

The Journal

Volume 4/1, June 2012

A peer-reviewed, open-access publication of the R Foundation
for Statistical Computing

Contents

Editorial 3

Contributed Research Articles


Analysing Seasonal Data 5
MARSS: Multivariate Autoregressive State-space Models for Analyzing Time-series Data . . 11
openair – Data Analysis Tools for the Air Quality Community 20
Foreign Library Interface 30
Vdgraph: A Package for Creating Variance Dispersion Graphs 41
xgrid and R: Parallel Distributed Processing Using Heterogeneous Groups of Apple Com-
puters 45
maxent: An R Package for Low-memory Multinomial Logistic Regression with Support for
Semi-automated Text Classification 56
Sumo: An Authenticating Web Application with an Embedded R Session 60

From the Core

Who Did What? The Roles of R Package Authors and How to Refer to Them 64

News and Notes

Changes in R 70
Changes on CRAN 80
R Foundation News 96

The  Journal is a peer-reviewed publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are licensed under the Creative Commons Attribution 3.0 Unported license (CC BY 3.0, <http://creativecommons.org/licenses/by/3.0/>).

Prospective authors will find detailed and up-to-date submission instructions on the Journal's homepage.

Editor-in-Chief:
Martyn Plummer

Editorial Board:
Heather Turner, Hadley Wickham, and Deepayan Sarkar

Editor Programmer's Niche:
Bill Venables

Editor Help Desk:
Uwe Ligges

Editor Book Reviews:
G. Jay Kerns
Department of Mathematics and Statistics
Youngstown State University
Youngstown, Ohio 44555-0002
USA
gkerns@ysu.edu

R Journal Homepage:
<http://journal.r-project.org/>

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

The R Journal is indexed/abstracted by EBSCO, DOAJ.

Editorial

by *Martyn Plummer*

Earlier this week I was at the “Premières Rencontres R” in Bordeaux, the first francophone meeting of R users (although I confess that I gave my talk in English). This was a very enjoyable experience, and not just because it coincided with the Bordeaux wine festival. The breadth and quality of the talks were both comparable with the International UseR! conferences. It was another demonstration of the extent to which R is used in diverse disciplines.

As R continues to expand into new areas, we see the development of packages designed to address the specific needs of different user communities. This issue of The R Journal contains articles on two such packages. Karl Ropkins and David Carslaw describe how the design of the **openair** package was influenced by the need to make an accessible set of tools available for people in the air quality community who are not experts in R. Elizabeth Holmes and colleagues introduce the **MARSS** package for multivariate autoregressive state-space models. Such models are used in many fields, but the **MARSS** package was motivated by the particular needs of researchers in the natural and environmental sciences,

There are two articles on the use of R outside the desktop computing environment. Sarah Anoke and colleagues describe the use of the Apple Xgrid system to do distributed computing on Mac OS X computers, and Timothy Bergsma and Michael Smith demonstrate the Sumo web application, which includes an embedded R session.

Two articles are of particular interest to developers. Daniel Adler demonstrates a Foreign Function

Interface for R to call arbitrary native functions without the need for C wrapper code. There is also an article from our occasional series “From the Core”, designed to highlight ideas and methods in the words of R development core team members. This article by Kurt Hornik, Duncan Murdoch and Achim Zeileis explains the new facilities for handling bibliographic information introduced in R 2.12.0 and R 2.14.0. I strongly encourage package authors to read this article and implement the changes to their packages. Doing so will greatly facilitate the bibliometric analyses and investigations of the social structure of the R community.

Elsewhere in this issue, we have articles describing the **season** package for displaying and analyzing seasonal data, which is a companion to the book *Analysing Seasonal Data* by Adrian Barnett and Annette Dobson; the **Vdgraph** package for drawing variance dispersion graphs; and the **maxent** package which allows fast multinomial logistic regression with a low memory footprint, and is designed for applications in text classification.

The R Journal continues to be the journal of record for the R project. The usual end matter summarizes recent changes to R itself and on CRAN. It is worth spending some time browsing these sections in order to catch up on changes you may have missed, such as the new CRAN Task View on differential equations. Peter Dalgaard highlighted the importance of this area in his editorial for volume 2, issue 2.

On behalf of the editorial board I hope you enjoy this issue.

Analysing Seasonal Data

by Adrian G Barnett, Peter Baker and Annette J Dobson

Abstract Many common diseases, such as the flu and cardiovascular disease, increase markedly in winter and dip in summer. These seasonal patterns have been part of life for millennia and were first noted in ancient Greece by both Hippocrates and Herodotus. Recent interest has focused on climate change, and the concern that seasons will become more extreme with harsher winter and summer weather. We describe a set of R functions designed to model seasonal patterns in disease. We illustrate some simple descriptive and graphical methods, a more complex method that is able to model non-stationary patterns, and the case-crossover to control for seasonal confounding.

In this paper we illustrate some of the functions of the **season** package (Barnett et al., 2012), which contains a range of functions for analysing seasonal health data. We were motivated by the great interest in seasonality found in the health literature, and the relatively small number of seasonal tools in R (or other software packages). The existing seasonal tools in R are:

- the `baysea` function of the **timsac** package and the `decompose` and `stl` functions of the **stats** package for decomposing a time series into a trend and season;
- the `dynlm` function of the **dynlm** package and the `ssm` function of the **sspir** package for fitting dynamic linear models with optional seasonal components;
- the `arima` function of the **stats** package and the `Arima` function of the **forecast** package for fitting seasonal components as part of an autoregressive integrated moving average (ARIMA) model; and
- the **bfast** package for detecting breaks in a seasonal pattern.

These tools are all useful, but most concern decomposing equally spaced time series data. Our package includes models that can be applied to seasonal patterns in unequally spaced data. Such data are common in observational studies when the timing of responses cannot be controlled (e.g. for a postal survey).

In the health literature much of the analysis of seasonal data uses simple methods such as comparing rates of disease by month or using a cosinor regression model, which assumes a sinusoidal seasonal pattern. We have created functions for these simple,

but often very effective analyses, as we describe below.

More complex seasonal analyses examine non-stationary seasonal patterns that change over time. Changing seasonal patterns in health are currently of great interest as global warming is predicted to make seasonal changes in the weather more extreme. Hence there is a need for statistical tools that can estimate whether a seasonal pattern has become more extreme over time or whether its phase has changed.

Ours is also the first R package that includes the case-crossover, a useful method for controlling for seasonality.

This paper illustrates just some of the functions of the **season** package. We show some descriptive functions that give simple means or plots, and functions whose goal is inference based on generalised linear models. The package was written as a companion to a book on seasonal analysis by Barnett and Dobson (2010), which contains further details on the statistical methods and R code.

Analysing monthly seasonal patterns

Seasonal time series are often based on data collected every month. An example that we use here is the monthly number of cardiovascular disease deaths in people aged ≥ 75 years in Los Angeles for the years 1987–2000 (Samet et al., 2000). Before we examine or plot the monthly death rates we need to make them more comparable by adjusting them to a common month length (Barnett and Dobson, 2010, Section 2.2.1). Otherwise January (with 31 days) will likely have more deaths than February (with 28 or 29).

In the example below the `monthmean` function is used to create the variable `mmean` which is the monthly average rate of cardiovascular disease deaths standardised to a month length of 30 days. As the data set contains the population size (`pop`) we can also standardise the rates to the number of deaths per 100,000 people. The highest death rate is in January (397 per 100,000) and the lowest in July (278 per 100,000).

```
> data(CVD)
> mmean = monthmean(data = CVD,
  resp = CVD$cvd, adjmonth = "thirty",
  pop = pop/100000)
> mmean
      Month  Mean
January 396.8
February 360.8
  March 327.3
  April 311.9
```

May 294.9
 June 284.5
 July 277.8
 August 279.2
 September 279.1
 October 292.3
 November 313.3
 December 368.5

Plotting monthly data

We can plot these standardised means in a circular plot using the `plotCircular` function:

```
> plotCircular(areal = mmean$mean,
  dp = 1, labels = month.abb,
  scale = 0.7)
```

This produces the circular plot shown in Figure 1. The numbers under each month are the adjusted averages, and the area of each segment is proportional to this average.

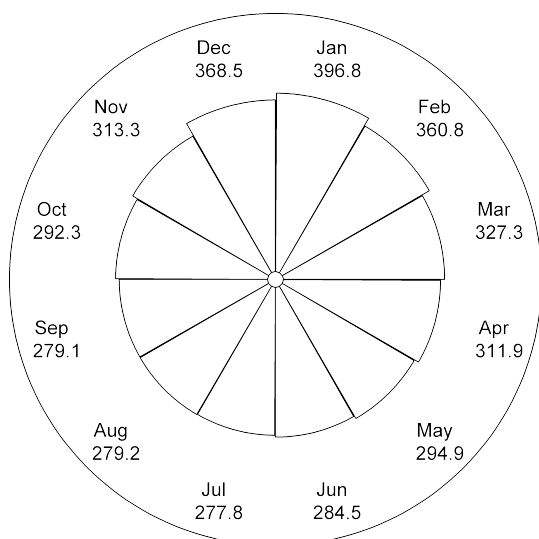


Figure 1: A circular plot of the adjusted monthly mean number of cardiovascular deaths in Los Angeles in people aged ≥ 75 , 1987–2000.

The peak in the average number of deaths is in January, and the low is six months later in July indicating an annual seasonal pattern. If there were no seasonal pattern we would expect the averages in each month to be equal, and so the plot would be perfectly circular. The seasonal pattern is somewhat non-symmetric, as the decrease in deaths from January to July does not mirror the seasonal increase from July to January. This is because the increase in deaths does not start in earnest until October.

Circular plots are also useful when we have an observed and expected number of observations in each month. As an example, Figure 2 shows the number of Australian Football League players by their month of birth.

their month of birth (for the 2009 football season) and the expected number of births per month based on national data. For this example we did not adjust for the unequal number of days in the months because we can compare the observed numbers to the expected (which are also based on unequal month lengths). Using the expected numbers also shows any seasonal pattern in the national birth numbers. In this example there is a very slight decrease in births in November and December.

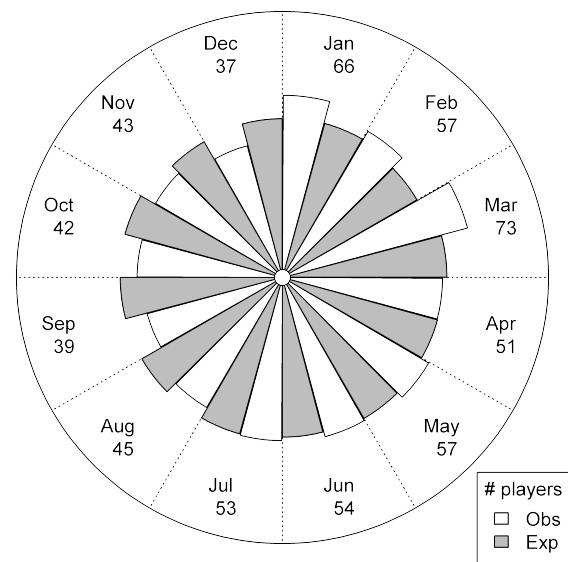


Figure 2: A circular plot of the monthly number of Australian Football League players by their month of birth (*white segments*) and the expected numbers based on national data for men born in the same period (*grey segments*). Australian born players in the 2009 football season.

The figure shows the greater than expected number of players born in January to March, and the fewer than expected born in August to December. The numbers around the outside are the observed number of players. The code to create this plot is:

```
> data(AFL)
> plotCircular(areal = AFL$players,
  area2 = AFL$expected, scale = 0.72,
  labels = month.abb, dp = 0, lines = TRUE,
  auto.legend = list(
  labels = c("Obs", "Exp"),
  title = "# players"))
```

The key difference from the code to create the previous circular plot is that we have given values for both `areal` and `area2`. The `'lines = TRUE'` option added the dotted lines between the months. We have also included a legend.

As well as a circular plot we also recommend a time series plot for monthly data, as these plots are useful for highlighting the consistency in the seasonal pattern and possibly also the secular trend and

any unusual observations. For the cardiovascular example data a time series plot is created using

```
> plot(CVD$yrmon, CVD$cvd, type = 'o',
      pch = 19,
      ylab = 'Number of CVD deaths per month',
      xlab = 'Time')
```

The result is shown in Figure 3. The January peak in CVD was clearly larger in 1992 and 1994 compared with 1991, 1993 and 1995. There also appears to be a slight downward trend from 1987 to 1992.

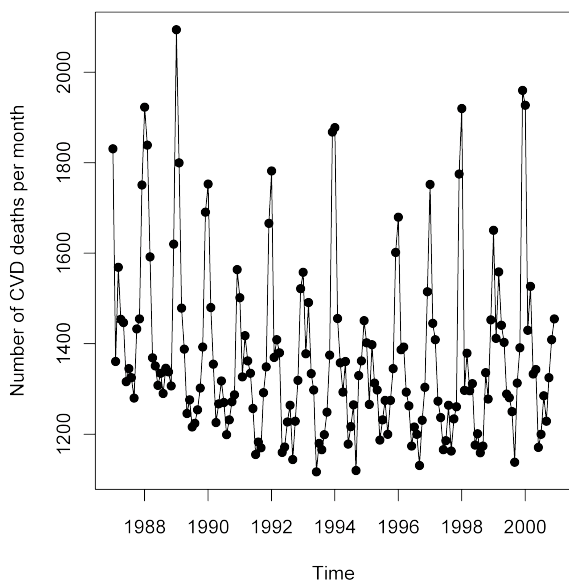


Figure 3: Monthly number of cardiovascular deaths in Los Angeles for people aged ≥ 75 , 1987–2000.

Modelling monthly data

A simple and popular statistical model for examining seasonal patterns in monthly data is to use a simple linear regression model (or generalised linear model) with a categorical variable of month. The code below fits just such a model to the cardiovascular disease data and then plots the rate ratios (Figure 4).

```
> mmodel = monthglm(formula = cvd ~ 1,
  data = CVD, family = poisson(),
  offsetpop = pop/100000,
  offsetmonth = TRUE, refmonth = 7)
> plot(mmodel)
```

As the data are counts we used a Poisson model. We adjusted for the unequal number of days in the month by using an offset (`offsetmonth = TRUE`), which divides the number of deaths in each month by the number of days in each month to give a daily rate. The reference month was set to July (`refmonth = 7`). We could have added other variables to the model, by adding them to the right hand side of the equation (e.g. `'formula = cvd ~ year'` to include a linear trend for year).

The plot in Figure 4 shows the mean rate ratios and 95% confidence intervals. The dotted horizontal reference line is at the rate ratio of 1. The mean rate of deaths in January is 1.43 times the rate in July. The rates in August and September are not statistically significantly different to the rates in July, as the confidence intervals in these months both cross 1.

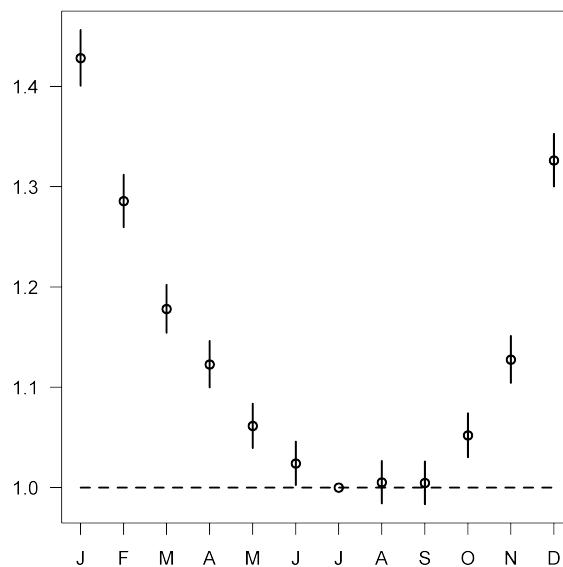


Figure 4: Mean rate ratios and 95% confidence intervals of cardiovascular disease deaths using July as a reference month.

Cosinor

The previous model assumed that the rate of cardiovascular disease varied arbitrarily in each month with no smoothing of or correlation between neighbouring months. This is an unlikely assumption for this seasonal pattern (Figure 4). The advantage of using arbitrary estimates is that it does not constrain the shape of the seasonal pattern. The disadvantage is a potential loss of statistical power. Models that assume some parametric seasonal pattern will have a greater power when the parametric model is correct. A popular parametric seasonal model is the cosinor model (Barnett and Dobson, 2010, Chapter 3), which is based on a sinusoidal pattern,

$$s_t = A \cos\left(\frac{2\pi t}{c} - P\right), \quad t = 1, \dots, n,$$

where A is the amplitude of the sinusoid and P is its phase, c is the length of the seasonal cycle (e.g. $c = 12$ for monthly data with an annual seasonal pattern), t is the time of each observation and n is the total number of times observed. The amplitude tells us the size of the seasonal change and the phase tells us where it peaks. The sinusoid assumes a smooth seasonal pattern that is symmetric about its peak (so the rate of the seasonal increase in disease is equal to the decrease). We fit the Cosinor as part of a generalised linear model.

