistical Computing, Vienna, Austria. 27, 28

R Development Core Team (2005b): Writing R Extensions. ISBN 3-900051-11-9. R Foundation for Statistical Computing, Vienna, Austria. 28

Ripley, B.D. (2005): Packages and their Management in R 2.1.0. *R News*, 5 (1), 8–11. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/. 28

[6] http://www.microsoft.com/office/ork/xp/appndx/appa06.htm
[7] http://www.miktex.org/setup.html
[8] http://www.murdoch-sutherland.com/Rtools/Rpath.bat
[9] http://cran.r-project.org/src/contrib/Descriptions/abind.html

Uwe Ligges
Fachbereich Statistik, Universität Dortmund, Germany
`ligges@statistik.uni-dortmund.de`
Duncan Murdoch
Statistical and Actuarial Sciences, University of Western Ontario, Canada
`murdoch@stats.uwo.ca`

Changes in R

by the R Core Team

User-visible changes

- `plot(<lm object>)` uses a new default 'which = 5' for the fourth panel when 'which' is not specified.

- The SVN revision number will appear after the date in the welcome message. The date shown is now the date of the last change to the sources rather than the date the sources were prepared.

- `is.null(expression())` now returns FALSE. Only NULL gives TRUE in `is.null()`.

- `graphics::xy.coords`, `xyz.coords` and `n2mfrow` have been moved to the grDevices name space (to be available for grid as well).

  `graphics::boxplot.stats`, `contourLines`, `nclass.*`, and `chull` have been moved to the grDevices name space. The C code underlying `chull()` has been moved to package grDevices.

- `split(x, f)`, `split<-()` and `unsplit()` now by default split by all levels of a factor f, even when some are empty. Use `split(x, f, drop = TRUE)` if you want the old behavior of dropping empty levels. `split()` and `split<-()` are S3 generic functions with new arguments 'drop' and '...' and all methods now should have 'drop' and '...' arguments as well.

- The default for 'allowEscapes' in both `read.table()` and `scan()` has been changed to FALSE.

- The default for 'gcFirst' in `system.time()` is now TRUE.

New features

- .Platform has a new component 'path.sep', the separator used between paths in environment variables such as PATH and TEXINPUTS.

- `anova.mlm()` now handles the single-model case.

- Hexadecimal values are now allowed for `as.numeric()` and `as.integer()` on all platforms, and as integer constants in R code.

- `attach()` now prints an information message when objects are masked on the search path by or from a newly attached database.

- `axis()` now returns 'at' positions.

- `axis()` has a new argument 'hadj' to control horizontal adjustment of labels.

- `axis.Date()` and `axis.POSIXct()` now accept a 'labels' argument (contributed by Gavin Simpson).

- `barplot()` now has arguments 'log = ""' and 'add = FALSE' (as in `barplot2()` from package 'gplots').

- `baseenv()` has been added, to return the base environment. This is currently still NULL, but will change in a future release.

- `boxplot()` now responds to supplying 'yaxs' (via `bxp()`). (Wish of PR#8072.)

- `capabilities()` has a new component 'NLS'.

- `cbind()` and `rbind()` now react to 'deparse.level' = 0,1,2 (as in another system not unlike R).

- Experimental versions of `cbind()` and `rbind()` in methods package, based on new generic function `cbind2(x,y)` and `rbind2()`. This will allow the equivalent of S4 methods for `cbind()` and `rbind()` — currently only after an explicit activation call, see ?cbind2.

- New functions `cdplot()` and `spineplot()` for conditional density plots and spine plots or spinograms. Spine plots are now used instead of bar plots for x-y scatterplots where y is a factor.

- `checkDocFiles()` in package 'tools' now checks for bad `\usage` lines (syntactically invalid R code).

- The nonparametric variants of `cor.test()` now behave better in the presence of ties. The "spearman" method uses the asymptotic approximation in that case, and the "kendall" method likewise, but adds a correction for ties (this is not necessary in the Spearman case).

- The X11 `dataentry()` now has support for X Input Methods (contributed by Ei-ji Nakama).
- `density()` is now an S3 generic where `density.default()` former `density()` has new argument 'weights' for specifying observation masses different than the default 1/N – based on a suggestion and code from Adrian Baddeley.
- `download.packages()` now carries on if it encounters a download error (e.g. a repository with a corrupt index).
- `dump()` now skips missing objects with a warning rather than throw an error.
- Added "POSIXlt" methods for `duplicated()` and `unique()`.
- Function `encoded_text_to_latex()` in package tools translates Latin 1,2,9 and UTF-8 encoded character vectors to LaTeX escape sequences where appropriate.
- `encodeString()` allows justify = "none" for consistency with `format.default()`. Some argument names have been lengthened for clarity.
- `file()`, `fifo()` and `pipe()` now (if possible) report a reason if they fail to open a connection.
- `format.default()` now has a 'width' argument, and 'justify' can now centre character strings.

  `format.default()` has new arguments 'na.encode' to control whether NA character strings are encoded (true by default), and 'scientific' to control the use of fixed/scientific notation for real/complex numbers.

  How `format()` works on a list is now documented, and uses arguments consistently with their usage on an atomic vector.
- `format.info()` now has a 'digits' argument, and is documented to work for all atomic vectors (it used to work for all but raw vectors.).
- New function `glob2rx()` for translating 'wildcard' aka 'globbing' to regular expressions.
- There is a new function `gregexpr()` which generalizes `regexpr()` to search for all matches in each of the input strings (not just the first match).
- `[g]sub()` now have a 'useBytes' argument like `grep()` and `regexpr()`.
- `[g]sub(perl = TRUE)` support `\L` and `\U` in the replacement.
- `iconv()` has been moved from 'utils' to 'base'.
- `identify()`'s default method has additional arguments 'atpen' and 'tolerance' (following S).
- `KalmanForecast()` and `KalmanLike()` now have an optional argument fast=FALSE to prevent their arguments being modified.
- Exact p-values are available in `ks.test()` for the one-sided and two-sided one-sample Kolmogorov-Smirnov tests.
- `labels()` now has a method for "dist" objects (replacing that for `names()` which was withdrawn in 2.1.0).
- `library()` now explicitly checks for the existence of directories in 'lib.loc': this avoids some warning messages.
- `loadNamespace(keep.source=)` now applies only to that namespace and not others it might load to satisfy imports: this is now consistent with `library()`.
- `match.arg()` has a new argument 'several.ok = FALSE'.
- `max.col()` has a new argument for non-random behavior in the case of ties.
- `memory.profile()` now uses the type names returned by `typeof()` and no longer has two unlabelled entries.
- `methods()` now warns if it appears to have been called on a non-generic function.
- The default `mosaicplot()` method by default draws grey boxes.
- `nlminb()`, similar to that in S-PLUS, added to package 'stats'.
- New algorithm "port" (the nl2sol algorithm available in the Port library on netlib) added to the `nls()` function in the 'stats' package.
- `object.size()` now supports more types, including external pointers and weak references.
- `options()` now returns its result in alphabetical order, and is documented more comprehensively and accurately. (Now all options used in base R are documented, including platform-specific ones.)