The example code below fits a cosinor model to the cardiovascular disease data. The results are for each month, so we used the `'type = 'monthly''` option with `'date = month'`.

```
> res = cosinor(cvd ~ 1, date = month,
  data = CVD, type = 'monthly',
  family = poisson(), offsetmonth = TRUE)
> summary(res)
Cosinor test
Number of observations = 168
Amplitude = 232.34 (absolute scale)
Phase: Month = 1.3
Low point: Month = 7.3
Significant seasonality based on adjusted
significance level of 0.025 = TRUE
```

We again adjusted for the unequal number of days in the months using an `offset` (`offsetmonth = TRUE`). The amplitude is 232 deaths which has been given on the absolute scale and the phase is estimated as 1.27 months (early January).

An advantage of these cosinor models is that they can be fitted to unequally spaced data. The example code below fits a cosinor model to data from a randomised controlled trial of physical activity with data on body mass index (BMI) at baseline (Eakin et al., 2009). Subjects were recruited as they became available and so the measurement dates are not equally spaced. In the example below we test for a sinusoidal seasonal pattern in BMI.

```
> data(exercise)
> res = cosinor(bmi ~ 1, date = date,
  type = 'daily', data = exercise,
  family = gaussian())
> summary(res)
Cosinor test
Number of observations = 1152
Amplitude = 0.3765669
Phase: Month = November , day = 18
Low point: Month = May , day = 19
Significant seasonality based on adjusted
significance level of 0.025 = FALSE
```

Body mass index has an amplitude of 0.38 kg/m² which peaks on 18 November, but this increase is not statistically significant. In this example we used `'type = 'daily''` as subjects' results related to a specific date (`'date = date'` specifies the day when they were measured). Thus the phase for body mass index is given on a scale of days, whereas the phase for cardiovascular death was given on a scale of months.

Non-stationary cosinor

The models illustrated so far have all assumed a stationary seasonal pattern, meaning a pattern that does not change from year to year. However, seasonal patterns in disease may gradually change because of

changes in an important exposure. For example, improvements in housing over the 20th century are part of the reason for a decline in the winter peak in mortality in London (Carson et al., 2006).

To fit a non-stationary cosinor we expand the previous sinusoidal equation thus

$$s_t = A_t \cos\left(\frac{2\pi t}{c} - P_t\right), \quad t = 1, \dots, n$$

so that both the amplitude and phase of the cosinor are now dependent on time. The key unknown is the extent to which these parameters will change over time. Using our `nscosinor` function the user has some control over the amount of change and a number of different models can be tested assuming different levels of change. The final model should be chosen using model fit diagnostics and residual checks (available in the `seasrescheck` function).

The `nscosinor` function uses the Kalman filter to decompose the time series into a trend and seasonal components (West and Harrison, 1997, Chapter 8), so can only be applied to equally spaced time series data. The code below fits a non-stationary sinusoidal model to the cardiovascular disease data (using the counts adjusted to the average month length, `adj`).

```
> nsmodel = nscosinor(data = CVD,
  response = adj, cycles = 12, niters = 5000,
  burnin = 1000, tau = c(10, 500), inits = 1)
```

The model uses Markov chain Monte Carlo (MCMC) sampling, so we needed to specify the number of iterations (`niters`), the number discarded as a burn-in (`burnin`), and an initial value for each seasonal component (`inits`). The `cycles` gives the frequency of the sinusoid in units of time, in this case a seasonal pattern that completes a cycle in 12 months. We can fit multiple seasonal components, for example 6 and 12 month seasonal patterns would be fitted using `'cycles = c(6, 12)'`. The `tau` are smoothing parameters, with `tau[1]` for the trend, `tau[2]` for the first seasonal parameter, `tau[3]` for the second seasonal parameter. They are fixed values that scale the time between observations. Larger values allow more time between observations and hence create a more flexible spline. The ideal values for `tau` should be chosen using residual checking and trial and error.

The estimated seasonal pattern is shown in Figure 5. The mean amplitude varies from around 230 deaths (winter 1989) to around 180 deaths (winter 1995), so some winters were worse than others. Importantly the results did not show a steady decline in amplitude, so over this period seasonal deaths continued to be a problem despite any improvements in health care or housing. However, the residuals from this model do show a significant seasonal pattern (checked using the `seasrescheck` function). This residual seasonal pattern is caused because the

seasonal pattern in cardiovascular deaths is non-sinusoidal (as shown in Figure 1) with a sharper increase in deaths than decline. The model assumed a sinusoidal pattern, albeit a non-stationary one. A better fit might be achieved by adding a second seasonal cycle at a shorter frequency, such as 6 months.

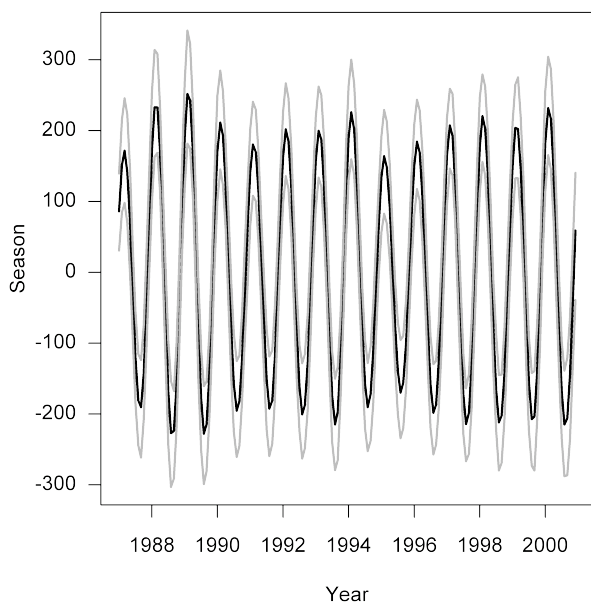


Figure 5: Estimated non-stationary seasonal pattern in cardiovascular disease deaths for Los Angeles, 1987–2000. Mean (black line) and 95% confidence interval (grey lines).

Case-crossover

In some circumstances seasonality is not the focus of investigation, but is important because its effects need to be taken into account. This could be because both the outcome and the exposure have an annual seasonal pattern, but we are interested in associations at a different frequency (e.g. daily).

The case-crossover can be used for individual-level data, e.g. when the data are individual cases with their date of heart attack and their recent exposure. However, we are concerned with regularly spaced time-series data, where the data are grouped, e.g. the number of heart attacks on each day in a year.

The case-crossover is a useful time series method for controlling for seasonality (Maclure, 1991). It is similar to the matched case-control design, where the exposure of cases with the disease are compared with one or more matched controls without the disease. In the case-crossover, cases act as their own control, since exposures are compared on case and control days (also known as index and referent days). The case day will be the day on which an event occurred (e.g. death), and the control days will be nearby days in the same season as the exposure but with a possibly different exposure. This means the cases and controls are matched for season, but not for some

other short-term change in exposure such as air pollution or temperature. A number of different case-crossover designs for time-series data have been proposed. We used the time-stratified method as it is a localisable and ignorable design that is free from overlap bias while other referent window designs that are commonly used in the literature (e.g. symmetric bi-directional) are not (Janes et al., 2005). Using this design the data is broken into a number of fixed strata (e.g. 28 days or the months of the year) and the case and control days are compared within the same strata.

The code below applies a case-crossover model to the cardiovascular disease data. In this case we use the daily cardiovascular disease (with the number of deaths on every day) rather than the data used above which used the number of cardiovascular deaths in each month. The independent variables are mean daily ozone (`o3mean`, which we first scale to a 10 unit increase) and temperature (`tmpd`). We also control for day of the week (using Sunday as the reference category). For this model we are interested in the effect of day-to-day changes in ozone on the day-to-day changes in mortality.

```
> data(CVDdaily)
> CVDdaily$o3mean = CVDdaily$o3mean / 10
> cmodel = casecross(cvd ~ o3mean + tmpd +
  Mon + Tue + Wed + Thu + Fri + Sat,
  data = CVDdaily)
> summary(cmodel, digits = 2)
Time-stratified case-crossover with a stratum
length of 28 days
Total number of cases 230695
Number of case days with available control
days 5114
Average number of control days per case day 23.2

Parameter Estimates:
      coef exp(coef) se(coef)      z Pr(>|z|)
o3mean -0.0072    0.99 0.00362 -1.98 4.7e-02
tmpd    0.0024    1.00 0.00059  4.09 4.3e-05
Mon     0.0323    1.03 0.00800  4.04 5.3e-05
Tue     0.0144    1.01 0.00808  1.78 7.5e-02
Wed    -0.0146    0.99 0.00807 -1.81 7.0e-02
Thu    -0.0118    0.99 0.00805 -1.46 1.4e-01
Fri     0.0065    1.01 0.00806  0.81 4.2e-01
Sat     0.0136    1.01 0.00788  1.73 8.4e-02
```

The default stratum length is 28, which means that cases and controls are compared in blocks of 28 days. This stratum length should be short enough to remove any seasonal pattern in ozone and temperature. Ozone is formed by a reaction between other air pollutants and sunlight and so is strongly seasonal with a peak in summer. Cardiovascular mortality is at its lowest in summer as warmer temperatures lower blood pressures and prevent flu outbreaks. So without removing these seasonal patterns we might find a significant negative association between ozone and mortality. The above results suggest a marginally significant negative association be-

tween ozone and mortality, as the odds ratio for a ten unit increase in ozone is $\exp(-0.0072) = 0.993$ (p-value = 0.047). This may indicate that we have not sufficiently controlled for season and so should reduce the stratum length using the `stratalength` option.

As well as matching cases and controls by stratum, it is also possible to match on another confounder. The code below shows a case-crossover model that matched case and control days by a mean temperature of ± 1 degrees Fahrenheit.

```
> mmodel = casecross(cvd ~ o3mean +
  Mon + Tue + Wed + Thu + Fri + Sat,
  matchconf = 'tmpd', confrange = 1,
  data = CVDdaily)
> summary(mmodel, digits = 2)
Time-stratified case-crossover with a stratum
length of 28 days
Total number of cases 205612
Matched on tmpd plus/minus 1
Number of case days with available control
days 4581
Average number of control days per case day 5.6
```

Parameter Estimates:

	coef	exp(coef)	se(coef)	z	Pr(> z)
o3mean	0.0046	1	0.0043	1.07	2.8e-01
Mon	0.0461	1	0.0094	4.93	8.1e-07
Tue	0.0324	1	0.0095	3.40	6.9e-04
Wed	0.0103	1	0.0094	1.10	2.7e-01
Thu	0.0034	1	0.0093	0.36	7.2e-01
Fri	0.0229	1	0.0094	2.45	1.4e-02
Sat	0.0224	1	0.0092	2.45	1.4e-02

By matching on temperature we have restricted the number of available control days, so there are now only an average of 5.6 control days per case, compared with 23.2 days in the previous example. Also there are now only 4581 case days with at least one control day available compared with 5114 days for the previous analysis. So 533 days have been lost (and 25,083 cases), and these are most likely the days with unusual temperatures that could not be matched to any other days in the same stratum. We did not use temperature as an independent variable in this model, as it has been controlled for by the matching. The odds ratio for a ten unit increase in ozone is now positive ($OR = \exp(0.0046) = 1.005$) although not statistically significant (p-value = 0.28).

It is also possible to match cases and control days by the day of the week using the `'matchdow = TRUE'` option.

Bibliography

- A. G. Barnett, P. Baker, and A. J. Dobson. *season: Seasonal analysis of health data*, 2012. URL <http://CRAN.R-project.org/package=season>. R package version 0.3-1.
- A. G. Barnett and A. J. Dobson. *Analysing Seasonal Health Data*. Springer, 2010.
- C. Carson, S. Hajat, B. Armstrong, and P. Wilkinson. Declining vulnerability to temperature-related mortality in London over the 20th Century. *Am J Epidemiol*, 164(1):77–84, 2006.
- E. Eakin, M. Reeves, S. Lawler, N. Graves, B. Oldenburg, C. DelMar, K. Wilke, E. Winkler, and A. Barnett. Telephone counseling for physical activity and diet in primary care patients. *Am J of Prev Med*, 36(2):142–149, 2009.
- H. Janes, L. Sheppard, and T. Lumley. Case-crossover analyses of air pollution exposure data: Referent selection strategies and their implications for bias. *Epidemiology*, 16(6):717–726, 2005.
- M. Maclure. The case-crossover design: a method for studying transient effects on the risk of acute events. *Am J Epidemiol*, 133(2):144–153, 1991.
- J. Samet, F. Dominici, S. Zeger, J. Schwartz, and D. Dockery. The national morbidity, mortality, and air pollution study, part I: Methods and methodologic issues. 2000.
- M. West and J. Harrison. *Bayesian Forecasting and Dynamic Models*. Springer Series in Statistics. Springer, New York; Berlin, 2nd edition, 1997.

Adrian Barnett
 School of Public Health
 Queensland University of Technology
 Australia
a.barnett@qut.edu.au

Peter Baker
 School of Population Health
 University of Queensland
 Australia

Annette Dobson
 School of Population Health
 University of Queensland
 Australia

MARSS: Multivariate Autoregressive State-space Models for Analyzing Time-series Data

by Elizabeth E. Holmes, Eric J. Ward, Kellie Wills

Abstract MARSS is a package for fitting multivariate autoregressive state-space models to time-series data. The MARSS package implements state-space models in a maximum likelihood framework. The core functionality of MARSS is based on likelihood maximization using the Kalman filter/smoothen, combined with an EM algorithm. To make comparisons with other packages available, parameter estimation is also permitted via direct search routines available in 'optim'. The MARSS package allows data to contain missing values and allows a wide variety of model structures and constraints to be specified (such as fixed or shared parameters). In addition to model-fitting, the package provides bootstrap routines for simulating data and generating confidence intervals, and multiple options for calculating model selection criteria (such as AIC).

The **MARSS** package (Holmes et al., 2012) is an R package for fitting linear multivariate autoregressive state-space (MARSS) models with Gaussian errors to time-series data. This class of model is extremely important in the study of linear stochastic dynamical systems, and these models are used in many different fields, including economics, engineering, genetics, physics and ecology. The model class has different names in different fields; some common names are dynamic linear models (DLMs) and vector autoregressive (VAR) state-space models. There are a number of existing R packages for fitting this class of models, including **sspir** (Dethlefsen et al., 2009) for univariate data and **d1m** (Petris, 2010), **dse** (Gilbert, 2009), **KFAS** (Helske, 2011) and **FKF** (Luethi et al., 2012) for multivariate data. Additional packages are available on other platforms, such as SsfPack (Durbin and Koopman, 2001), EViews (www.eviews.com) and Brodgar (www.brodgar.com). Except for Brodgar and **sspir**, these packages provide maximization of the likelihood surface (for maximum-likelihood parameter estimation) via quasi-Newton or Nelder-Mead type algorithms. The **MARSS** package was developed to provide an alternative maximization algorithm, based instead on an Expectation-Maximization (EM) algorithm and to provide a standardized model-specification framework for fitting different model structures.

The **MARSS** package was originally developed for researchers analyzing data in the natural and environmental sciences, because many of the problems often encountered in these fields are not commonly encountered in disciplines like engineering or finance. Two typical problems are high fractions of irregularly spaced missing observations and observation error variance that cannot be estimated or known a priori (Schnute, 1994). Packages developed for other fields did not always allow estimation of the parameters of interest to ecologists because these parameters are always fixed in the package authors' field or application. The **MARSS** package was developed to address these issues and its three main differences are summarized as follows.

First, maximum-likelihood optimization in most packages for fitting state-space models relies on quasi-Newton or Nelder-Mead direct search routines, such as provided in **optim** (for **d1m**) or **nlm** (for **dse**). Multidimensional state-space problems often have complex, non-linear likelihood surfaces. For certain types of multivariate state-space models, an alternative maximization algorithm exists; though generally slower for most models, the EM algorithm (Metaxoglou and Smith, 2007; Shumway and Stoffer, 1982), is considerably more robust than direct search routines (this is particularly evident with large amounts of missing observations). To date, no R package for the analysis of multivariate state-space models has implemented the EM algorithm for maximum-likelihood parameter estimation (**sspir** implements it for univariate models). In addition, the **MARSS** package implements an EM algorithm for constrained parameter estimation (Holmes, 2010) to allow fixed and shared values within parameter matrices. To our knowledge, this constrained EM algorithm is not implemented in any package, although the Brodgar package implements a limited version for dynamic factor analysis.