  Some options are now set in the package which makes use of them (grDevices, stats or utils) if not already set when the package is loaded.

- New option("OutDec") to set the decimal point for output conversions.

- New option("add.smooth") to add smoothers to a plot, currently only used by plot.lm().

- pie() has new optional arguments 'clockwise' and 'init.angle'.

- plot.lm() has two new plots (for 'which' = 5 or 6), plotting residuals or cook distances versus (transformed) leverages - unless these are constant. Further, the new argument 'add.smooth' adds a loess smoother to the point plots by default, and 'qqline = TRUE' adds a qqline() to the normal plot. The default for 'sub.caption' has been improved for long calls.

- R.home() has been expanded to return the paths to components (which can as from this version be installed elsewhere).

- readbin() and writeBin() now support raw vectors as well as filenames and connections.

- read.dcf() can now read gzipped files.

- read.table() now passes 'allowEscapes' to scan().

- sample(x, size, prob, replace = TRUE) now uses a faster algorithm if there are many reasonably probable values. (This does mean the results will be different from earlier versions of R.) The speedup is modest unless 'x' is very large _and_ 'prob' is very diffuse so that thousands of distinct values will be generated with an appreciable frequency.

- scatter.smooth() now works a bit more like other plotting functions (e.g., accepts a data frame for argument 'x'). Improvements suggested by Kevin Wright.

- signif() on complex numbers now rounds jointly to give the requested number of digits in the larger component, not independently for each component.

- New generic function simulate() in the 'stats' package with methods for some classes of fitted models.

- smooth.spline() has a new argument 'keep.data' which allows to provide residuals() and fitted() methods for smoothing splines.

- Attempting source(file, chdir=TRUE) with a URL or connection for 'file' now gives a warning and ignores 'chdir'.

- source() closes its input file after parsing it rather than after executing the commands, as used to happen prior to 2.1.0. (This is probably only significant on Windows where the file is locked for a much shorter time.)

- split(), split<-(), unsplit() now have a new argument 'drop = FALSE', by default not dropping empty levels; this is *not* back compatible.

- sprintf() now supports asterisk '*' width or precision specification (but not both) as well as '*1$' to '*99$'. Also the handling of '%' as conversion specification terminator is now left to the system and doesn't affect following specifications.

- The plot method for stl() now allows the colour of the range bars to be set (default unchanged at "light gray").

- Added tclServiceMode() function to the tcltk package to allow updating to be suspended.

- terms.formula() no longer allows '.' in a formula unless there is a (non-empty) 'data' argument or 'allowDotAsName = TRUE' is supplied. We have found several cases where 'data' had not been passed down to terms() and so '.' was interpreted as a single variable leading to incorrect results.

- New functions trans3d(), the 3D -> 2D utility from persp()'s example, and extendrange(), both in package 'grDevices'.

- TukeyHSD() now returns p-values adjusted for multiple comparisons (based on a patch contributed by Fernando Henrique Ferraz P. da Rosa).

- New functions URLencode() and URLdecode(), particularly for use with file:// URLs. These are used by e.g. browse.env(), download.file(), download.packages() and various help() print methods.

- Functions utf8ToInt() and intToUtf8() to work with UTF-8 encoded character strings (irrespective of locale or OS-level UTF-8 support).

- [dqp]wilcox and wilcox.test work better with one very large sample size and an extreme first argument.

- write() has a new argument 'sep'.

- write.csv[2] now also support 'row.names = FALSE'.

- The specification of the substitutions done when processing Renviron files is more liberal: see ?Startup. It now accepts forms like

R_LIBS=${HOME}/Rlibrary:${WORKGRP}/R/lib
.

- Added recommendation that packages have an overview man page <pkg>-package.Rd, and the `promptPackage()` function to create a skeleton version.
- Replacement indexing of a data frame by a logical matrix index containing NAs is allowed in a few more cases, in particular always when the replacement value has length one.
- Conversion of .Rd files to LaTeXnow handles encoding more comprehensively, including some support for UTF-8.
- The internal regex code has been upgraded to glibc-2.3.5. Apart from a number of bug fixes, this should be somewhat faster, especially in UTF-8 locales.
- PCRE has been updated to version 6.2.
- zlib has been updated to version 1.2.3.
- bzip2 has been updated to version 1.0.3.
- Complex arithmetic is now done by C99 complex types where supported. This is likely to boost performance, but is subject to the accuracy with which it has been implemented.
- The printing of complex numbers has changed, handling numbers as a whole rather than in two parts. So both real and imaginary parts are shown to the same accuracy, with the 'digits' parameter referring to the accuracy of the larger component, and both components are shown in fixed or scientific notation (unless one is entirely zero when it is always shown in fixed notation).
- Error messages from `.C()` and `.Fortran()`, and from parsing errors, are now more informative.
- The date and date-time functions work better with dates more than 5000 years away from 1970-01-01 (by making dubious assumptions about the calendar in use).
- There is now a traditional Chinese translation, and a much more extensive Russian translation.

Deprecated & defunct

- Capability "IEEE754" is defunct.
- `loadURL()` is defunct: use `load(url())`.
- `delay()` is defunct: use `delayedAssign()` instead.
- The 'CRAN' argument to `update.packages()`, `old.packages()`, `new.packages()`, `download.packages()` and `install.packages()` is defunct in favour of 'repos'.
- `write.table0()` is deprecated in favour of the much faster `write.table()`.
- `format.char()` is deprecated in favour of `format.default()`.
- R_HOME/etc/Rprofile is no longer looked for if R_HOME/etc/Rprofile.site does not exist. (This has been undocumented since R 1.4.0.)
- `CRAN.packages()` is deprecated in favour of `available.packages()`.
- Rd.sty no longer processes pre-2.0.0 conversions containing `\Link`.
- The stubs for the defunct device GNOME/gnome have been removed.
- `print.matrix()` (which has been identical to print.default since R 1.7.0) has been removed.

Installation

- LDFLAGS now defaults to -L/usr/local/lib64 on most Linux 64-bit OSes (but not ia64). The use of lib/lib64 can be overridden by the new variable LIBnn.
- The default installation directory is now *prefix*/LIBnn/R, /usr/local/lib64/R on most 64-bit Linux OSes and /usr/local/lib/R elsewhere.
- The places where the doc, include and share directory trees are installed can be specified independently: see the R-admin manual.
- We now test for wctrans_t, as apparently some broken OSes have wctrans but not wctrans_t (which is required by the relevant standards).
- Any external BLAS found is now tested to see if the complex routine zdotu works correctly: this provides a compatibility test of compiler return conventions.
- Installation without NLS is now cleaner, and does not install any message catalogues.
- src/modules/lapack/dlamc.f is now compiled with -ffloat-store if f2c/gcc are used, as well as if g77 is used.
- All the Fortran code has been checked to be fully F77 compliant so there are no longer any warnings from F95 compilers such as gfortran.

- The (not-recommended) options –with-system-zlib, –with-system-bzlib and -with-system-pcre now have 'system' in the name.
- If a Java runtime environment is detected at configure time its library path is appended to LD_LIBRARY_PATH or equivalent. New Java-related variables JAVA_HOME (path to JRE/JDK), JAVA_PROG (path to Java interpreter), JAVA_LD_PATH (Java library path) and JAVA_LIBS (flags to link against JNI) are made available in Makeconf.
- Ei-ji Nakama was contributed a patch for FPU control with the Intel compilers on ix86 Linux.

Mac OS X Installation

- –with-blas="-framework vecLib" –with-lapack and –with-aqua are now the default configure options.
- The default framework version name was changed to not contain the patch level (i.e. it is now 2.2 instead of 2.2.0). Also it can be overridden at configure time by setting FW_VERSION to the desired name.
- The Rmath stand-alone library is now correctly installed inside the R.framework if R was configured as a framework. In addition, make install-Rmath-framework will install a stand-alone Rmath framework in /Library/Frameworks (unless overridden by RMATH_FRAMEWORK_DIR specifying full framework path and name including the .framework extension).

Package Installation

- The encoding for a packages' 00Index.html is chosen from the Encoding: field (if any) of the DESCRIPTION file and from the `\encoding{}` fields of any Rd files with non-ASCII titles. If there are conflicts, first-found wins with a warning.
- R_HOME/doc/html/packages.html is now remade by R not Perl code. This may result in small changes in layout and a change in encoding (to UTF-8 where supported).
- The return value of `new.packages()` is now updated for any packages which may be installed.
- `available.packages()` will read a compressed PACKAGES.gz file in preference to PACKAGES if available on the repository: this will reduce considerably the download time on a dialup connection.

  The downloaded information about a repository is cached for the current R session.
- The information about library trees found by `installed.packages()` is cached for the current session, and updated only if the modification date of the top-level directory has been changed.
- A data index is now installed for a package with a 'data' dir but no 'man' dir (even though it will have undocumented data objects).
- contrib.url path for type="mac.binary" has changed from bin/macosx/<version> to bin/macosx/<arch>/contrib/<version> where <arch> corresponds to R.version$arch

Utilities

- `checkFF()` used by R CMD check has since R 2.0.0 not reported missing PACKAGE arguments when testing installed packages with namespaces. It now
  - treats installed and source packages in the same way.
  - reports missing arguments unless they are in a function in the namespace with a useDynLib declaration (as the appropriate DLL for such calls can be searched for).
- Rd2dvi sets the `encoding(s)` used appropriately. If UTF-8 encoding is used, latex >= 2003/12/01 is required.
- `codoc()` allows help files named pkg_name-defunct.Rd to have undocumented arguments (and not just base-defunct.Rd).

C-level facilities

- C function `massdist()` called from `density()` has new argument 'xmass' (= weights).
- Raw vectors passed to `.C()` are now passed as unsigned char * rather than as SEXPs. (Wish of Keith Frost, PR#7853)
- The search for symbols in a .C/.Call/... call without a package argument now searches for an enclosing namespace and so finds functions defined within functions in a namespace.
- `R_max_col()` has new (5th) argument '*ties_meth' allowing non-random behavior in the case of ties.

- The header files have been rationalized: the BLAS routine LSAME is now declared in BLAS.h not Linpack.h, Applic.h no longer duplicates routines from Linpack.h, and Applic.h is divided into API and non-API sections.
- memory.c has been instrumented so that Valgrind can track R's internal memory management. To use this, configure using –with-valgrind-instrumentation=level where level is 1 or 2. Both levels will find more bugs with `gctorture(TRUE)`. Level 2 makes Valgrind run extremely slowly.
- Some support for raw vectors has been added to Rdefines.h.
- R_BaseEnv has been added, to refer to the base environment. This is currently equal to R_NilValue, but it will change in a future release.

Bug fixes

- %/% has been adjusted to make `x == (x %% y) + y * ( x %/% y )` more likely in cases when extended-precision registers were interfering.
- Operations on POSIXct objects (such as `seq()`, `max()` and subsetting) try harder to preserve time zones and warn if inconsistent time zones are used.
- `as.function.default()` no longer asks for a bug report when given an invalid body. (PR#1880, PR#7535, PR#7702)
- Hershey fonts and grid output (and therefore lattice output) now rescale correctly in fit-to-window resizing on a Windows graphics device. Line widths also scale now.
- Plotmath has more support for multibyte characters (contributed by Ei-ji Nakama).
- The `X11()` device now hints the window manager so that decorations appear reliably under e.g. the GNOME WM (contributed by Ei-ji Nakama).
- Subsetting a matrix or an array as a vector used to attempt to use the row names to name the result, even though the array might be longer than the row names. Now this is only done for 1D arrays when it is done in all cases, even matrix indexing. (Tidies up after the fix to PR#937.)
- Constants in mathlib are declared 'const static double' to avoid performance issues with the Intel Itanium compiler.
- The parser checks the format of numeric constants more thoroughly so for example '123E-' is no longer valid.
- `contourLines()` no longer requires an open device (used to start a device unnecessarily). Fix suggested by Barry Rowlingson.
- `capabilities()` used partial matching but was not documented to: it no longer does so.
- `kernel(1,0)` printed wrongly; `kernel(<name-string>, *)` now returns a named kernel in all cases; `plot(kernel(.),..)` is more flexible.
- `qgamma(1,s)` didn't give +Inf for some s.
- `installed.packages()` and `download.packages()` now always return a matrix as documented, possibly with 0 rows (rather than a 0-length character vector or NULL).
- Arithmetic operations on data frames no longer coerce the names to syntatically valid names.
- Units are now properly recycled in grid layouts when 'widths' or 'heights' are shorter than the number of columns or rows (PR#8014).
- DF <- `data.frame(A=1:2, B=3:4)`; DF[1, 1:3] <- NULL gave a wrong error message.
- `spline()/spinefun()`'s C code had a memory access buglet which never lead to incorrect results. (PR#8030)
- `sum()` was promoting logical arguments to double not integer (as `min()` and other members of its group do).
- `loess()` had a bug causing it to occasionally miscalculate standard errors (PR#7956). Reported by Benjamin Tyner, fixed by Berwin Turlach.
- `library(keep.source=)` was ignored if the package had a namespace (the setting of `options("keep.source.pkgs")` was always used).
- `hist.POSIXct()` and `hist.Date()` now respect `par("xaxt")`.
- The 'vfont' argument was not supported correctly in `title()`, `mtext()`, and `axis()`. The 'vfont' argument is superseded by the `par(family=)` approach introduced in 2.0.0. This bug-fix just updates the warning messages and documentation to properly reflect the new order of things.