Second, model specification in the **MARSS** package has a one-to-one relationship to a MARSS model as written in matrix form on paper. Any model that can be written in MARSS form can be fitted without extra code by the user. In contrast, other packages require users to write unique functions in matrix form (a non-trivial task for many non-expert R users). For example, while **d1m** includes linear and polynomial univariate models, multivariate regression is not readily accessible without these custom functions; in **MARSS**, all models written in matrix form are fitted using the same model specification.

The **MARSS** package also allows degenerate multivariate models to be fitted, which means that some or all observation or process variances can be set to 0. This allows users to include deterministic features in their models and to rewrite models with longer lags or moving averaged errors as a MARSS model.

Third, the **MARSS** package provides algorithms for computing model selection criteria that are specific for state-space models. Model selection criteria are used to quantify the data support for different model and parameter structures by balancing the ability of the model to fit the data against the flexibility of the model. The criteria computed in the **MARSS** package are based on Akaike's Information Criterion (AIC). Models with the lowest AIC are interpreted as receiving more data support. While AIC and its small sample corrected version AICc are easily calculated for fixed effects models (these are standard output for `lm` and `glm`, for instance), these criteria are biased for hierarchical or state-space autoregressive models. The **MARSS** package provides an unbiased AIC criterion via innovations bootstrapping (Cavanaugh and Shumway, 1997; Stoffer and Wall, 1991) and parametric bootstrapping (Holmes, 2010). The package also provides functions for approximate and bootstrap confidence intervals, and bias correction for estimated parameters.

The package comes with an extensive user guide that introduces users to the package and walks the user through a number of case studies involving ecological data. The selection of case studies includes estimating trends with univariate and multivariate data, making inferences concerning spatial structure with multi-location data, estimating inter-species interaction strengths using multi-species data, using dynamic factor analysis to reduce the dimension of a multivariate dataset, and detecting structural breaks in data sets. Though these examples have an ecological focus, the analysis of multivariate time series models is cross-disciplinary work and researchers in other fields will likely benefit from these examples.

The MARSS model

The MARSS model includes a process model and an observation model. The process component of a MARSS model is a multivariate first-order autoregressive (MAR-1) process. The multivariate process model takes the form

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t; \quad \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \quad (1)$$

The \mathbf{x} is an $m \times 1$ vector of state values, equally spaced in time, and \mathbf{B} , \mathbf{u} and \mathbf{Q} are the state process parameters. The $m \times m$ matrix \mathbf{B} allows interaction between state processes; the diagonal elements of \mathbf{B} can also be interpreted as coefficients of auto-regression in the state vectors through time. The vector \mathbf{u} describes the mean trend or mean level (de-

pending on the \mathbf{B} structure), and the correlation of the process deviations is determined by the structure of the matrix \mathbf{Q} . In version 2.x of the **MARSS** package, the \mathbf{B} and \mathbf{Q} parameters are time-invariant; however, in version 3.x, all parameters can be time-varying and also the \mathbf{u} can be moved into the \mathbf{x} term to specify models with an auto-regressive trend process (called a stochastic level model).

Written out for two state processes, the MARSS process model would be:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{t-1} + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t, \\ \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \sim \text{MVN} \left(0, \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix} \right)$$

Some of the parameter elements will generally be fixed to ensure identifiability. Within the MAR-1 form, MAR-p or autoregressive models with lag-p and models with exogenous factors (covariates) can be included by properly defining the \mathbf{B} , \mathbf{x} and \mathbf{Q} elements. See for example Tsay (2010) where the formulation of various time-series models as MAR-1 models is covered.

The initial state vector is specified at $t = 0$ or $t = 1$ as

$$\mathbf{x}_0 \sim \text{MVN}(\boldsymbol{\pi}, \boldsymbol{\Lambda}) \quad \text{or} \quad \mathbf{x}_1 \sim \text{MVN}(\boldsymbol{\pi}, \boldsymbol{\Lambda}) \quad (2)$$

where $\boldsymbol{\pi}$ is the mean of the initial state distribution or a constant. $\boldsymbol{\Lambda}$ specifies the variance of the initial states or is specified as 0 if the initial state is treated as fixed (but possibly unknown).

The multivariate observation component in a MARSS model is expressed as

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t; \quad \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \quad (3)$$

where \mathbf{y}_t is an $n \times 1$ vector of observations at time t , \mathbf{Z} is an $n \times m$ matrix, \mathbf{a} is an $n \times 1$ matrix, and the correlation structure of observation errors is specified with the matrix \mathbf{R} . Including \mathbf{Z} and \mathbf{a} is not required for every model, but these parameters are used when some state processes are observed multiple times (perhaps with different scalings) or when observations are linear combinations of the state processes (such as in dynamic factor analysis). Note that time steps in the model are equidistant, but there is no requirement that there be an observation at every time step; \mathbf{y} may have missing values scattered throughout it.

Written out for three observation processes and two state processes, the MARSS observation model would be

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}_t = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \\ z_{31} & z_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t \\ \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t \sim \text{MVN} \left(0, \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \right)$$

Again, some of the parameter elements will generally be fixed to ensure identifiability.

The MARSS process and observation models are flexible and many multivariate time-series models can be rewritten in MARSS form. See textbooks on time series analysis where reformulating AR-p, ARMA-p as a MARSS model is covered (such as Tsay, 2010; Harvey, 1989).

Package overview

The package is designed to fit MARSS models with fixed and shared elements within the parameter matrices. The following shows such a model using a mean-reverting random walk model with three observation time series as an example:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{t-1} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \quad (4a)$$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \sim \text{MVN} \left(0, \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix} \right) \quad (4b)$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_0 \sim \text{MVN} \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \quad (4c)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t + \begin{bmatrix} a_1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t \quad (4d)$$

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t \sim \text{MVN} \left(0, \begin{bmatrix} r_1 & 0 & 0 \\ 0 & r_2 & 0 \\ 0 & 0 & r_2 \end{bmatrix} \right) \quad (4e)$$

Notice that many parameter elements are fixed, while others are shared (have the same symbol).

Model specification

Model specification has a one-to-one relationship to the model written on paper in matrix form (Equation 4). The user passes in a list which includes an element for each parameter in the model: B, U, Q, Z, A, R, x0, V0. The list element specifies the form of the corresponding parameter in the model.

The most general way to specify a parameter form is to use a list matrix. The list matrix allows one to combine fixed and estimated elements in one's parameter specification. For example, to specify the parameters in Equation 4, one would write the following list matrices:

```
> B1 = matrix(list("b", 0, 0, "b"), 2, 2)
> U1 = matrix(0, 2, 1)
> Q1 = matrix(c("q11", "q12", "q12", "q22"), 2, 2)
> Z1 = matrix(c(1, 0, 1, 0, 1, 0), 3, 2)
> A1 = matrix(list("a1", 0, 0), 3, 1)
> R1 = matrix(list("r1", 0, 0, 0, "r2", 0,
+ 0, 0, "r2"), 3, 3)
> pi1 = matrix(0, 2, 1)
> V1 = diag(1, 2)
```

```
> model.list = list(B = B1, U = U1, Q = Q1, Z = Z1,
+ A = A1, R = R1, x0 = pi1, V0 = V1)
```

When printed at the R command line, each parameter matrix looks exactly like the parameters as written out in Equation 4. Numeric values are fixed and character values are names of elements to be estimated. Elements with the same character name are constrained to be equal (no sharing across parameter matrices, only within).

List matrices allow the most flexible model structures, but MARSS also has text shortcuts for many common model structures. The structure of the variance-covariance matrices in the model are specified via the Q, R, and V0 components, respectively. Common structures are errors independent with one variance on the diagonal (specified with "diagonal and equal") or all variances different ("diagonal and unequal"); errors correlated with the same correlation parameter ("equalvarcov") or correlated with each process having unique correlation parameters ("unconstrained"), or all zero ("zero"). Common structures for the matrix B are an identity matrix ("identity"); a diagonal matrix with one value on the diagonal ("diagonal and equal") or all different values on the diagonal ("diagonal and unequal"), or all values different ("unconstrained"). Common structures for the u and a parameters are all equal ("equal"); all unequal ("unequal" or "unconstrained"), or all zero ("zero").

The Z matrix is idiosyncratic and there are fewer shortcuts for its specification. One common form for Z is a design matrix, composed of 0s and 1s, in which each row sum equals 1. This is used when each y in y is associated with one x in x (one x may be associated with multiple y's). In this case, Z can be specified using a factor of length n. The factor specifies to which x each of the n y's correspond. For example, the Z in Equation 4 could be specified factor(c(1,2,1)) or factor(c("a", "b", "a")).

Model fitting with the MARSS function

The main package function is MARSS. This function fits a MARSS model (Equations 1 and 3) to a matrix of data and returns the maximum-likelihood estimates for the B, u, Q, Z, a, R, π, and Λ parameters or more specifically, the free elements in those parameters since many elements will be fixed for a given model form. The basic MARSS call takes the form:

```
> MARSS(data, model = model.list)
```

where model.list has the form shown in the R code above. The data must be passed in as an $n \times T$ matrix, that is time goes across columns. A vector is not a matrix, nor is a data frame. A matrix of 3 inputs ($n = 3$) measured for 6 time steps might look like

$$\mathbf{y} = \begin{bmatrix} 1 & 2 & \text{NA} & \text{NA} & 3.2 & 8 \\ 2 & 5 & 3 & \text{NA} & 5.1 & 5 \\ 1 & \text{NA} & 2 & 2.2 & \text{NA} & 7 \end{bmatrix}$$

where NA denotes a missing value. Note that the time steps are equidistant, but there may not be data at each time step, thus the observations are not constrained to be equidistant.

The only required argument to `MARSS` is a matrix of data. The `model` argument is an optional argument, although typically included, which specifies the form of the MARSS model to be fitted (if left off, a default form is used). Other `MARSS` arguments include initial parameter values (`inits`), a non-default value for missing values (`miss.value`), whether or not the model should be fitted (`fit`), whether or not output should be verbose (`silent`), and the method for maximum-likelihood estimation (`method`). The default method is the EM algorithm (`method = "kem"`), but the quasi-Newton method BFGS is also allowed (`method = "BFGS"`). Changes to the default fitting algorithm are specified with `control`; for example, `control$minit` is used to change the minimum number of iterations used in the maximization algorithm. Use `?MARSS` to see all the `control` options.

Each call to the `MARSS` function returns an object of class `"marssMLE"`, an estimation object. The `marssMLE` object is a list with the following elements: the estimated parameter values (`par`), Kalman filter and smoother output (`kf`), convergence diagnostics (`convergence`, `numIter`, `errors`, `logLik`), basic model selection metrics (`AIC`, `AICc`), and a model object `model` which contains both the `MARSS` model specification and the data. Further list elements, such as bootstrap AIC or confidence intervals, are added with functions that take a `marssMLE` object as input.

Model selection and confidence intervals

One of the most commonly used model selection tools in the maximum-likelihood framework is Akaike's Information Criterion (AIC) or the small sample variant (AICc). Both versions are returned with the call to `MARSS`. A state-space specific model selection metric, `AICb` (Cavanaugh and Shumway, 1997; Stoffer and Wall, 1991), can be added to a `marssMLE` object using the function `MARSSaic`.

The function `MARSSparamCIs` is used to add confidence intervals and bias estimates to a `marssMLE` object. Confidence intervals may be computed by resampling the Kalman innovations if all data are present (innovations bootstrapping) or implementing a parametric bootstrap if some data are missing (parametric bootstrapping). All bootstrapping in the `MARSS` package is done with the lower-level function `MARSSboot`. Because bootstrapping can be time-consuming, `MARSSparamCIs` also allows approximate confidence intervals to be calculated with a numerically estimated Hessian matrix. If bootstrapping is used to generate confidence intervals, `MARSSparamCIs` will also return a bootstrap estimate of parameter bias.

Estimated state trajectories

In addition to outputting the maximum-likelihood estimates of the model parameters, the `MARSS` call will also output the maximum-likelihood estimates of the state trajectories (the `x` in the `MARSS` model) conditioned on the entire dataset. These states represent output from the Kalman smoother using the maximum-likelihood parameter values. The estimated states are in the `states` element in the `marssMLE` object output from a `MARSS` call. The standard errors of the state estimates are in element `states.se` of the `marssMLE` object. For example, if a `MARSS` model with one state ($m = 1$) was fitted to data and the output placed in `fit`, the state estimates and ± 2 standard errors could be plotted with the following code:

```
> plot(fit$states, type = "l", lwd = 2)
> lines(fit$states - 2*fit$states.se)
> lines(fit$states + 2*fit$states.se)
```

Example applications

To demonstrate the `MARSS` package functionality, we show a series of short examples based on a few of the case studies in the `MARSS` User Guide. The user guide includes much more extensive analysis and includes many other case studies. All the case studies are based on analysis of ecological data, as this was the original motivation for the package.

Trend estimation

A commonly used model in ecology describing the dynamics of a population experiencing density-dependent growth is the Gompertz model (Reddingius and den Boer, 1989; Dennis and Taper, 1994). The population dynamics are stochastic and driven by the combined effects of environmental variation and random demographic variation.

The log-population sizes from this model can be described by an AR-1 process:

$$x_t = bx_{t-1} + u + w_t; \quad w_t \sim \text{Normal}(0, \sigma^2) \quad (5)$$

In the absence of density dependence ($b = 1$), the exponential growth rate or rate of decline of the population is controlled by the parameter u ; when the population experiences density-dependence ($|b| < 1$), the population will converge to an equilibrium $u/(1 - b)$. The initial states are given by

$$x_0 \sim \text{Normal}(\pi, \lambda) \quad (6)$$

The observation process is described by

$$y_t = x_t + v_t; \quad v_t \sim \text{Normal}(0, \eta^2) \quad (7)$$

where y is a vector of observed log-population sizes. The bias parameter, a , has been set equal to 0.

This model with b set to 1 is a standard model used for species of concern (Dennis et al., 1991) when the population is well below carrying capacity and the goal is to estimate the underlying trend (u) for use in population viability analysis (Holmes, 2001; Holmes et al., 2007; Humbert et al., 2009). To illustrate a simple trend analysis, we use the `graywhales` dataset included in the `MARSS` package. This data set consists of 24 abundance estimates of eastern North Pacific gray whales (Gerber et al., 1999). This population was depleted by commercial whaling prior to 1900 and has steadily increased since the first abundance estimates were made in the 1950s (Gerber et al., 1999). The dataset consists of 24 annual observations over 39 years, 1952–1997; years without an estimate are denoted as NA. The time step between observations is one year and the goal is to estimate the annual rate of increase (or decrease).

After log-transforming the data, maximum-likelihood parameter estimates can be calculated using `MARSS`:

```
> data(graywhales)
> years = graywhales[,1]
> loggraywhales = log(graywhales[,2])
> kem = MARSS(loggraywhales)
```

The `MARSS` wrapper returns an object of class "marssMLE", which contains the parameter estimates (`kem$par`), state estimates (`kem$states`) and their standard errors (`kem$states.se`). The state estimates can be plotted with ± 2 standard errors, along with the original data (Figure 1).

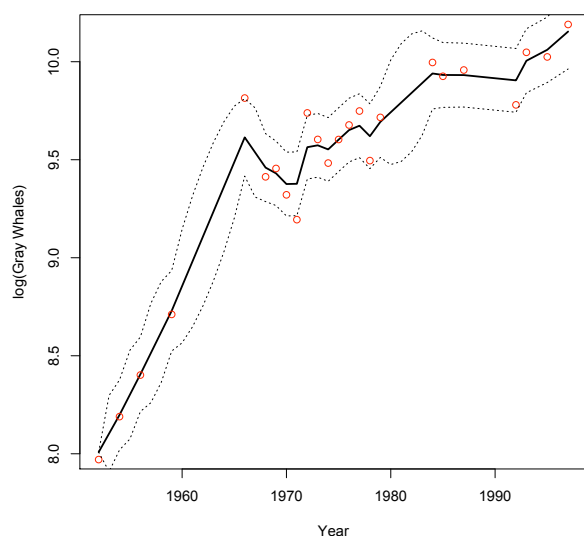


Figure 1: Maximum-likelihood estimates of eastern North Pacific gray whales (solid line) population size, with data (points) and 2 standard errors (dashed lines). Additional examples of trend estimation may be found in `MARSS` User Guide.