- The C-level function PrintGenericVector could overflow if asked to print a length-1 character vector of several thousand characters. This could happen when printing a list matrix, and was fatal up to 2.1.1 and silently truncated in 2.1.1 patched.
- What happened for `proc.time()` and `system.time()` on (Unix-alike) systems which do not support timing was incorrectly documented. (They both exist but throw an error.) Further, `systen.time()` would give an error in its on.exit expression.
- `weighted.residuals()` now does sensible things for `glm()` fits: in particular it now agrees with an `lm()` fit for a Gaussian `glm()` fit. (PR#7961).
- The 'lm' and 'glm' methods for `add1()` took the weights and offset from the original fit, and so gave errors in the (dubious) usage where the upper scope resulted in a smaller number of cases to fit (e.g. by omitting missing values in new variables). (PR#8049)
- `demo()` had a 'device' argument that did nothing (although it was documented to): it has been removed.
- Setting new levels on a factor dropped all existing attributes, including class "ordered".
- `format.default(justify="none")` now by default converts NA character strings, as the other values always did.
- `format.info()` often gave a different field width from `format()` for character vectors (e.g. including missing values or non-printable characters).
- `axis()` now ensures that if 'labels' are supplied as character strings or expressions then 'at' is also supplied (since the calculated value for 'at' can change under resizing).
- Defining S4 methods for "[" had resulted in changed behavior of S3 dispatch in a very rare case which no longer happens.
- Fixed segfault when PostScript font loading fails, e.g., when R is unable to find afm files (reported by Ivo Welch).
- R CMD BATCH <file> now also works when <file> does not end in a newline on Unix-alike platforms.
- `terms.formula()` got confused if the 'data' argument was a list with non-syntactic names.
- `prompt()` and hence `package.skeleton()` now produce *.Rd files that give no errors (but warnings) when not edited, much more often.
- `promptClass()` and `promptMethods()` now also escape "%" e.g. in '%*%' and the latter gives a message about the file written.
- `wilcox.test()` now warns when conf.level is set higher than achievable, preventing errors (PR#3666) and incorrect answers with extremely small sample sizes.
- The default (protection pointer) stack size (the default for '–max-ppsize') has been increased from 10000 to 50000 in order to match the increased default `options("expressions")` (in R 2.1.0).
- The R front-end was expecting –gui=tk not Tk as documented, and rejecting –gui=X11.
- Rdconv -t latex protected only the first « and » in a chunk against conversion to guillemets.
- `callNextMethod()` and `callGeneric()` have fixes related to handling arguments.
- `ls.diag()` now works for fits with missing data. (PR#8139)
- `window.default()` had an incorrect tolerance and so sometimes created too short a series if 'start' or 'end' were zero.
- Some (fairly pointless) cases of reshape left a temporary id variable in the result (PR#8152)
- R CMD build used 'tar xhf' which is invalid on FreeBSD systems (and followed tar chf, so there could be no symbolic links in the tarball).
- Subassignment of length zero vectors to NULL gave garbage answers. (PR#8157)
- Automatic coercion of raw vectors to lists was missing, so for a list (or data frame) z, z[["a"]] <- raw_vector did not work and now does. This also affected DF$a <- raw_vector for a data frame DF.
- The internal code for `commandArgs()` was missing PROTECTs.
- The width for `strwrap()` was used as one less than specified.
- R CMD INSTALL was not cleaning up after an unsuccessful install of a non-bundle which was not already installed.

Changes on CRAN

by Kurt Hornik

New contributed packages

ArDec Implements the autoregressive decomposition of a time series based on the constructive approach in West (1997). Particular cases include the extraction of trend and seasonal components from a monthly time series. Uncertainty on the resulting components can be derived from sampling of the autoregressive model which is written as a linear regression model and handled on a Bayesian framework. By S. M. Barbosa.

BACCO Bayesian analysis of computer code software. The bundle contains routines that evaluate the formulae of Kennedy, O'Hagan, and Oakley. By Robin K. S. Hankin.

BMA Bayesian model averaging for linear models, generalizable linear models and survival models (Cox regression). By Adrian Raftery, Jennifer Hoeting, Chris Volinsky, and Ian Painter.

BRugs An R package containing OpenBUGS and its R interface BRugs. The Chief Software Bug is Andrew Thomas, with web assistance from Real Bug Bob O'Hara. Other members of the BUGS team are statisticians David Spiegelhalter, Nicky Best, Dave Lunn, and Ken Rice. Dave Lunn has also made major contributions to the software development. R Code modified, extended and packaged for R by Uwe Ligges and Sibylle Sturtz. Some ideas taken from the R2WinBUGS package based on code by Andrew Gelman.

BSDA Data sets for book "Basic Statistics and Data Analysis" by Larry J. Kitchens. By Alan T. Arnholt.

Biodem Provides a number of functions for Biodemographycal analysis. By Alessio Boattini and Federico C. F. Calboli; Vincente Canto Cassola together with Martin Maechler authored the function `mtx.exp()`.

DDHFm Contains the normalizing and variance stabilizing Data-Driven Haar-Fisz algorithm. Also contains related algorithms for simulating from certain microarray gene intensity models and evaluation of certain transformations. By Efthimios S. Motakis, Guy P. Nason, and Piotr Fryzlewicz.

DEoptim Provides the DEoptim function which performs Differential Evolution Optimization (evolutionary algorithm). By David Ardia.

DICOM Provides functions to import and manipulate medical imaging data via the Digital Imaging and Communications in Medicine (DICOM) Standard. By Brandon Whitcher.

ElemStatLearn Data sets, functions and examples from the book "The Elements of Statistical Learning, Data Mining, Inference, and Prediction" by Trevor Hastie, Robert Tibshirani and Jerome Friedman.

Epi Useful functions for demographic and epidemiological analysis in the Lexis diagram, i.e., register and cohort follow-up data. Also some useful functions for tabulation and plotting. Contains some epidemiological datasets. By David Clayton, Martyn Plummer, Bendix Carstensen, Mark Myatt et. al.

Fahrmeir Data and functions for the book "Multivariate Statistical Modelling Based on Generalized Linear Models" by Ludwig Fahrmeir and Gerhard Tutz. Compiled by Kjetil Halvorsen.

ICE Kernel Estimators for Interval-Censored Data. By W. John Braun.

JLLprod Implements the Lewbel and Linton (2003), Lewbel and Linton (2005) and Jacho-Chávez, Lewbel and Linton (2005) nonparametric estimators of Homothetic and Generalized Homothetic production functions. By David Tomás Jacho-Chávez.

Kendall Computes the Kendall rank correlation. By A.I. McLeod.

LMGene Analysis of microarray data using a linear model and glog data transformation. By David Rocke and Geun Cheol Lee.

Lmoments Contains functions to estimate L-moments and trimmed L-moments from the data. Also contains functions to estimate the parameters of the normal polynomial quantile mixture and the Cauchy polynomial quantile mixture from L-moments and trimmed L-moments. By Juha Karvanen.

MSBVAR Provides methods for estimating frequentist and Bayesian Vector Autoregression (VAR) models. Also includes methods for the generating posterior inferences for VAR forecasts, impulse responses (using likelihood-based error bands), and forecast error decompositions. Also includes utility functions for plotting forecasts and impulse responses, and generating draws from Wishart and singular multivariate normal densities. Future versions will include Bayesian Structural VAR (B-SVAR) models and possibly some models with Markov switching. By Patrick T. Brandt.

MarkedPointProcess Non-parametric Analysis of the Marks of Marked Point Processes. By Martin Schlather.

POT Some functions useful to perform a Peak Over Threshold analysis. Some piece of code come from the **evd** Package of Alec Stephenson. By Mathieu Ribatet.

R.utils This package provides utility classes and methods useful when programming in R and developing R packages. By Henrik Bengtsson.

RFA Several function to perform a Regional Frequency Analysis. By Mathieu Ribatet.

RGraphics Data and functions from the book "R Graphics". There is a function to produce each figure in the book, plus several functions, classes, and methods defined in Chapter 7. By Paul Murrell.

RLMM A classification algorithm, based on a multi-chip, multi-SNP approach for Affymetrix SNP arrays. Using a large training sample where the genotype labels are known, this algorithm will obtain more accurate classification results on new data. **RLMM** is based on a robust, linear model and uses the Mahalanobis distance for classification. The chip-to-chip non-biological variation is removed through normalization. This model-based algorithm captures the similarities across genotype groups and probes, as well as thousands other SNPs for accurate classification. NOTE: 100K-Xba only at for now. By Nusrat Rabbee and Gary Wong.

RWinEdt A plug in for using WinEdt as an editor for R. By Uwe Ligges.

RXshrink Identify and display TRACEs for a specified shrinkage path and determine the extent of shrinkage most likely, under normal distribution theory, to produce an optimal reduction in MSE Risk in estimates of regression (beta) coefficients. By Bob Obenchain.

RandVar Implementation of random variables by means of S4 classes and methods. By Matthias Kohl.

Rfwdmv Explore Multivariate Data with the Forward Search. By Anthony Atkinson, Andrea Cerioli, and Marco Riani.

Rlsf Functions for using R with the LSF cluster/grid queuing system. By Chris Smith and Gregory Warnes.

Rpad A workbook-style user interface to R through a web browser. Provides convenient plotting, HTML GUI generation, and HTML output routines. Can be used with R in standalone mode or with a web server to serve Rpad pages to other users. By Tom Short and Philippe Grosjean.

SAGx Retrieval, preparation and analysis of data from the Affymetrix GeneChip. In particular the issue of identifying differentially expressed genes is addressed. By Per Broberg.

SensoMineR For analysing sensory data. By François Husson and Sébastien Lê.

SharedHT2 Derives per gene means and fits the Wishart/inverse Wishart conjugate family to the per gene empirical covariance matrices. Derives a Hotelling T^2 statistic having an F-distribution using an empirical Bayes variance. By Grant Izmirlian.

SwissAir Ozone, NO_x (= Sum of Nitrogenmonoxide and Nitrogendioxide), Nitrogenmonoxide, ambient temperature, dew point, wind speed and wind direction at 3 sites around lake of Lucerne in Central Switzerland in 30 min time resolution for year 2004. By Rene Locher.

TeachingDemos A set of demonstration functions that can be used in a classroom to demonstrate statistical concepts, or on your own to better understand the concepts or the programming. By Greg Snow.

USPS Identify and display the distribution of Local Treatment Differences (LTDs) and Local Average Treatment Effects (LATEs) in Outcome within Clusters of patients chosen to be relatively well matched on baseline X-covariates. By Bob Obenchain.

WhatIf Inferences about counterfactuals are essential for prediction, answering what if questions, and estimating causal effects. However, when the counterfactuals posed are too far from the data at hand, conclusions drawn from well-specified statistical analyses become based largely on speculation hidden in convenient modeling assumptions that few would be willing to defend. Unfortunately, standard statistical approaches assume the veracity of the model rather than revealing the degree of model-dependence, which makes this problem hard to detect. **WhatIf** offers easy-to-apply methods to evaluate counterfactuals that do not require sensitivity testing over specified classes of models. If an analysis fails the tests offered here, then we know that substantive inferences will be sensitive to at least some modeling choices that are not based on empirical evidence, no matter what method of inference one chooses to use. **WhatIf** implements the methods for evaluating counterfactuals discussed in Gary King and Langche Zeng (2006), The Dangers of Extreme Counterfactuals, *Political Analysis* **14**(2), and Gary King and Langche Zeng (2006), When Can History Be Our Guide? The Pitfalls of Counterfactual Inference, *International Studies Quarterly*. By Heather Stoll, Gary King, and Langche Zeng.

aaMI Contains five functions. `read.FASTA()` reads in a FASTA format alignment file and parses it into a data frame. `read.CX()` reads in a ClustalX '`.aln`' format file and parses it into a data frame. `read.Gdoc()` reads in a GeneDoc '`.msf`' format file and parses it into a data frame. The alignment data frame returned by each of these functions has the sequence IDs as the row names and each site in the alignment is a column in the data frame. `aaMI()` calculates the mutual information between each pair of sites (columns) in the protein sequence alignment data frame. `aaMIn()` calculates the normalized mutual information between pairs of sites in the protein sequence alignment data frame. The normalized mutual information of sites i and j is the mutual information of these sites divided by their joint entropy. By Kurt Wollenberg.

aod Provides a set of functions to analyze overdispersed counts or proportions. Most of the methods are already available elsewhere but are scattered in different packages. The proposed functions should be considered as complements to more sophisticated methods such as generalized estimating equations (GEE) or generalized linear mixed effect models (GLMM). By Matthieu Lesnoff and Renaud Lancelot.

apTreeshape Mainly dedicated to simulation and analysis of phylogenetic tree topologies using statistical indices. It is a companion library of the **ape** package. It provides additional functions for reading, plotting, manipulating phylogenetic trees. It also offers convenient web-access to public databases, and enables testing null models of macroevolution using corrected test statistics. Trees of class `"phylo"` (from package **ape**) can be converted easily. By Nicolas Bortolussi, Eric Durand, Michael Blum, and Olivier François.

aster Functions and datasets for Aster modeling (forest graph exponential family conditional or unconditional canonical statistic models for life history analysis). By Charles J. Geyer.

baymvb Fits multivariate binary data using Bayesian approach. By S. M. Mwalili.

bicreduc Reduction algorithm for the NPMLE for the distribution function of bivariate interval-censored data. The main function is `HMA()`, the HeightMapAlgorithm. This algorithm is based on the idea of an "height map", and is described in the following paper: "Reduction algorithm for the NPMLE for the distribution function of bivariate interval-censored data", by Marloes Maathuis, *Journal of Computational and Graphical Statistics*, **14** (2) (to appear). By Marloes Maathuis.

biopara A parallel system designed to be used with R. By Peter Lazar and David Schoenfeld.

bivpois Fitting Bivariate Poisson Models using the EM algorithm. Details can be found in Karlis and Ntzoufras (2003, JRSS D; 2004, AUEB Technical Report). By Dimitris Karlis and Ioannis Ntzoufras.

butler Testing, profiling and benchmarking of code. By Hadley Wickham.

caMassClass Processing and classification of protein mass spectra (SELDI) data. Also includes reading and writing of mzXML files. By Jarek Tuszynski.