Identification of spatial population structure using model selection

In population surveys, censuses are often taken at multiple sites and those sites may or may not be connected through dispersal. Adjacent sites with high exchange rates of individuals will synchronize subpopulations (making them behave as a single subpopulation), while sites with low mixing rates will behave more independently as discrete subpopulations. We can use `MARSS` models and model selection to test hypotheses concerning the spatial structure of a population (Hinrichsen, 2009; Hinrichsen and Holmes, 2009; Ward et al., 2010).

For the multi-site application, \mathbf{x} is a vector of abundance in each of m subpopulations and \mathbf{y} is a vector of n observed time series associated with n sites. The number of underlying processes, m , is controlled by the user and not estimated. In the multi-site scenario, \mathbf{Z} , controls the assignment of survey sites to subpopulations. For instance, if the population is modeled as having four independent subpopulations and the first two were surveyed at one site each and the third and fourth were surveyed at two sites, then \mathbf{Z} is a 4×3 matrix with $(1, 0, 0, 0)$ in the first column, $(0, 1, 0, 0)$ in the second and $(0, 0, 1, 1)$ in the third. In the model argument for the `MARSS` function, we could specify this as `'Z = as.factor(c(1,2,3,3))'`. The \mathbf{Q} matrix controls the temporal correlation in the process errors for each subpopulation; these errors could be correlated or uncorrelated. The \mathbf{R} matrix controls the correlation structure of observation errors between the n survey sites.

The dataset `harborSeal` in the `MARSS` package contains survey data for harbor seals on the west coast of the United States. We can test the hypothesis that the four survey sites in Puget Sound (Washington state) are better described by one panmictic population measured at four sites versus four independent subpopulations.

```
> dat = t(log(harborSeal[,4:7]))
> harbor1 = MARSS(dat,
+   model = list(Z = factor(rep(1, 4))))
> harbor2 = MARSS(dat,
+   model = list(Z = factor(1:4)))
```

The default model settings of `'Q = "diagonal and unequal"', 'R = "diagonal and equal"',` and `'B = "identity"'` are used for this example. The AICc value for the one subpopulation surveyed at four sites is considerably smaller than the four subpopulations model. This suggests that these four sites are within one subpopulation (which is what one would expect given their proximity and lack of geographic barriers).

```
> harbor1$AICc
```

```
[1] -280.0486
```

```
> harbor2$AICc
```

```
[1] -254.8166
```

We can add a bootstrapped AIC using the function `MARSSaic`.

```
> harbor1 = MARSSaic(harbor1,
+   output = c("AICbp"))
```

We can add a confidence intervals to the estimated parameters using the function `MARSSparamCIs`.

```
> harbor1 = MARSSparamCIs(harbor1)
```

The MARSS default is to compute approximate confidence intervals using a numerically estimated Hessian matrix. Two full model selection exercises can be found in the MARSS User Guide.

Analysis of animal movement data

Another application of state-space modeling is identification of true movement paths of tagged animals from noisy satellite data. Using the MARSS package to analyze movement data assumes that the observations are equally spaced in time, or that enough NAs are included to make the observations spaced accordingly. The MARSS package includes the satellite tracks from eight tagged sea turtles, which can be used to demonstrate estimation of location from noisy data.

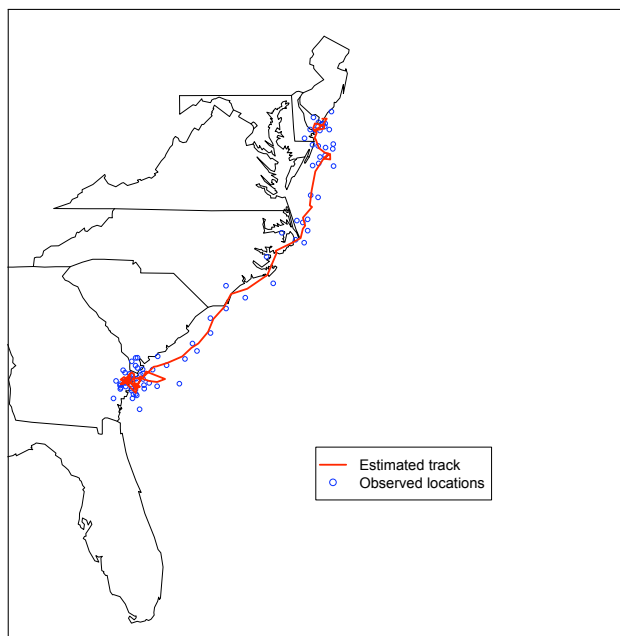


Figure 2: Estimated movement track for one turtle (“Big Mama”) using a two-dimensional state-space model. The MARSS User Guide shows how to produce this figure from the satellite data provided with the package.

The data consist of daily longitude and latitude for each turtle. The true location is the underlying

state \mathbf{x} , which is two-dimensional (latitude and longitude) and we have one observation time series for each state process. To model movement as a random walk with drift in two dimensions, we set \mathbf{B} equal to an identity matrix and constrain \mathbf{u} to be unequal to allow for non-isotropic movement. We constrain the process and observation variance-covariance matrices (\mathbf{Q} , \mathbf{R}) to be diagonal to specify that the errors in latitude and longitude are independent. Figure 2 shows the estimated locations for the turtle named “Big Mama”.

Estimation of species interactions from multi-species data

Ecologists are often interested in inferring species interactions from multi-species datasets. For example, Ives et al. (2003) analyzed four time series of predator (zooplankton) and prey (phytoplankton) abundance estimates collected weekly from a freshwater lake over seven years. The goal of their analysis was to estimate the per-capita effects of each species on every other species (predation, competition), as well as the per-capita effect of each species on itself (density dependence). These types of interactions can be modeled with a multivariate Gompertz model:

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t; \quad \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \quad (8)$$

The \mathbf{B} matrix is the interaction matrix and is the main parameter to be estimated. Here we show an example of estimating \mathbf{B} using one of the Ives et al. (2003) datasets. This is only for illustration; in a real analysis, it is critical to take into account the important environmental covariates, otherwise correlation driven by a covariate will affect the \mathbf{B} estimates. We show how to incorporate covariates in the MARSS User Guide.

The first step is to log-transform and de-mean the data:

```
> plank.dat = t(log(ivesDataByWeek[,c("Large Phyto",
+   "Small Phyto", "Daphnia", "Non-daphnia"))))
> d.plank.dat = (plank.dat - apply(plank.dat, 1,
+   mean, na.rm = TRUE))
```

Second, we specify the MARSS model structure. We assume that the process variances of the two phytoplankton species are the same and that the process variances for the two zooplankton are the same. But we assume that the abundance of each species is an independent process. Because the data have been de-meaned, the parameter \mathbf{u} is set to zero. Following Ives et al. (2003), we set the observation error to 0.04 for phytoplankton and to 0.16 for zooplankton. The model is specified as follows

```
> Z = factor(rownames(d.plank.dat))
> U = "zero"
> A = "zero"
> B = "unconstrained"
```



```
> Q = matrix(list(0), 4, 4)
> diag(Q) = c("Phyto", "Phyto", "Zoo", "Zoo")
> R = diag(c(0.04, 0.04, 0.16, 0.16))
> plank.model = list(Z = Z, U = U, Q = Q,
+                   R = R, B = B, A = A, tinitx=1)
```

We do not specify x_0 or v_0 since we assume the default behavior which is that the initial states are treated as estimated parameters with 0 variance.

We can fit this model using MARSS:

```
> kem.plank = MARSS(d.plank.dat,
+                  model = plank.model)

> B.est = matrix(kem.plank$par$B, 4, 4)
> rownames(B.est) = colnames(B.est) =
+   c("LP", "SP", "D", "ND")
> print(B.est, digits = 2)
```

	LP	SP	D	ND
LP	0.549	-0.23	0.096	0.085
SP	-0.058	0.52	0.059	0.014
D	0.045	0.27	0.619	0.455
ND	-0.097	0.44	-0.152	0.909

where "LP" and "SP" represent large and small phytoplankton, respectively, and "D" and "ND" represent the two zooplankton species, "Daphnia" and "Non-Daphnia", respectively.

Elements on the diagonal of \mathbf{B} indicate the strength of density dependence. Values near 1 indicate no density dependence; values near 0 indicate strong density dependence. Elements on the off-diagonal of \mathbf{B} represent the effect of species j on the per-capita growth rate of species i . For example, the second time series (small phytoplankton) appears to have relatively strong positive effects on both zooplankton time series. This is somewhat expected, as more food availability might lead to higher predator densities. The predators appear to have relatively little effect on their prey; again, this is not surprising as phytoplankton densities are often strongly driven by environmental covariates (temperature and pH) rather than predator densities. Our estimates of \mathbf{B} are different from those presented by Ives et al. (2003); in the MARSS User Guide we illustrate how to recover the estimates in Ives et al. (2003) by fixing elements to zero and including covariates.

Dynamic factor analysis

In the previous examples, each observed time series was representative of a single underlying state process, though multiple time series may be observations of the same state process. This is not a requirement for a MARSS model, however. Dynamic factor analysis (DFA) also uses MARSS models but treats each observed time series as a linear combination of multiple state processes. DFA has been applied in a number of fields (e.g. Harvey, 1989) and can be thought of as a principal components analysis for time-series data (Zuur et al., 2003a,b). In a DFA,

the objective is to estimate the number of underlying state processes, m , and the \mathbf{Z} matrix now represents how these state processes are linearly combined to explain the larger set of n observed time series. Model selection is used to select the size of m and the objective is to find the most parsimonious number of underlying trends (size of m) plus their weightings (\mathbf{Z} matrix) that explain the larger dataset.

To illustrate, we show results from a DFA analysis for six time series of plankton from Lake Washington (Hampton et al., 2006). Using model selection with models using $m = 1$ to $m = 6$, we find that the best (lowest AIC) model has four underlying trends, reduced from the original six. When $m = 4$, the DFA observation process then has the following form:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \\ y_{6,t} \end{bmatrix} = \begin{bmatrix} \gamma_{11} & 0 & 0 & 0 \\ \gamma_{21} & \gamma_{22} & 0 & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & \gamma_{44} \\ \gamma_{51} & \gamma_{52} & \gamma_{53} & \gamma_{54} \\ \gamma_{61} & \gamma_{62} & \gamma_{63} & \gamma_{64} \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \\ v_{6,t} \end{bmatrix}$$

Figure 3 shows the fitted data with the four state trajectories superimposed.

The benefit of reducing the data with DFA is that we can reduce a larger dataset into a smaller set of underlying trends and we can use factor analysis to identify whether some time series fall into clusters represented by similar trend weightings. A more detailed description of dynamic factor analysis, including the use of factor rotation to interpret the \mathbf{Z} matrix, is presented in the MARSS User Guide.

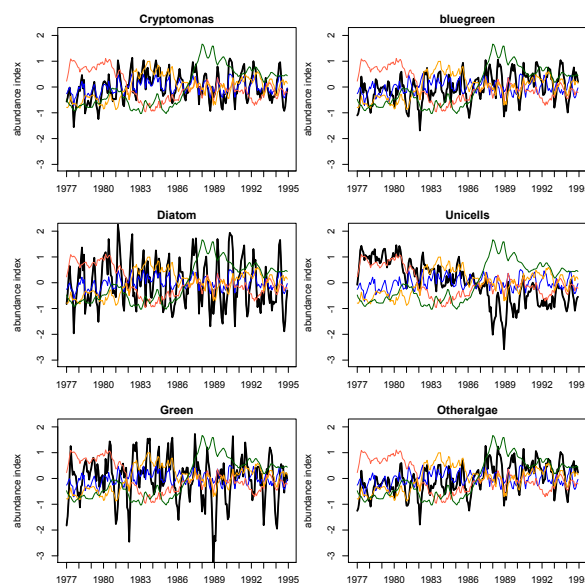


Figure 3: Results of a dynamic factor analysis applied to the Lake Washington plankton data. Colors represent estimated state trajectories and black lines represent the fitted values for each species.

Summary

We hope this package will provide scientists in a variety of disciplines some useful tools for the analysis of noisy univariate and multivariate time-series data, and tools to evaluate data support for different model structures for time-series observations. Future development will include Bayesian estimation and constructing non-linear time-series models.

Bibliography

- J. Cavanaugh and R. Shumway. A bootstrap variant of AIC for state-space model selection. *Statistica Sinica*, 7:473–496, 1997.
- B. Dennis and M. L. Taper. Density dependence in time series observations of natural populations: Estimation and testing. *Ecological Monographs*, 64(2):205–224, 1994.
- B. Dennis, P. L. Munholland, and J. M. Scott. Estimation of growth and extinction parameters for endangered species. *Ecological Monographs*, 61:115–143, 1991.
- C. Dethlefsen, S. Lundbye-Christensen, and A. L. Christensen. *sspir: state space models in R*, 2009. URL <http://CRAN.R-project.org/package=sspir>. R package version 0.2.8.
- J. Durbin and S. J. Koopman. *Time series analysis by state space methods*. Oxford University Press, Oxford, 2001.
- L. R. Gerber, D. P. D. Master, and P. M. Kareiva. Grey whales and the value of monitoring data in implementing the U.S. endangered species act. *Conservation Biology*, 13:1215–1219, 1999.
- P. D. Gilbert. *Brief user's guide: dynamic systems estimation*, 2009. URL <http://cran.r-project.org/web/packages/dse/vignettes/Guide.pdf>.
- S. E. Hampton, M. D. Scheuerell, and D. E. Schindler. Coalescence in the Lake Washington story: interaction strengths in a planktonic food web. *Limnology and Oceanography*, 51(5):2042–2051, 2006.
- A. C. Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge University Press, Cambridge, UK, 1989.
- J. Helske. *KFAS: Kalman filter and smoothers for exponential family state space models.*, 2011. URL <http://CRAN.R-project.org/package=KFAS>. R package version 0.6.1.
- R. Hinrichsen. Population viability analysis for several populations using multivariate state-space models. *Ecological Modelling*, 220(9-10):1197–1202, 2009.
- R. Hinrichsen and E. E. Holmes. Using multivariate state-space models to study spatial structure and dynamics. In R. S. Cantrell, C. Cosner, and S. Ruan, editors, *Spatial Ecology*. CRC/Chapman Hall, 2009.
- E. Holmes, E. Ward, K. Wills, NOAA, Seattle, and USA. *MARSS: multivariate autoregressive state-space modeling*, 2012. URL <http://CRAN.R-project.org/package=MARSS>. R package version 2.9.
- E. E. Holmes. Estimating risks in declining populations with poor data. *Proceedings of the National Academy of Sciences of the United States of America*, 98(9):5072–5077, 2001.
- E. E. Holmes. Derivation of the EM algorithm for constrained and unconstrained MARSS models. Technical report, Northwest Fisheries Science Center, Mathematical Biology Program, 2010.
- E. E. Holmes, J. L. Sabo, S. V. Viscido, and W. F. Fagan. A statistical approach to quasi-extinction forecasting. *Ecology Letters*, 10(12):1182–1198, 2007.
- J.-Y. Humbert, L. S. Mills, J. S. Horne, and B. Dennis. A better way to estimate population trends. *Oikos*, 118(12):1940–1946, 2009.
- A. R. Ives, B. Dennis, K. L. Cottingham, and S. R. Carpenter. Estimating community stability and ecological interactions from time-series data. *Ecological Monographs*, 73(2):301–330, 2003.
- D. Luethi, P. Erb, and S. Otziger. *FKF: fast Kalman filter*, 2012. URL <http://CRAN.R-project.org/package=FKF>. R package version 0.1.2.
- K. Metaxoglou and A. Smith. Maximum likelihood estimation of VARMA models using a state-space EM algorithm. *Journal of Time Series Analysis*, 28:666–685, 2007.
- G. Petris. An R package for dynamic linear models. *Journal of Statistical Software*, 36(12):1–16, 2010. URL <http://www.jstatsoft.org/v36/i12/>.
- J. Reddingius and P. den Boer. On the stabilization of animal numbers. Problems of testing. 1. Power estimates and observation errors. *Oecologia*, 78:1–8, 1989.
- J. T. Schnute. A general framework for developing sequential fisheries models. *Canadian Journal of Fisheries and Aquatic Sciences*, 51:1676–1688, 1994.
- R. H. Shumway and D. S. Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 3(4):253–264, 1982.
- D. S. Stoffer and K. D. Wall. Bootstrapping state-space models: Gaussian maximum likelihood estimation and the Kalman filter. *Journal of the American Statistical Association*, 86(416):1024–1033, 1991.

R. S. Tsay. *Analysis of financial time series*. Wiley Series in Probability and Statistics. Wiley, 2010.

E. J. Ward, H. Chirakkal, M. González-Suárez, D. Aurióles-Gamboa, E. E. Holmes, and L. Gerber. Inferring spatial structure from time-series data: using multivariate state-space models to detect metapopulation structure of California sea lions in the Gulf of California, Mexico. *Journal of Applied Ecology*, 1(47):47–56, 2010.

A. F. Zuur, R. J. Fryer, I. T. Jolliffe, R. Dekker, and J. J. Beukema. Estimating common trends in multivariate time series using dynamic factor analysis. *Environmetrics*, 14(7):665–685, 2003a.

A. F. Zuur, I. D. Tuck, and N. Bailey. Dynamic factor analysis to estimate common trends in fisheries time series. *Canadian Journal of Fisheries and Aquatic Sciences*, 60(5):542–552, 2003b.

Elizabeth E. Holmes
Northwest Fisheries Science Center
2725 Montlake Blvd E, Seattle WA 98112
USA
eli.holmes@noaa.gov

Eric J. Ward
Northwest Fisheries Science Center
2725 Montlake Blvd E, Seattle WA 98112
USA
eric.ward@noaa.gov

Kellie Wills
University of Washington
2725 Montlake Blvd E, Seattle WA 98112
USA
willsk@u.washington.edu

openair – Data Analysis Tools for the Air Quality Community

by Karl Ropkins and David C. Carslaw

Abstract The **openair** package contains data analysis tools for the air quality community. This paper provides an overview of data importers, main functions, and selected utilities and workhorse functions within the package and the function output class, as of package version 0.4-14. It is intended as an explanation of the rationale for the package and a technical description for those wishing to work more interactively with the main functions or develop additional functions to support ‘higher level’ use of **openair** and R.

Large volumes of air quality data are routinely collected for regulatory purposes, but few of those in local authorities and government bodies tasked with this responsibility have the time, expertise or funds to comprehensively analyse this potential resource (Chow and Watson, 2008). Furthermore, few of these institutions can routinely access the more powerful statistical methods typically required to make the most effective use of such data without a suite of often expensive and niche-application proprietary software products. This in turn places large cost and time burdens on both these institutions and others (e.g. academic or commercial) wishing to contribute to this work. In addition, such collaborative working practices can also become highly restricted and polarised if data analysis undertaken by one partner cannot be validated or replicated by another because they lack access to the same licensed products.

Being freely distributed under general licence, R has the obvious potential to act as a common platform for those routinely collecting and archiving data and the wider air quality community. This potential has already been proven in several other research areas, and commonly cited examples include the Bioconductor project (Gentleman et al, 2004) and the Epitools collaboration (<http://www.medepi.com/epitools>). However, what is perhaps most inspiring is the degree of transparency that has been demonstrated by the recent public analysis of climate change data in R and associated open debate (<http://chartsgraphs.wordpress.com/category/r-climate-data-analysis-tool/>). Anyone affected by a policy decision, could potentially have unlimited access to scrutinise both the tools and data used to shape that decision.

The openair rationale

With this potential in mind, the openair project was funded by UK NERC (award NE/G001081/1) specifically to develop data analysis tools for the wider air quality community in R as part of the NERC Knowledge Exchange programme (<http://www.nerc.ac.uk/using/introduction/>).

One potential issue was identified during the very earliest stages of the project that is perhaps worth emphasising for the existing R users.

Most R users already have several years of either formal or self-taught experience in statistical, mathematical or computational working practices before they first encounter R. They probably first discovered R because they were already researching a specific technique that they identified as beneficial to their research and saw a reference to a package or script in an expert journal or were recommended R by a colleague. Their first reaction on discovering R, and in particular the packages, was probably one of excitement. Since then they have most likely gone on to use numerous packages, selecting an appropriate combination for each new application they undertook.

Many in the air quality community, especially those associated with data collection and archiving, are likely to be coming to both **openair** (Carlsaw and Ropkins, 2010) and R with little or no previous experience of statistical programming. Like other R users, they recognise the importance of highly evolved statistical methods in making the most effective use of their data; but, for them, the step-change to working with R is significantly larger.

As a result many of the decisions made when developing and documenting the **openair** package were shaped by this issue.

Data structures and importers

Most of the main functions in **openair** operate on a single data frame. Although it is likely that in future this will be replaced with an object class to allow data unit handling, the data frame was initially adopted for two reasons. Firstly, air quality data is currently collected and archived in numerous formats and keeping the import requirements for **openair** simple minimised the frustrations associated with data importation. Secondly, restricting the user to working in a single data format greatly simplifies data management and working practices for those less familiar with programming environments.

Typically, the data frame should have named

fields, three of which are specifically reserved, namely: `date`, a field of 'POSIXt' class time stamps, and `ws` and `wd`, numeric fields for wind speed and wind direction data. There are no restrictions on the number of other fields and the names used outside the standard conventions of R. This means that the 'work up' to make a new file **openair**-compatible is minimal: Read in data; reformat and rename `date`; and rename wind speed and wind direction as `ws` and `wd`, if present.

That said, many users new to programming still found class structures, in particularly 'POSIXt', daunting. Therefore, a series of importer functions were developed to simplify this process.

The first to be developed was `import`, a general purpose function intended to handle comma and tab delimited file types. It defaults to a file browser (via `file.choose`), and is intended to be used in the common form, e.g.:

```
newdata <- import()
newdata <- import(file.type = "txt") #etc
```

(Here, as elsewhere in **openair**, argument options have been selected pragmatically for users with limited knowledge of file structures or programming conventions. Note that the `file.type` option is the file extension "txt" that many users are familiar with, rather than either the `delim` from `read.delim` or the "\t" separator.)

A wide range of monitoring, data logging and archiving systems are used by the air quality community and many of these employ unique file layouts, including e.g. multi-column date and stamps, isolated or multi-row headers, or additional information of different dimensions to the main data set. So, `import` includes a number of arguments, described in detail in `?import`, that can be used to fine-tune its operation for novel file layouts.

Dedicated importers have since been written for some of the file formats and data sources most commonly used by the air quality community in the UK. These operate in the common form:

```
newdata <- import[Name]()
```

And include:

- `importADMS`, a general importer for 'bgd', 'met', 'mop' and 'pst' file formats used by the Atmospheric Dispersion Modelling System (McHugh et al, 1997) developed by CERC (<http://www.cerc.co.uk/index.php>). ADMS is widely used in various forms to model both current and future air quality scenarios (<http://www.cerc.co.uk/environmental-software/ADMS-model.html>).
- `importAURN` and `importAURNcsv`, importers for hourly data from the UK (Automatic Urban and Rural Network) Air Quality Data Archive

(http://www.airquality.co.uk/data_and_statistics.php). `importAURN` provides a direct link to the archive and downloads data directly from the online archive. `importAURNcsv` allows '.csv' files previously downloaded from the archive to be read into **openair**.

- `importKCL`, an importer for direct online access to data from the King's College London databases (<http://www.londonair.org.uk/>).

Here, we gratefully acknowledge the very significant help and support of AEAT, King College London's Environmental Research Group (ERG) and CERC in the development of these importers. AEAT and ERG operate the AURN and LondonAir archives, respectively, and both specifically set up dedicated services to allow the direct download of '.RData' files from their archives. CERC provided extensive access to multiple generations of ADMS file structures and ran an extensive programme of compatibility testing to ensure the widest possible body of ADMS data was accessible to **openair** users.

Example data

The **openair** package includes one example dataset, `mydata`. This is data frame of hourly measurements of air pollutant concentrations, wind speed and wind direction collected at the Marylebone (London) air quality monitoring supersite between 1st January 1998 and 23rd June 2005 (source: London Air Quality Archive; <http://www.londonair.org.uk>).

The same dataset is available to download as a '.csv' file from the **openair** website (http://www.openair-project.org/CSV/OpenAir_example_data_long.csv). This file can be directly loaded into **openair** using the `import` function. As a result, many users, especially those new to R, have found it a very useful template when loading their own data.

Manuals

Two manuals are available for use with **openair**. The standard R manual is available alongside the package at its 'CRAN' repository site. An extended manual, intended to provide new users less familiar with either R or **openair** with a gentler introduction, is available on the **openair** website: <http://www.openair-project.org>.

Main functions

Most of the main functions within **openair** share a highly similar structure and, wherever possible, common arguments. Many in the air quality community are very familiar with 'GUI' interfaces and data

analysis procedures that are very much predefined by the software developers. R allows users the opportunity to really explore their data. However, a command line framework can sometimes feel frustratingly slow and awkward to users more used to a ‘click and go’ style of working. Standardising the argument structure of the main functions both encourages a more interactive approach to data analysis and minimises the amount of typing required of users more used to working with a mouse than keyboard.

Common `openair` function arguments include: `pollutant`, which identifies the data frame field or fields to select data from; `statistic`, which, where data are grouped, e.g. share common coordinates on a plot, identifies the summary statistic to apply if only a single value is required; and, `avg.time`, which, where data series are to be averaged on longer time periods, identifies the required time resolution. However, perhaps the most important of these common arguments is `type`, a simplified form of the conditioning term `cond` widely used elsewhere in R.

Rapid data conditioning is only one of a large number of benefits that R provides, but it is probably the one that has most resonance with air quality data handlers. Most can instantly appreciate its potential power as a data visualisation tool and its flexibility when used in a programming environment like R. However, many new users can struggle with the fine control of `cond`, particularly with regards to the application of `format.POSIX*` to time stamps. The `type` argument therefore uses an `openair` workhorse function called `cutData`, which is discussed further below, to provide a robust means of conditioning data using options such as "hour", "weekday", "month" and "year")

These features are perhaps best illustrated with an example.

The `openair` function `trendLevel` is basically a wrapper for the `lattice` (Sarkar, 2009) function `levelplot` that incorporates a number of built-in conditioning and data handling options based on these common arguments. So, many users will be very familiar with the basic implementation.

The function generates a `levelplot` of `pollutant` $\sim x * y | type$ where `x`, `y` and `type` are all cut/factorised by `cutData` and in each `x/y/type` case `pollutant` data is summarised using the option `statistic`.

When applied to the `openair` example dataset `mydata` in its default form, `trendLevel` uses `x = "month"` (month of year), `y = "hour"` (time of day) and `type = "year"` to provide information on trends, seasonal effects and diurnal variations in mean NO_x concentrations in a single output (Figure 1).

However, `x`, `y`, `type` and `statistic` can all be user-defined.

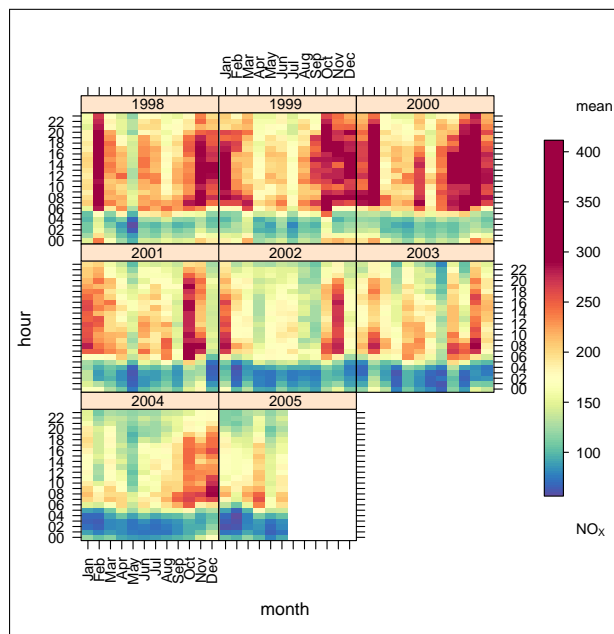


Figure 1: `openair` plot `trendLevel(mydata, "nox")`. Note: The seasonal and diurnal trends, high in winter months, and daytime hours, most notably early morning and evening, are very typical of man-made sources such as traffic and the general, by-panel, decrease in mean concentrations reflects the effect of incremental air quality management regulations introduced during the last decade.

The function arguments `x`, `y` and `type` can be set to a wide range of time/date options or to any of the fields within the supplied data frame, with numerics being cut into quantiles, characters converted to factors, and factors used as is.

Similarly `statistic` can also be either a pre-coded option, e.g. "mean", "max", etc, or be a user defined function. This ‘tiered approach’ provides both simple, robust access for new users and a very flexible structure for more experienced R users. To illustrate this point, the default `trendLevel` plot (Figure 1) can be generated using three equivalent calls:

```
# predefined
trendLevel(mydata, statistic = "mean")

# using base::mean
trendLevel(mydata, statistic = mean)

# using local function
my.mean <- function(x) {
  x <- na.omit(x)
  sum(x) / length(x)}
trendLevel(mydata, pollutant = "nox",
  statistic = my.mean)
```

The `type` argument can accept one or two options, depending on function, and in the latter case strip labelling is handled using the `latticeExtra` (Sarkar and Andrews, 2011) function `useOuterStrips`.

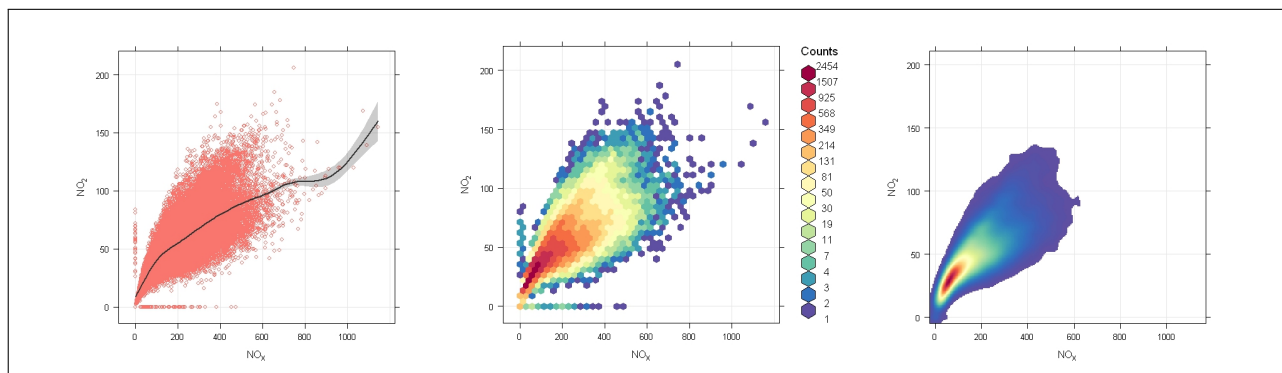


Figure 2: **openair** plots generated using `scatterPlot(mydata, "nox", "no2", ...)` and `method = "scatter"` (default; left), `"hexbin"` (middle) and `"density"` (right).

The other main functions include:

- `summaryPlot`, a function which generates a rug plot and histogram view of one or more data frame fields, as well as calculating several key statistics that can be used to characterise the quality and capture efficiency of data collecting in extended monitoring programmes. The plot provides a useful screening prior to the main data analysis.
- `timePlot` and `scatterPlot`, time-series and traditional scatter plot functions. These were originally introduced to provide such plots in a format consistent with other **openair** outputs, but have evolved through user-feedback to include additional hard-coded options that may be of more general use. Perhaps, the most obvious of these being the `"hexbin"` (hexagonal binning using the `hexbin` package (Carr et al, 2011)) and `"density"` (2D kernel density estimates using `.smoothScatterCalcDensity` in `grDevices`) methods for handling over-plotting (Figure 2).
- Trend analysis is an important component of air quality management, both in terms of historical context and abatement strategy evaluation. **openair** includes three trend analysis functions: `MannKendall`, `smoothTrend` and `LinearRelation`. `MannKendall` uses methods based on Hirsch et al (1982) and Helsel and Hirsch (2002) to evaluate monotonic trends. Sen-Theil slope and uncertainty are estimated using code based on that published on-line by Rand Wilcox (<http://www-rcf.usc.edu/~rwilcox/>) and the basic method has been extended to block bootstrap simulation to account for auto-correlation (Kunsch, 1989). `smoothTrend` fits a generalized additive model (GAM) to monthly averaged data to provide a non-parametric description of trends using methods and functions in the package `mgcv` (Wood, 2004, 2006). Both functions incorporate an option to deseasonalise data prior

to analysis using the `stl` function in **stats** (Cleveland et al, 1990). The other function, `linearRelation`, uses a rolling window linear regression method to visualise the degree of change in the relations between two species over larger timescales.