caTools Several basic utility functions including: moving (rolling, running) window statistic functions, read/write for GIF and ENVI binary files, fast calculation of AUC, LogitBoost classifier, base64 encoder/decoder, round-off error free sum and cumsum, etc. By Jarek Tuszynski.

cba Clustering techniques such as Proximus and Rock, utility functions for efficient computation of cross distances and data manipulation. By Christian Buchta.

compositions Provides functions for the consistent analysis of compositional data (e.g., portions of substances) and positive numbers (e.g., concentrations) in the way proposed by Aitchison. By K. Gerald van den Boogaart and Raimon Tolosana, with contributions of Matevz Bren.

copula Classes (S4) of commonly used copulas including elliptical and Archimedean copulas. Implemented copulas include normal, t, Clayton, Frank, and Gumbel. Methods for `density()`, distribution, random number generators, `persp()`, and `contour()`. By Jun Yan.

corpcor Implements a shrinkage estimator to allow the efficient inference of large-scale covariance matrices from small sample data. The resulting estimates are always positive definite, more accurate than the empirical estimate, well conditioned, computationally inexpensive, and require only little a priori modeling. The package also contains similar functions for inferring correlations and partial correlations. In addition, it provides functions for fast SVD computation, for computing the pseudoinverse, and for checking the rank and positive definiteness of a matrix. By Juliane Schaefer and Korbinian Strimmer.

corpora Utility functions for the statistical analysis of corpus frequency data. By Stefan Evert.

cslogistic Functions for likelihood and posterior analysis of conditionally specified logistic regression models. All calculus and simulation is done in compiled FORTRAN. By Alejandro Jara Vallejos and Maria Jose Garcia-Zattera.

depmix Fit (multigroup) mixtures of latent Markov models on mixed categorical and continuous (time series) data. By Ingmar Visser.

dglm Fitting double generalized linear models. By Peter K Dunn and Gordon K Smyth.

distrEx Extensions of package **distr** and some additional functionality. By Matthias Kohl.

distrSim Simulation (S4-)classes based on package **distr**. By Florian Camphausen, Matthias Kohl, Peter Ruckdeschel, and Thomas Stabla.

distrTEst Evaluation (S4-)classes based on package **distr** for evaluating procedures (estimators/tests) at data/simulation in a unified way. By Florian Camphausen, Matthias Kohl, Peter Ruckdeschel, and Thomas Stabla.

dprep Functions for normalization, treatment of missing values, discretization, outlier detection, feature selection, and visualization. By Edgar Acuna and Caroline Rodriguez.

dyn Time series regression. The dyn class interfaces `ts`, `irts`, `its`, `zoo` and `zooreg` time series classes to `lm()`, `glm()`, `loess()`, `quantreg::rq()`, `MASS::rlm()`, `randomForest::randomForest()` and other regression functions allowing those functions to be used with time series including specifications that may contain lags, diffs and missing values. By Gabor Grothendieck.

dynlm Dynamic linear models and time series regression. By Achim Zeileis.

elliptic A suite of elliptic and related functions including Weierstrass and Jacobi forms. Also includes various tools for manipulating and visualizing complex functions. By Robin K. S. Hankin.

equivalence Provides some statistical tests and graphics for assessing tests of equivalence. Such tests have similarity as the alternative hypothesis instead of the null. Sample datasets are included. By Andrew Robinson.

fCalendar The Rmetrics module for "Date, Time and Calendars".

fMultivar The Rmetrics module for "Multivariate Data Analysis".

fPortfolio The Rmetrics module for "Pricing and Hedging of Options".

fgac Generalized Archimedean Copula. Bi-variate data fitting is done by two ingredients: the margins and the dependency structure. The dependency structure is modeled through a copula. An algorithm was implemented considering seven families of copulas (Generalized Archimedean Copulas), the best fitting can be obtained looking all copula's options. By Veronica Andrea Gonzalez-Lopez.

filehash Simple file-based hash table. By Roger D. Peng.

gamlss The GAMLSS library and datasets. By Mikis Stasinopoulos and Bob Rigby, with contributions from Calliope Akantziliotou.

geometry Makes the qhull library (`www.qhull.org`) available in R, in a similar manner as in Octave and MATLAB. Qhull computes convex hulls, Delaunay triangulations, halfspace intersections about a point, Voronoi diagrams, furthest-site Delaunay triangulations, and furthest-site Voronoi diagrams. It runs in 2-d, 3-d, 4-d, and higher dimensions. It implements the Quickhull algorithm for computing the convex hull. Qhull does not support constrained Delaunay triangulations, or mesh generation of non-convex objects, but the package does include some R functions that allow for this. Currently the package only gives access to Delaunay triangulation and convex hull computation. By Raoul Grasman.

giRaph Supply data structures and algorithms for computations on graphs. By Jens Henrik Badsberg, Claus Dethlefsen, and Luca La Rocca.

glpk The GNU Linear Programming Kit (GLPK) version 4.8. This interface mirrors the GLPK C API. Almost all GLPK lpx routines are supported. This release is beta status due to the fact that not all routines have been tested. Suggestions and improvements are solicited. By Lopaka Lee.

gmt Interface between GMT 4.0 map-making software and R, enabling the user to manipulate geographic data within R and call GMT programs to draw and annotate maps in postscript format. By Arni Magnusson.

gnm Functions to specify and fit generalized nonlinear models, including models with multiplicative interaction terms such as the UNIDIFF model from sociology and the AMMI model from crop science. This is a major re-working of an earlier Xlisp-Stat package, "Llama". Over-parameterized representations of models are used throughout; functions are provided for inference on estimable parameter combinations, as well as standard methods for diagnostics etc. By Heather Turner and David Firth.

grouped Regression models for grouped and coarse data, under the Coarsened At Random assumption. By Dimitris Rizopoulos and Spyridoula Tsonaka.

hapsim Package for haplotype data simulation. Haplotypes are generated such that their allele frequencies and linkage disequilibrium coefficients match those estimated from an input data set. By Giovanni Montana.

hoa A bundle with various functions for higher order likelihood-based inference. Contains **cond** for approximate conditional inference for logistic and log-linear models, **csampling** for conditional simulation in regression-scale models, **marg** for approximate marginal inference for regression-scale models, and **nlreg** for higher order inference for nonlinear heteroscedastic models. By Alessandra R. Brazzale.

howmany When testing multiple hypotheses simultaneously, this package provides functionality to calculate a lower bound for the number of correct rejections (as a function of the number of rejected hypotheses), which holds simultaneously—with high probability—for all possible number of rejections. As a special case, a lower bound for the total number of false null hypotheses can be inferred. Dependent test statistics can be handled for multiple tests of associations. For independent test statistics, it is sufficient to provide a list of *p*-values. By Nicolai Meinshausen.

iid.test Testing whether data is independent and identically distributed. By Rasmus E. Benestad.