- `windRose` generates a traditional 'wind rose' style plot of wind speed and direction. The associated wrapper function `pollutionRose` allows the user to substitute wind speed with another data frame field, most commonly a pollutant concentration time-series, to produce 'pollution roses' similar to those used by Henry et al (2009).
- `polarFreq`, `polarPlot` and `polarAnnulus` are a family of functions that extend polar visualisations. In its default form `polarFreq` provides an alternative to wind speed/direction description to `windRose`, but pollutant and statistic arguments can also be included in the call to produce a wide range of other polar data summaries. `polarPlot` uses `mgcv::gam` to fit a surface to data in the form `polarFreq(..., statistic = "mean")` to provide a useful visualisation tool for pollutant source characterisation. This point is illustrated by Figure 3 which shows three related plots. Figure 3 left is a basic polar presentation of mean NO_x concentrations produced using `polarFreq(mydata, "nox", statistic = "mean")`. Figure 3 middle is a comparable `polarPlot`, which although strictly less quantitatively accurate, greatly simplifies the identifications of the main features, namely a broad high concentration feature to the Southwest with a maxima at lower wind speeds (indicating a local source) and a lower concentration but more resolved high wind speed feature to the East (indicating a more distant source). Then, finally Figure 3 right presents a similar `polarPlot` of SO₂, which demonstrates that the local source (most likely near-by traffic) is rel-

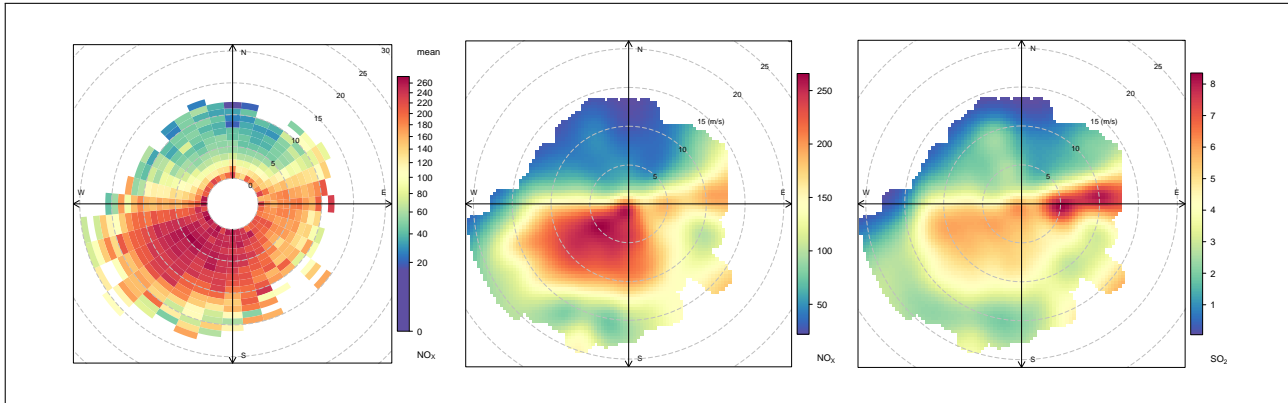


Figure 3: **openair** plots generated using (left) `polarFreq(mydata, "nox", statistic = "mean")`; (middle) `polarPlot(mydata, "nox")`; and, (right) `polarPlot(mydata, "so2")`.

atively NO_x rich while the more distant Easterly feature (most likely power station emissions) is relatively SO_2 rich. This theme is discussed in further detail in Carslaw et al (2006). The `polarAnnulus` function provides an alternative polar visualisation based on wind direction and time frequency (e.g. hour of day, day of year, etc.) to explore similar diagnostics to those discussed above with regard to `trendLevel`.

- Likewise, `timeVariation` generates a range of hour-of-the-day and day-of-the-week and month-of-the-year plots that can provide useful insights regarding the time frequency of one or more pollutants.
- `calendarPlot` presents daily data in a conventional calendar-style layout. This is a highly effective format for the presentation of information, especially when working with non-experts.
- Air quality standards are typically defined in terms of upper limits that concentrations of a particular pollutant may not exceed or may only exceed a number of times in any year. `kernelExceed` uses a kernel density function (`.smoothScatterCalcDensity` in **grDevices**) to map the distribution of such exceedances relative to two other parameters. The function was developed for use with daily mean (European) limit value for PM_{10} (airborne particulate matter up to 10 micrometers in size) of $50 \mu g/m^3$ not to be exceeded on more than 35 days, and in its default form plots PM_{10} exceedances relative to wind speed and direction.
- **openair** also includes a number of specialist functions. The `calcFno2` function provides an estimate of primary NO_2 emissions ratios, a question of particular concern for the air quality community at the moment. Associated theory is provided in Carslaw

and Beevers (2005), and further details of the function's use are given in the extended version of the **openair** manual (<http://www.openair-project.org>). Functions `modStats` and `conditionalQuantile` were developed for model evaluation.

Utilities and workhorse functions

The **openair** package includes a number of utilities and workhorse functions that can be directly accessed by users and therefore used more generally. These include:

- `cutData`, a workhorse function for data conditioning, intended for use with the `type` option in main **openair** functions. It accepts a data frame and returns the conditioned form:

```
head(olddata)
      date    ws   wd nox
1 1998-01-01 00:00:00 0.60 280 285
2 1998-01-01 01:00:00 2.16 230  NA
3 1998-01-01 02:00:00 2.76 190  NA
4 1998-01-01 03:00:00 2.16 170 493
5 1998-01-01 04:00:00 2.40 180 468
6 1998-01-01 05:00:00 3.00 190 264

newdata <- cutData(olddata,
                  type = "hour")
head(newdata)
      date    ws   wd nox hour
1 1998-01-01 00:00:00 0.60 280 285  00
2 1998-01-01 01:00:00 2.16 230  NA  01
3 1998-01-01 02:00:00 2.76 190  NA  02
4 1998-01-01 03:00:00 2.16 170 493  03
5 1998-01-01 04:00:00 2.40 180 468  04
6 1998-01-01 05:00:00 3.00 190 264  05
```


Here `type` can be the name of one of the fields in the data frame or one of a number of pre-defined terms. Data frame fields are handled pragmatically, e.g. factors are returned unmodified, characters are converted to factors, numerics are subset by `stats::quantile`. By default numerics are converted into four quantile bands but this can be modified using the additional option `n.levels`. Pre-defined terms include "hour" (hour-of-day), "daylight" (daylight or nighttime), "weekday" (day-of-week), "weekend" (weekday or weekend), "month" (month-of-year), "monthyear" (month and year), "season" (season-of-year), "gmtbst" (GMT or BST) and "wd" (wind direction sector).

With the exception of "daylight", "season", "gmtbst" and "wd" these are wrappers for conventional `format.POSIXt` operations commonly required by **openair** users.

"daylight" conditions the data relative to estimated sunrise and sunset to give either daylight or nighttime. The cut is actually made by a specialist function, `cutDaylight`, but more conveniently accessed via `cutData` or the main functions. The 'daylight' estimation, which is valid for dates between 1901 and 2099, is made using the measurement date, time, location and astronomical algorithms to estimate the relative positions of the Sun and the measurement location on the Earth's surface, and is based on NOAA methods (<http://www.esrl.noaa.gov/gmd/grad/solcalc/>). Date and time are extracted from the `date` field but can be modified using the additional option `local.hour.offset`, and location is determined by `latitude` and `longitude` which should be supplied as additional options.

"season" conditions the data by month-of-year (as "month") and then regroups data into the four seasons. By default, the operation assumes the measurement was made in the northern hemisphere, and winter = December, January and February, spring = March, April and May, etc., but can be reset using the additional option `hemisphere` (`hemisphere = "southern"` which returns winter = June, July and August, spring = September, October and November, etc.). Note: for convenience/traceability these are uniquely labelled, i.e. northern hemisphere: winter (DJF), spring (MAM), summer (JJA), autumn (SON); southern hemisphere: winter (JJA), spring (SON), summer (DJF), autumn (MAM).

"gmtbst" (and the alternative form "bstgmt") conditions the data according to daylight saving. The operation returns two cases: GMT or BST for measurement date/times where

daylight saving was or was not enforced (or more directly GMT and BST time stamps are or are not equivalent), respectively, and resets the `date` field to local time (BST). Man-made sources, such as NO_x emissions from vehicles in urban areas where daylight saving is enforced will tend to be associated with local time. So, for example, higher 'rush-hour' concentrations will tend to be associated with BST time stamps (and remain relatively unaffected by the BST/GMT case). By contrast a variable such as wind speed or temperature that is not as directly influenced by daylight saving should show a clear BST/GMT shift when expressed in local time. Therefore, when used with an **openair** function such as `timeVariation`, this type conditioning can help determine whether variations in pollutant concentrations are driven by man-made emissions or natural processes.

"wd" conditions the data by the conventional eight wind sectors, N (0-22.5° and 337.5-360°), NE (22.5-67.5°), E (67.5-112.5°), SE (112.5-157.5°), S (157.5-202.5°), SW (202.5-247.5°), W (247.5-292.5°) and NW (292.5-337.5°).

- `selectByDate` and `splitByDate` are functions for conditioning and subsetting a data frame using a range of `format.POSIXt` operations and options similar to `cutData`. These are mainly intended as a convenient alternative for newer **openair** users.
- `drawOpenKey` generates a range of colour keys used by other **openair** functions. It is a modification of the `lattice::draw.colorkey` function and here we gratefully acknowledge the help and support of Deepayan Sarkar in providing both `draw.colorkey` and significant advice on its use and modification. More widely used colour key operations can be accessed from main **openair** functions using standard options, e.g. `key.position` (= "right", "left", "top", or "bottom"), and `key.header` and `key.footer` (text to be added as headers and footers, respectively, on the colour key). In addition, finer control can be obtained using the option `key` which should be a list. `key` is similar to `key` in `lattice::draw.colorkey` but allows the additional components: `header` (a character vector of text or list including header text and formatting options for text to be added above the colour key), `footer` (as header but for below the colour key), `auto.text` (TRUE/FALSE for using **openair** workhorse `quickText`), and `tweaks`, `fit`, `slot` (a range of options to control the relative positions of header, footer and the colour key) and `plot.style` (a list of options controlling the type of colour key produced). One simple example of the use of

`drawOpenKey(key(plot.style))` is the paddle scale in `windRose` - compare `windRose(mydata)` and `windRose(mydata,paddle = FALSE)`.

`drawOpenKey` can be used with other **lattice** outputs using methods previously described by Deepayan Sarkar (Sarkar, 2008, 2009), e.g.:

```
## some silly data and colour scale
range <- 1:200; cols <- rainbow(200)

## my.key -- an openair plot key
my.key <- list(col = cols, at = range,
             space = "right",
             header = "my header",
             footer = "my footer")

## pass to lattice::xyplot
xyplot(range ~ range,
       cex = range/10, col = cols,
       legend = list(right =
                     list(fun = drawOpenKey,
                          args = list(key = my.key)
                     )))
```

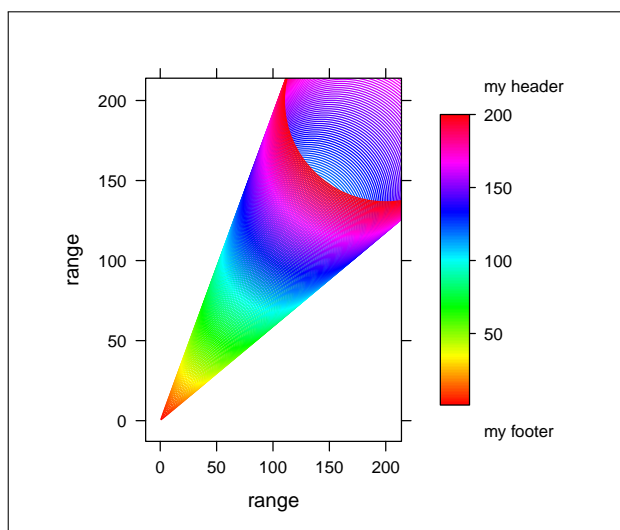


Figure 4: Trivial example of the use of `openColourKey` with a **lattice** plot.

- `openColours` is a workhorse function that produces a range of colour gradients for other **openair** plots. The main option is `scheme` and this can be either the name of a pre-defined colour scheme, e.g. "increment", "default", "brewer1", "heat", "jet", "hue", "greyscale", or two or more terms that can be coerced into valid colours and between which a colour gradient can be extrapolated, e.g. `c("red", "blue")`. In most **openair** plot functions `openColours(scheme)` can be accessed using the common option `cols`. Most of the colour gradient operations are based on standard R and **RColorBrewer** (Neuwirth, 2011) methods.

The "greyscale" scheme is a special case intended for those producing figures for use in black and white reports that also automatically resets some other **openair** plot parameters (e.g. strip backgrounds to white and other preset text, point and line to black or a 'best guess' grey).

- `quickText` is a look-up style wrapper function that applies routine text formatting to expressions widely used in the air quality community, e.g. the super- and sub-scripting of chemical names and measurement units. The function accepts a 'character' class object and returns it as an 'expression' with any recognised text reformatted according to common convention. Labels in **openair** plots (`xlab`, `ylab`, `main`, etc) are typically passed via `quickText`. This, for examples, handles the automatic formatting of the colour key and axes labels in Figures 1–3, where the inputs were data frame field names, "nox", "no2", etc (most convenient for command line entry) and the conventional chemical naming convention (International Union of Pure and Applied Chemistry, IUPAC, nomenclature) were NO_x , NO_2 , etc.

`quickText` can also be used as a label wrapper with non-**openair** plots, e.g.:

```
my.lab <- "Particulates as pm10, ug/m3"
plot(pm10 ~ date, data = mydata[1:1000,],
     ylab = quickText(my.lab))
```

(While many of us regard 'expressions' as trivial, such label formatting can be particularly confusing for those new both programming languages and command line work, and functions like `quickText` really help those users to focus on the data rather than becoming frustrated with the periphery.)

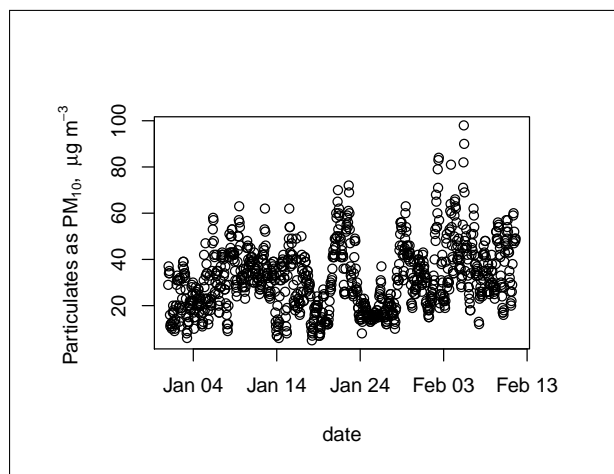


Figure 5: Trivial example of the use of `quickText` outside **openair**.

- `timeAverage` is a function for the aggregation of **openair** data frames using the 'POSIXt' field "date". By default it calculates the day average using an operation analogous to `mean(..., na.rm = TRUE)`, but additional options also allow it to be used to calculate a wide range of other statistics. The aggregation interval can be set using option `avg.time`, e.g. "hour", "2 hour", etc (cut.POSIXt conventions). The statistic can be selected from a range of pre-defined terms, including "mean", "max", "min", "median", "sum", "frequency" (using `length`), "sd" and "percentile". If `statistic = "percentile"`, the default 95 (%) may be modified using the additional option `percentile`. The data capture threshold can be set using `data.thresh` which defines the percentage of valid cases required in any given (aggregation) period where the statistics are to be calculated.

While for the most part `timeAverage` could be considered merely a convenient wrapper for a number of more complex 'POSIXt' aggregation operations, one important point that should be emphasised is that it handles the polar measure wind direction correctly. Assuming wind direction and speed are supplied as the data frame fields "wd" and "ws", respectively, these are converted to wind vectors and the average wind direction is calculated using these. If this were not done and wind direction averages were calculated from "wd" alone then measurements about North (e.g. 355–360° and 0–5°) would average at about 180° not 0° or 360°.