kappalab Kappalab, which stands for "laboratory for capacities", is an S4 tool box for capacity (or non-additive measure, fuzzy measure) and integral manipulation on a finite setting. It contains routines for handling various types of set functions such as games or capacities. It can be used to compute several non-additive integrals: the Choquet integral, the Sugeno integral, and the symmetric and asymmetric Choquet integrals. An analysis of capacities in terms of decision behavior can be performed through the computation of various indices such as the Shapley value, the interaction index, the orness degree, etc. The well-known Möbius transform, as well as other equivalent representations of set functions can also be computed. **Kappalab** further contains four capacity identification routines: two least squares based approaches, a maximum entropy like method based on variance minimization and an unsupervised approach grounded on parametric entropies. The functions contained in **Kappalab** can for instance be used for multicriteria decision making or in the framework of cooperative game theory. By Michel Grabisch, Ivan Kojadinovic, and Patrick Meyer.

latentnet Latent position and cluster models for statistical networks. These models are fit within the "ergm" framework of the **statnet** package. By Mark S. Handcock, based on original code from Jeremy Tantrum, Susan Shortreed, and Peter Hoff.

lodplot Assorted plots of location score versus genetic map position. By David L Duffy.

longmemo Data sets and functionality from the textbook "Statistics for Long-Memory Processes" by Jan Beran. Original S code by Jan Beran, data sets via Brandon Whitcher, top-level R functions and much more by Martin Maechler.

mblm Linear models based on Theil-Sen single median and Siegel repeated medians. They are very robust (29 or 50 percent breakdown point, respectively), and if no outliers are present, the estimators are very similar to OLS. By Lukasz Komsta.

micEcdat Individual data sets for microeconometrics, which come mainly from the Journal of Applied Econometrics' and the Journal of Business and Economic Statistics' data archives. By Yves Croissant.

mice Multivariate Imputation by Chained Equations. By S. Van Buuren and C.G.M. Oudshoorn.

misc3d A collection of miscellaneous 3d plots, including rgl-based isosurfaces. By Dai Feng and Luke Tierney.

modeltools A collection of tools to deal with statistical models. The functionality is experimental and the user interface is likely to change in the future. The documentation is rather terse, but packages **coin** and **party** have some working examples. By Torsten Hothorn and Friedrich Leisch.

moments Functions to calculate: moments, Pearson's kurtosis, Geary's kurtosis and skewness; tests related to them (Anscombe-Glynn, D'Agostino, Bonett-Seier). By Lukasz Komsta.

monoProc Monotonizes a given fit in one or two variables. By Regine Scheder.

multtest Non-parametric bootstrap and permutation resampling-based multiple testing procedures for controlling the family-wise error rate (FWER), generalized family-wise error rate (gFWER), tail probability of the proportion of false positives (TPPFP), and false discovery rate (FDR). Single-step and step-wise methods are implemented. Tests based on a variety of t- and F-statistics (including t-statistics based on regression parameters from linear and survival models) are included. Results are reported in terms of adjusted p-values, confidence regions and test statistic cutoffs. By Katherine S. Pollard, Yongchao Ge, and Sandrine Dudoit.

nFDR Implements the non-parametric estimate of FDR based on Bernstein polynomials. It calculates the proportion of true null hypotheses, FDR, FNR, and q-values. By Zhong Guan, Baolin Wu and Hongyu Zhao.

network Tools to create and modify network objects. The network class can represent a range of relational data types, and supports arbitrary vertex/edge/graph attributes. By Carter T. Butts, with help from David Hunter and Mark S. Handcock.

neural RBF and MLP neural networks with graphical user interface. By Ádám Nagy.

onion A collection of routines to manipulate and visualize quaternions and octonions. By Robin K. S. Hankin.

optmatch Functions to perform optimal matching, particularly full matching. By Ben B. Hansen, with embedded Fortran code due to Dimitri P. Bertsekas and Paul Tseng.

papply Similar to `apply()` and `lapply()`, applies a function to all items of a list, and returns a list with the results. Uses **Rmpi** to distribute the processing evenly across a cluster. If **Rmpi** is not available, implements this as a non-parallel algorithm. Includes some debugging support. By Duane Currie.

partsm Basic functions to fit and predict periodic autoregressive time series models. These models are discussed in the book "Periodicity and Stochastic Trends in Economic Time Series" by P.H. Franses. Data sets analyzed in that book are also provided. By Javier López-de-Lacalle.

party Unbiased recursive partitioning in a conditional inference framework. This package implements a unified framework for recursive partitioning which embeds tree-structured regression models into a well defined theory of conditional inference procedures. Stopping criteria based on multiple test procedures are implemented. The methodology is applicable to all kinds of regression problems, including nominal, ordinal, numeric, censored as well as multivariate response variables and arbitrary measurement scales of the covariates. Extensible functionality for visualizing tree-structured regression models is available. By Torsten Hothorn, Kurt Hornik, and Achim Zeileis.

permtest Uses permutations in order to compare the variability within and distance between two groups within a set of microarray data. By Douglas Hayden.

pgirmess Miscellaneous functions for analysis and display of ecological and spatial data. By Patrick Giraudoux.

popgen Implements a variety of statistical and population genetic methodology. By J L Marchini.

powerpkg Power analyses for the affected sib pair and the TDT design. By Daniel E. Weeks.

pscl Classes and/or methods for discrete data models; e.g., models for counts, ordinal data, item-response theory (IRT) models for binary data, computing highest density regions. By Simon Jackman.

quantchem Statistical evaluation of calibration curves by different regression techniques: ordinary, weighted, robust (up to 4th order polynomial). Log-log and Box-Cox transform, estimation of optimal power and weighting scheme. Tests for heteroscedascity and normality of residuals. Different kinds of plots commonly used in illustrating calibrations. Easy "inverse prediction" of concentration by given responses and statistical evaluation of results (comparison of precision and accuracy by common tests). By Lukasz Komsta.

randaes The deterministic part of the Fortuna cryptographic pseudorandom number generator, described in "Practical Cryptography" by Scheier and Ferguson. By Thomas Lumley.

rcdd R interface to (some of) cddlib. By Charles J. Geyer and Glen D. Meeden.

relax Functions for report writing, presentation, and programming. By Hans Peter Wolf.

reshape Flexibly reshape data. By Hadley Wickham.

sac Semi-parametric empirical likelihood ratio based test of change-point with one-change or epidemic alternatives with data-based model diagnostic. By Zhong Guan.

sampling A set of tools to select and to calibrate samples. By Yves Tillé and Alina Matei.

samr Significance Analysis of Microarrays. By R. Tibshirani, G. Chu, T. Hastie, and Balasubramanian Narasimhan.

scape Import and plot results from statistical catch-at-age models, used in fisheries stock assessments. By Arni Magnusson.

scapeMCMC Markov-chain Monte Carlo diagnostic plots, accompanying the **scape** package. The purpose of the package is to combine existing tools from the **coda** and **lattice** packages, and make it easy to adjust graphical details. It can be useful for anyone using MCMC analysis, regardless of the application. By Arni Magnusson and Ian Stewart.

simex Implementation of the SIMEX-Algorithm by Cook & Stefanski and MCSIMEX by Küchenhoff, Mwalili & Lesaffre. By Wolfgang Lederer.

sp Classes and methods for spatial data. The classes document where the spatial location information resides, for 2D or 3D data. Utility functions are provided, e.g. for plotting data as maps, spatial selection, as well as methods for retrieving coordinates, for subsetting, print, summary, etc. By Edzer J. Pebesma, Roger Bivand and others.

spectralGP Routines for creating, manipulating, and performing Bayesian inference about Gaussian processes in one and two dimensions using the Fourier basis approximation: simulation and plotting of processes, calculation of coefficient variances, calculation of process density, coefficient proposals (for use in MCMC). It uses R environments to store GP objects as references/pointers. By Chris Paciorek.

sspir A glm-like formula language to define dynamic generalized linear models (state space models). Includes functions for Kalman filtering and smoothing. By Claus Dethlefsen and Søren Lundbye-Christensen.

tdthap TDT tests for extended haplotypes. By David Clayton.

time A function for drawing a progress bar using standard text output, and additional time-tracking routines that allow developers to track how long their processor-intensive calculations take. By Toby Dylan Hocking.

tlnise Functions for two level normal models as described in Everson and Morris (2000), JRSS B. S-PLUS original by Phil Everson; R port by Roger D. Peng.

trust Local optimization using two derivatives and trust regions. By Charles J. Geyer.

tseriesChaos Routines for the analysis of nonlinear time series. This work is largely inspired by the TISEAN project, by Rainer Hegger, Holger Kantz and Thomas Schreiber (http://www.mpipks-dresden.mpg.de/~tisean/. By Antonio Fabio Di Narzo.

ump Uniformly most powerful tests. By Charles J. Geyer and Glen D. Meeden.

varSelRF Variable selection from random forests using both backwards variable elimination (for the selection of small sets of non-redundant variables) and selection based on the importance spectrum (somewhat similar to scree plots; for the selection of large, potentially highly-correlated variables). Main applications in high-dimensional data (e.g., microarray data, and other genomics and proteomics applications). By Ramon Diaz-Uriarte.

varmixt Performs mixture models on the variance for the analysis of gene expression data. By Paul Delmar and Julie Aubert.

wavelets Functions for computing and plotting discrete wavelet transforms (DWT) and maximal overlap discrete wavelet transforms (MODWT), as well as their inverses. Additionally, it contains functionality for computing and plotting wavelet transform filters that are used in the above decompositions as well as multi-resolution analyses. By Eric Aldrich.

Other changes

- Packages **anm**, **fdim**, **haplo.score**, **meanscore**, **mscalib**, **phyloarray**, **seao**, **seao.gui**, **sound**, and **twostage** were moved from the main CRAN section to the Archive.
- Package **gpls** was resurrected from the Archive.
- Packages **rcom** and **GPArotation** were moved from the Devel section of CRAN into the main section.

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

Forthcoming Events: useR! 2006

The second international R user conference 'useR! 2006' will take place at the Wirtschaftsuniversität Wien, Vienna, Austria, June 15-17, 2006.

This second world-meeting of the R user community will focus on

- R as the 'lingua franca' of data analysis and statistical computing,
- providing a platform for R users to discuss and exchange ideas how R can be used to do statistical computations, data analysis, visualization and exciting applications in various fields,
- giving an overview of the new features of the rapidly evolving R project.

The program comprises keynote lectures, user-contributed sessions and pre-conference tutorials. A special highlight of the conference program will be a panel discussion on 'Getting Recognition for Excellence in Computational Statistics' where editors of well established journals will discuss the impact of recent developments in computational statistics on peer-reviewed journal publications.

Keynote Lectures

R has become the standard computing engine in more and more disciplines, both in academia and the business world. How R is used in different areas will be presented in keynote lectures addressing hot topics including

- Data Mining in R: Path Algorithms (Trevor Hastie)
- Cluster Analysis: Past, Present and Future (Brian Everitt)
- R Graphics (Paul Murrell)
- History of S and R (John Chambers)
- Analyzing Marketing Data with an R-based Bayesian Approach (Peter Rossi)
- Psychometrics in R (Jan de Leeuw)
- R on Different Platforms: The useRs' Point of View (Stefano Iacus, Uwe Ligges, Simon Urbanek)
- Teaching Statistics Courses with R (John Fox, Sanford Weisberg).

User-contributed Sessions

The sessions will be a platform to bring together R users, contributers, package maintainers and developers in the S spirit that 'users are developers'. People from different fields will show us how they solve problems with R in fascinating applications. The scientific program is organized by members of the program committee, including Claudio Agostinelli, Roger Bivand, Peter Bühlmann, Ramón Díaz-Uriarte, Sandrine Dudoit, Dirk Eddelbuettel, Frank Harrell, Simon Jackman, Roger Koenker, Uwe Ligges, Andrew Martin, Balasubramanian Narasimhan, Peter Rossi, Anthony Rossini, Gerhard Tutz and Antony Unwin and will cover topics such as

- Applied Statistics & Biostatistics
- Bayesian Statistics
- Bioinformatics
- Econometrics & Finance
- Machine Learning
- Marketing
- Robust Statistics
- Spatial Statistics
- Statistics in the Social and Political Sciences
- Teaching
- Visualization & Graphics
- and many more.

Pre-conference Tutorials

Before the start of the official program, half-day tutorials will be offered on Wednesday, June 14th, including

- Embedding R in Windows Applications (Thomas Baier and Erich Neuwirth)
- Mixed-Effects Models in R (Douglas Bates)
- Analysing Spatial Data in R (Roger Bivand)
- Survival Analysis in R (Rafael E. Borges)
- Survey Analysis in R (Thomas Lumley)
- R Graphics (Paul Murrell)
- Using R for Customer Analytics (Jim Porzak)
- Non-Linear Regression Models in R (Christian Ritz and Jens C. Streibig)
- Bayesian Statistics and Marketing (Peter Rossi)
- Using ESS and Emacs for R Programming and Data Analysis (Anthony Rossini)
- Generalized Additive Models (Simon Wood)
- Rmetrics (Diethelm Wuertz).

After the official presentations, Vienna's famous wine and beer pubs, cafes and restaurants proved to be a very stimulating environment for fruitful discussions at previous meetings of the R community.

Call for Papers

We invite all R users to submit abstracts on topics presenting innovations or exciting applications of R. A web page offering more information on 'useR! 2006', abstract submission and registration is available at http://www.R-project.org/useR-2006/

We hope to meet you in Vienna!

The organizing committee:
Torsten Hothorn, Achim Zeileis, David Meyer, Bettina Grün, Kurt Hornik and Friedrich Leisch
useR-2006@R-project.org

Editor-in-Chief:
Douglas Bates
Department of Statistics
1300 University Ave University of Wisconsin
Madison, WI 53706
USA

Editorial Board:
Paul Murrell and Torsten Hothorn.

Editor Programmer's Niche:
Bill Venables

Editor Help Desk:
Uwe Ligges

Email of editors and editorial board:
firstname.lastname@R-project.org

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/