Output class

Many of the main functions in **openair** return an object of "openair" class, e.g.:

```
#From:
[object] <- openair.function(...)

#object structure
[object] #list[S3 "openair"]
  $call [function call]
  $data [data.frame generated/used in plot]
        [or list if multiple part]
  $plot [plot from function]
        [or list if multiple part]

#Example
ans <- windRose(mydata)
ans

openair object created by:
  windRose(mydata = mydata)
```

this contains:

```
a single data frame:
  $data [with no subset structure]
a single plot object:
  $plot [with no subset structure]
```

Associated generic methods (`head`, `names`, `plot`, `print`, `results`, `summary`) have a common structure:

```
#method structure for openair generics
[generic method].[class] #method.name
([object], [class-specific options],
[method-specific options]) #options
```

As would be expected, most `.openair` methods work like associated `.default` methods, and object and method-specific options are based on those of the `.default` method. Typically, **openair** methods return outputs consistent with the associated `.default` method unless either `$data` or `$plot` have multiple parts in which cases outputs are returned as lists of data frames or plots, respectively. The main class-specific option is `subset`, which can be used to select specific sub-data or sub-plots if these are available. The local method `results` extracts the data frames generated during the plotting process. Figure 6 shows some trivial examples of current usage.

Conclusions and future directions

As with many other R packages, the feedback process associated with users and developers working in an open-source environment means that **openair** is subject to continual optimisation. As a result, **openair** will undoubtedly continue to evolve further through future versions. Obviously, the primary focus of **openair** will remain the development of tools to help the air quality community make better use of their data. However, as part of this work we recognise that there is still work to be done.

One area that is likely to undergo significant updates is the use of classes and methods. The current 'S3' implementation of output class is crude, and future options currently under consideration include improvements to `plot.openair` and `print.openair`, the addition of an `update.openair` method (for reworking **openair** plots), the release of the `openairApply` (a currently un-exported wrapper function for apply-type operations with **openair** objects), and the migration of the object to 'S4'.

In light of the progress made with the output class, we are also considering the possibility of replacing the current simple data frame input with a dedicated class structure, as this could provide access to extended capabilities such as measurement unit management.

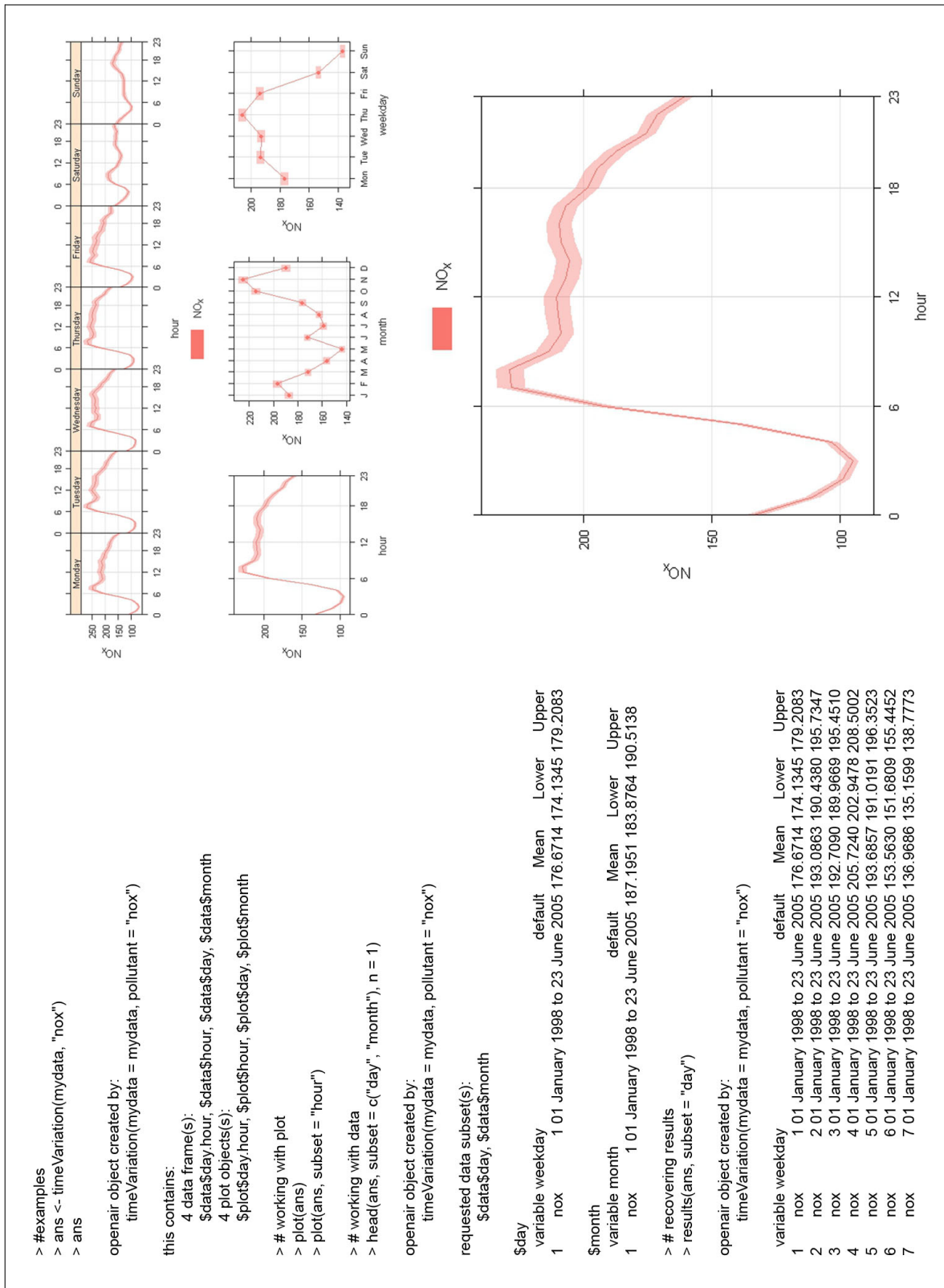


Figure 6: Trivial examples of "openair" object handling, with the outputs of `plot(ans)` and `plot(ans, subset = "hour")` shown as inserts right top and right bottom, respectively.

One other particularly exciting component of recent work on **openair** is international compatibility. The initial focus of the **openair** package was very much on the air quality community in the UK. However, distribution of the package through CRAN has meant that we now have an international user group with members in Europe, the United States, Canada, Australia and New Zealand. This is obviously great. However, it has also brought with it several challenges, most notably in association with local time stamp and language formats. Here, we greatly acknowledge the help of numerous users and colleagues who bug-tested and provided feedback as part of our work to make **openair** less 'UK-centric'. We will continue to work on this aspect of **openair**.

We also greatly acknowledge those in our current user group who were less familiar with programming languages and command lines but who took a real 'leap of faith' in adopting both R and **openair**. We will continue to work to minimise the severity of the learning curves associated with both the uptake of **openair** and the subsequent move from using **openair** in a standalone fashion to its much more efficient use as part of R.

Bibliography

- D. Carr, N. Lewin-Koh and M. Maechler. **hexbin**: Hexagonal Binning Routines. R package version 1.26.0. URL <http://CRAN.R-project.org/package=hexbin>.
- D.C. Carslaw and S.D. Beevers. Estimations of road vehicle primary NO₂ exhaust emission fractions using monitoring data in London. *Atmospheric Environment*, 39(1):167–177, 2005.
- D.C. Carslaw, S.D. Beevers, K. Ropkins and M.C. Bell. Detecting and quantifying aircraft and other on-airport contributions to ambient nitrogen oxides in the vicinity of a large international airport. *Atmospheric Environment*, 40(28):5424–5434, 2006.
- D.C. Carslaw and K. Ropkins. **openair**: Open-source tools for the analysis of air pollution data. R package version 0.4-14. URL <http://www.openair-project.org/>.
- J.C. Chow and J.G. Watson. New Directions: Beyond compliance air quality measurements. *Atmospheric Environment*, 42:5166–5168, 2008.
- R.B. Cleveland, W.S. Cleveland, J.E. McRae and I. Terpenning, I. STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics*, 6:3–73, 1990.
- R. Gentleman, V.J. Carey, D.M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A.J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J.Y.H. Yang and J. Zhang. **Bioconductor**: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. URL <http://www.bioconductor.org/>.
- D. Helsel and R. Hirsch. **R**: Statistical methods in water resources. US Geological Survey. URL <http://pubs.usgs.gov/twri/twri4a3/>.
- R. Henry, G.A. Norris, R. Vedantham and J.R. Turner. Source Region Identification Using Kernel Smoothing. *Environmental Science and Technology*, 43(11):4090–4097, 2009.
- R.M. Hirsch, J.R. Slack, and R.A. Smith. Techniques of trend analysis for monthly water-quality data. *Water Resources Research*, 18(1):107–121, 1982.
- H.R. Kunsch. The jackknife and the bootstrap for general stationary observations. *Annals of Statistics*, 17(3):1217–1241, 1989.
- C.A. McHugh, D.J. Carruthers and H.A. Edmunds. ADMS and ADMS570 Urban. *International Journal of Environment and Pollution*, 8(3–6):438–440, 1997.
- E. Neuwirth. **RColorBrewer**: ColorBrewer palettes. R package version 1.0-5. URL <http://CRAN.R-project.org/package=RColorBrewer/>.
- D. Sarkar. **Lattice**: Multivariate Data Visualization with R. Springer. ISBN: 978-0-387-75968-5. URL <http://lmdvr.r-forge.r-project.org/>.
- D. Sarkar. **lattice**: Lattice Graphics. R package version 0.18-5. URL <http://r-forge.r-project.org/projects/lattice/>.
- D. Sarkar and F. Andrews. **latticeExtra**: Extra Graphical Utilities Based on Lattice. R package version 0.6-18. URL <http://CRAN.R-project.org/package=latticeExtra>.
- S.N. Wood. Stable and efficient multiple smoothing parameter estimation for generalized additive models. *Journal of the American Statistical Association*, 99:673–686, 2004.
- S.N. Wood. **Generalized Additive Models: An Introduction with R**. Chapman and Hall/CRC, 2006.

Karl Ropkins
Institute for Transport Studies
University of Leeds, LS2 9JT, UK
k.ropkins@its.leeds.ac.uk

David C. Carslaw
King's College London
Environmental Research Group
Franklin Wilkins Building, 150 Stamford Street
London SE1 9NH, UK
david.carslaw@kcl.ac.uk

Foreign Library Interface

by Daniel Adler

Abstract We present an improved Foreign Function Interface (FFI) for R to call arbitrary native functions without the need for C wrapper code. Further we discuss a dynamic linkage framework for binding standard C libraries to R across platforms using a universal type information format. The package `rdyncall` comprises the framework and an initial repository of cross-platform bindings for standard libraries such as (legacy and modern) *OpenGL*, the family of *SDL* libraries and *Expat*. The package enables system-level programming using the R language; sample applications are given in the article. We outline the underlying automation tool-chain that extracts cross-platform bindings from C headers, making the repository extendable and open for library developers.

Introduction

We present an improved Foreign Function Interface (FFI) for R that significantly reduces the amount of C wrapper code needed to interface with C. We also introduce a *dynamic* linkage that binds the C interface of a pre-compiled library (*as a whole*) to an interpreted programming environment such as R - hence the name *Foreign Library Interface*. Table 1 gives a list of the C libraries currently supported across major R platforms. For each library supported, abstract interface specifications are declared in a compact platform-neutral text-based format stored in so-called *DynPort* file on a local repository.

R was chosen as the first language to implement a proof-of-concept implementation for this approach. This article describes the `rdyncall` package which implements a toolkit of low-level facilities that can be used as an alternative FFI to interface with C. It also facilitates direct and convenient access to common C libraries from R without compilation.

The project was motivated by the fact that high-quality software solutions implemented in portable C are often not available in interpreter-based languages such as R. The pool of freely available C libraries is quite large and represents an invaluable resource for software development. For example, *OpenGL* (OpenGL Architecture Review Board et al., 2005) is the most portable and standard interface to accelerated graphics hardware for developing real-time graphics software. The combination of *OpenGL* with the *Simple DirectMedia Layer* (SDL) (Lantinga, 2009) core and extension libraries offers a foundation framework for developing interactive multime-

dia applications that can run on a multitude of platforms.

Foreign function interfaces

FFIs provide the backbone of a language to interface with foreign code. Depending on the design of this service, it can largely unburden developers from writing additional wrapper code. In this section, we compare the built-in R FFI with that provided by `rdyncall`. We use a simple example that sketches the different work flow paths for making an R binding to a function from a foreign C library.

FFI of base R

Suppose that we wish to invoke the C function `sqrt` of the *Standard C Math* library. The function is declared as follows in C:

```
double sqrt(double x);
```

The `.C` function from the base R FFI offers a call gate to C code with very strict conversion rules, and strong limitations regarding argument- and return-types: R arguments are passed as C pointers and C return types are not supported, so only C void functions, which are procedures, can be called. Given these limitations, we are not able to invoke the foreign `sqrt` function directly; intermediate wrapper code written in C is needed:

```
#include <math.h>
void R_C_sqrt(double * ptr_to_x)
{
  double x = ptr_to_x[0], ans;
  ans = sqrt(x);
  ptr_to_x[0] = ans;
}
```

We assume that the wrapper code is deployed as a shared library in a package named `testsqrt` which links to the *Standard C Math* library¹. Then we load the `testsqrt` package and call the C wrapper function directly via `.C`.

```
> library(testsqrt)
> .C("R_C_sqrt", 144, PACKAGE="testsqrt")
[[1]]
[1] 12
```

To make `sqrt` available as a public function, an additional R wrapper layer is needed to carry out type-safety checks:

¹We omit here the details such as registering C functions which is described in detail in the R Manual 'Writing R Extensions' (R Development Core Team, 2010).

Lib/DynPort	Description	Functions	Constants	Struct/Union
GL	OpenGL	336	3254	-
GLU	OpenGL Utility	59	155	-
R	R library	238	700	27
SDL	Audio/Video/UI abstraction	201	416	34
SDL_image	Pixel format loaders	35	3	-
SDL_mixer	Music format loaders and playing	71	27	-
SDL_net	Network programming	34	5	3
SDL_ttf	Font format loaders	38	7	-
cuda	GPU programming	387	665	84
expat	XML parsing framework	65	70	-
glew	GL extensions	1857	-	-
gl3	OpenGL 3 (strict)	317	838	1
ode	Rigid Physics Engine	547	109	11
opencl	GPU programming	79	263	10
stdio	Standard I/O	75	3	-

Table 1: Overview of available DynPorts for portable C Libraries

```
sqrtViaC <- function(x)
{
  x <- as.numeric(x) # type(x) should be C double.
  # make sure length > 0:
  length(x) <- max(1, length(x))
  .C("R_C_sqrt", x, PACKAGE="example")[[1]]
}
```

We can conclude that – in realistic settings – the built-in FFI of R almost always needs support by a wrapper layer written in C. The "foreign" in the FFI of **base** is in fact relegated to the C wrapper layer.

FFI of `rdyncall`

`rdyncall` provides an alternative FFI for R that is accessible via the function `.dyncall`. In contrast to the base R FFI, which uses a C wrapper layer, the `sqrt` function is invoked dynamically and directly by the interpreter at run-time. Whereas the *Standard C Math* library was loaded implicitly via the `testsqrt` package, it now has to be loaded explicitly.

R offers functions to deal with shared libraries at run-time, but the location has to be specified as an absolute file path, which is platform-specific. A platform-portable solution is discussed in a following section on *Portable loading of shared library*. For now, we assume that the example is done on Mac OS X where the *Standard C Math* library has the file path `'usr/lib/libm.dylib'`:

```
> libm <- dyn.load("/usr/lib/libm.dylib")
> sqrtAddr <- libm$sqrt$address
```

We first need to load the R package `rdyncall`:

```
> library(rdyncall)
```

²The maximum number of arguments is limited by the amount of memory required for prebuffering a single call. It is currently fixed to 4 kilobyte (approx. 512-1024 arguments).

Finally, we invoke the foreign C function `sqrt` *directly* via `.dyncall`:

```
> .dyncall(sqrtAddr, "d)d", 144)
[1] 12
```

The last call pinpoints the core solution for a direct invocation of foreign code within R: The first argument specifies the address of the foreign code, given as an external pointer. The second argument is a *call signature* that specifies the argument- and return types of the target C function. This string `"d)d"` specifies that the foreign function expects a double scalar argument and returns a double scalar value in accordance with the C declaration of `sqrt`. Arguments following the call signature are passed to the foreign function in the form specified by the *call signature*. In the example we pass 144 as a C double argument type as first argument and receive a C double value converted to an R numeric.

Call signatures

The introduction of a type descriptor for foreign functions is a key component that makes the FFI flexible and type-safe. The format of the call signature has the following pattern:

argument-types ') ' *return-type*

The signature can be derived from the C function declaration: Argument types are specified first, in the direct *left-to-right* order of the corresponding C function prototyp declaration, and are terminated by the symbol `') '` followed by a single return type signature.

Almost all fundamental C types are supported and there is no restriction² regarding the number of arguments supported to issue a call. Table 2 gives an

Type	Sign.	Type	Sign.
void	v	bool	B
char	c	unsigned char	C
short	s	unsigned short	S
int	i	unsigned int	I
long	j	unsigned long	J
long long	l	unsigned long long	L
float	f	double	d
void*	p	struct <i>name</i> *	*< <i>name</i> >
<i>type</i> *	*...	const char*	Z

Table 2: C/C++ Types and Signatures

C function declaration	Call signature
void rsort_with_index(double*, int*, int n)	*d*ii)v
SDL_Surface * SDL_SetVideoMode(int, int, int, Uint32_t)	iiiI)*<SDL_Surface>
void glClear(GLfloat, GLfloat, GLfloat, GLfloat)	ffff)v

Table 3: Examples of C functions and corresponding call signatures

overview of supported C types and the corresponding text encoding; Table 3 provides some examples of C functions and call signatures.

A public R function that encapsulates the details of the `sqrt` call is simply defined by

```
> sqrtViaDynCall <- function(...)
+ .dyncall(sqrtAddr, "d)d", ...)
```

No further guard code is needed here because `.dyncall` has built-in type checks that are specified by the signature. In contrast to the R wrapper code using `.C`, no explicit cast of the arguments via `as.numeric` is required, because automatic coercion rules for fundamental types are implemented as specified by the call signature. For example, using the integer literal `144L` instead of `double` works here as well.

```
> sqrtViaDynCall(144L)
[1] 12
```

If any incompatibility is detected, such as a wrong number of arguments, empty atomic vectors or incompatible type mappings, the invocation is aborted and an error is reported without risking an application crash.

Pointer type arguments, expressed via `'p'`, are handled differently. The type signature `'p'` indicates that the argument is an address. When passing R atomic vectors, the C argument value is the *address* of the *first element* of the vector. External pointers and the `NULL` object can also be passed as values for pointer type arguments. Automatic coercion is deliberately not implemented for pointer types. This is to support C functions that write into memory referenced by *out* pointer types.

Typed pointers, specified by the prefix `'*'` followed by the signature of the base type, offer a measure of type-safety for pointer types; if an R vector is passed and the R atomic type does not match the base type, the call will be rejected. Typed pointers to C `struct` and `union` types are also supported; they are briefly described in the section *Handling of C Types in R*.

In contrast to the R FFI, where the argument conversion is dictated solely by the R argument type at call-time in a one-way fashion, the introduction of an additional specification with a call signature gives several advantages.

- Almost all possible C functions can be invoked by a single interface; no additional C wrapper is required.
- The built-in type-safety checks enhance stability and significantly reduce the need for assertion code.
- The same call signature works across platforms, given that the C function type remains constant.
- Given that our FFI is implemented in multiple languages (e.g. Python, Ruby, Perl, Lua), call signatures represent a universal type description for C libraries.

Package overview

Besides dynamic calling of foreign code, the package provides essential facilities for interoperability between the R and C programming languages. An

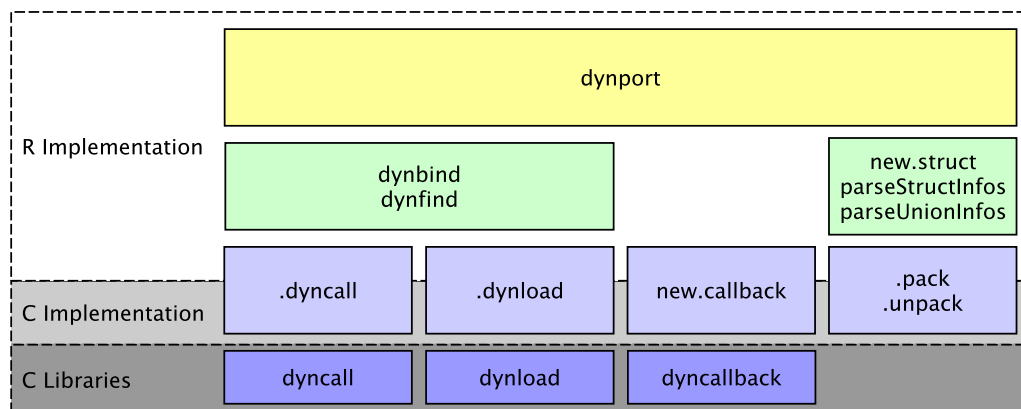


Figure 1: Package Overview

overview of components that make up the package is given in Figure 1.

We already described the `.dyncall` FFI. It is followed by a brief description of portable loading of shared libraries using `dynfind`, installation of wrappers via `dynbind`, handling of foreign data types via `new.struct` and wrapping of R functions as C callbacks via `new.callback`. Finally the high-level `dynport` interface for accessing *whole* C libraries is briefly discussed. The technical details at low-level of some components are described briefly in the section *Architecture*.

Portable loading of shared libraries

The *portable* loading of shared libraries across platforms is not trivial because the file path is different across operating systems. Referring back to the previous example, to load a particular library in a portable fashion, one would have to check the platform to locate the C library.³

Although there is variation among the operating systems, library file paths and search patterns have common structures. For example, among all the different locations, prefixes and suffixes, there is a part within a full library filename that can be taken as a *short library name* or label.

The function `dynfind` takes a list of short library names to locate a library using common search heuristics. For example, to load the *Standard C Math* library, depending on the operating system the library is either the *Microsoft Visual C Run-Time DLL* labeled `'msvcrt'` on Windows or the *Standard C Math* shared library labeled `'m'` or `'c'` on other operating systems.

```
> mLib <- dynfind(c("msvcrt", "m", "c"))
```

`dynfind` also supports more exotic schemes, such as Mac OS X Framework folders. Depending on the

³Possible C math library names are `'libm.so'` and `'MSVCRT.DLL'` in locations such as `'/lib'`, `'/usr/lib'`, `'/lib64'`, `'/usr/lib64'`, `'C:\WINDOWS\SYSTEM32'` etc..

library, it is sometimes enough to have a single short filename - e.g. `"expat"` for the *Expat* library.

Wrapping C libraries

Functional R interfaces to foreign code can be defined with small R wrapper functions, which effectively delegate to `.dyncall`. Each function interface is parameterized by a target address and a matching call signature.

```
f <- function(...) .dyncall(target, signature, ...)
```

Since an *Application Programming Interface* (API) often consist of hundreds of functions (see Table 1), `dynbind` can create and install a batch of function wrappers for a library with a single call by using a *library signature* that consists of concatenated function names and signatures separated by semicolons.

For example, to install wrappers to the C functions `sqrt`, `sin` and `cos` from the *math* library, one could use

```
> dynbind( c("msvcrt", "m", "c"),
+ "sqrt(d)d; sin(d)d; cos(d)d; " )
```

The function call has the side-effect that three R wrapper functions are created and stored in an environment that defaults to the global environment. Let us review the `sin` wrapper (on the 64-bit Version of R running on Mac OS X 10.6):

```
> sin
function (...)
.dyncall.cdecl(<pointer: 0x7fff81fd13f0>,
"d)d", ...)
```

The wrapper directly uses the address of the `sin` symbol from the *Standard C Math* library. In addition, the wrapper uses `.dyncall.cdecl`, which is a concrete selector of a particular calling convention, as outlined below.

Calling conventions

Calling conventions specify how arguments and return values are passed across sub-routines and functions at machine-level. This information is vital for interfacing with the binary interface of C libraries. The package has support for multiple calling conventions. Calling conventions are controlled by `.dyncall` via the named argument `callmode` to specify a non-default calling convention. Most supported operating systems and platforms only have support for a single "default" calling convention at run-time. An exception to this is the Microsoft Windows platform on the Intel *i386* processor architecture: While the default C calling convention on *i386* (excluding Plan9) is "default", system shared libraries from Microsoft such as 'KERNEL32.DLL', 'USER32.DLL' as well as the *OpenGL* library 'OPENGL32.DLL' use the "stdcall" calling convention. Only on this platform does the `callmode` argument have an effect. All other platforms currently ignore this argument.

Handling of C types in R

C APIs often make use of high-level C `struct` and `union` types for exchanging information. Thus, to make interoperability work at that level the handling of C data types is addressed by the package.

To illustrate this concept we consider the following example: A user-interface library has a function to set the 2D coordinates and dimension of a graphical output window. The coordinates are specified using a C `struct Rect` data type and the C function receives a pointer to that object:

```
void setWindowRect(struct Rect *pRect);
```

The structure type is defined as follows:

```
struct Rect {
  short      x, y;
  unsigned short w, h;
};
```

Before we can issue a call, we have to allocate an object of that size and initialize the fields with values encoded in C types that are not part of the supported set of R data types. The framework provides R helper functions and objects to deal with C data types. Type information objects can be created with a description of the C structure type. First, we create a type information object in R for the `struct Rect` C data type with the function `parseStructInfos` using a *structure type signature*.

```
> parseStructInfos("Rect{ssSS}x y w h;")
```

After registration, an R object named `Rect` is installed that contains C type information that corresponds to `struct Rect`. The format of a *structure type signature* has the following pattern:

```
Struct-name '{ ' Field-types ' } ' Field-names ' ; '
```

Field-types use the same type signature encoding as that of *call signatures* for argument and return types (Table 2). *Field-names* consist of a list of white-space separated names, that label each field component left to right.

An instance of a C type can be allocated via `new.struct`:

```
> r <- new.struct(Rect)
```

Finally, the extraction (`'$'`, `'['`) and replacement (`'$<-'`, `'[<-'`) operators can be used to access structure fields symbolically. During value transfer between R and C, automatic conversion of values with respect to the underlying C field type takes place.

```
> r$x <- -10 ; r$y <- -20 ; r$w <- 40 ; r$h <- 30
```

In this example, R numeric values are converted on the fly to signed and unsigned short integers (usually 16-bit values). On printing `r` a detailed picture of the data object is given:

```
> r
struct Rect {
  x: -10
  y: -20
  w: 40
  h: 30
}
```

At low-level, one can see that `r` is stored as an R raw vector object:

```
> r[]
[1] f6 ff ec ff 28 00 1e 00
attr(,"struct")
[1] "Rect"
```

To follow the example, we issue a foreign function call to `setRect` via `.dyncall` and pass in the `r` object, assuming the library is loaded and the symbol is resolved and stored in an external pointer object named `setWindowRectAddr`:

```
> .dyncall( setWindowRectAddr, "**<Rect>"v", r)
```

We make use of a typed pointer expression `**<Rect>` instead of the untyped pointer signature `'p'`, which would also work but does not prevent users from passing other objects that do not reference a `struct Rect` data object. Typed pointer expressions increase type-safety and use the pattern `**<Type-Name>`. The invocation will be rejected if the argument passed in is not of C type `Rect`. As `r` is tagged with an attribute `struct` that refers to `Rect`, the call will be issued. Typed pointers can also occur as return types that permit the manipulation of returned objects in the same symbolic manner as above.

C union types are supported as well but use the `parseUnionInfos` function instead for registration, and a slightly different signature format:

```
Union-name ' | ' Field-types ' } ' Field-names ' ; '
```

The underlying low-level C type read and write operations and conversions from R data types are performed by the functions `.pack` and `.unpack`. These can be used for various low-level operations as well, such as dereferencing of pointer to pointers.

R objects such as external pointers and atomic raw, integer and numeric vectors can be used as C struct/union types via the attribute `struct`. To *cast* a type in the style of C, one can use `as.struct`.

Wrapping R functions as C callbacks

Some C libraries, such as user-interface toolkits and I/O processing frameworks, use *callbacks* as part of their interface to enable registration and activation of user-supplied event handlers. A callback is a user-defined function that has a library-defined function type. Callbacks are usually registered via a registration function offered by the library interface and are activated later from within a library run-time context.

`rdyncall` has support for wrapping ordinary R functions as C callbacks via the function `new.callback`. Callback wrappers are defined by a *callback signature* and the user-supplied R function to be wrapped. *Callback signatures* look very similar to *call signatures* and should match the functional type of the underlying C callback. `new.callback` returns an external pointer that can be used as a low-level function pointer for the registration as a C callback. See Section *Parsing XML using Expat* below for applications of `new.callback`.

Foreign library interface

At the highest level, `rdyncall` provides the front-end function `dynport` to dynamically set up an interface to a C Application Programming Interface. This includes loading of the corresponding shared C library and resolving of symbols. During the binding process, a new R name space (Tierney, 2003) will be populated with thin R wrapper objects that represent abstractions to C counterparts such as functions, pointers-to-functions, type-information objects for C struct and union types and symbolic constant equivalents of C enums and macro definitions. The mechanism works across platforms; as long as the corresponding shared libraries of a *DynPort* have been installed in a system standard location on the host.

An initial repository of *DynPorts* is available in the package that provides bindings for several popular C APIs; see Table 1 for available bindings.

Sample applications

We give examples that demonstrate the direct usage of C APIs from within R through the `rdyncall` pack-

age. The R interface to C libraries looks very similar to the actual C API. For details on the usage of a particular C library, the programming manuals and documentation of the libraries should be consulted.

Before loading R bindings via `dynport`, the shared library should have been installed onto the system. Currently this is to be done manually and the installation method depends on the target operating system. While *OpenGL* and *Expat* is often pre-installed on typical desktop-systems, *SDL* usually has to be installed explicitly which is described in the package; see `?rdyncall-demos` for details.

OpenGL programming in R

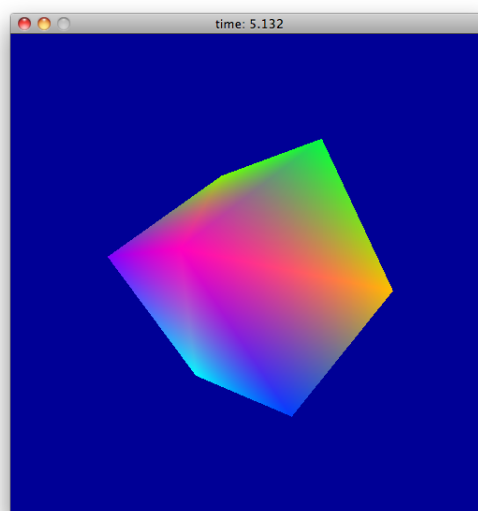


Figure 2: demo (SDL)

In the first example, we make use of the *Simple DirectMedia Layer* library (*SDL*) (Pendleton, 2003) and the *Open Graphics Library* (*OpenGL*) (OpenGL Architecture Review Board et al., 2005) to implement a portable multimedia application skeleton in R.

We first need to load bindings to *SDL* and *OpenGL* via `dynport`:

```
> dynport (SDL)
> dynport (GL)
```

Now we initialize the *SDL* library, e.g. we initialize the video subsystem, and open a 640x480 window surface in 32-bit color depths with support for *OpenGL* rendering:

```
> SDL_Init (SDL_INIT_VIDEO)
> surface <- SDL_SetVideoMode (640, 480, 32, SDL_OPENGL)
```

Next, we implement the application loop which updates the display repeatedly and processes the event queue until a *quit* request is issued by the user via the window close button.

```
> mainloop <- function()
{
  ev <- new.struct(SDL_Event)
  quit <- FALSE
  while(!quit) {
    draw()
    while(SDL_PollEvent(ev)) {
      if (ev$type == SDL_QUIT) {
        quit <- TRUE
      }
    }
  }
}
```

SDL event processing is implemented by collecting events that occur in a queue. Typical SDL applications poll the event queue once per update frame by calling `SDL_PollEvent` with a pointer to a user-allocated buffer of C type union `SDL_Event`. Event records have a common type identifier which is set to `SDL_QUIT` when a quit event has occurred, e.g. when users press a close button on a window.

Next we implement our `draw` function making use of the OpenGL API. We clear the background with a blue color and draw a light-green rectangle.

```
> draw <- function()
{
  glClearColor(0,0,1,0)
  glClear(GL_COLOR_BUFFER_BIT)
  glColor3f(0.5,1,0.5)
  glRectf(-0.5,-0.5,0.5,0.5)
  SDL_GL_SwapBuffers()
}
```

Now we can run the application `mainloop`.

```
> mainloop()
```

To stop the application, we press the close button of the window. A similar example is also available via `demo(SDL)`. Here the `draw` function displays a rotating 3D cube displayed in Figure 2.

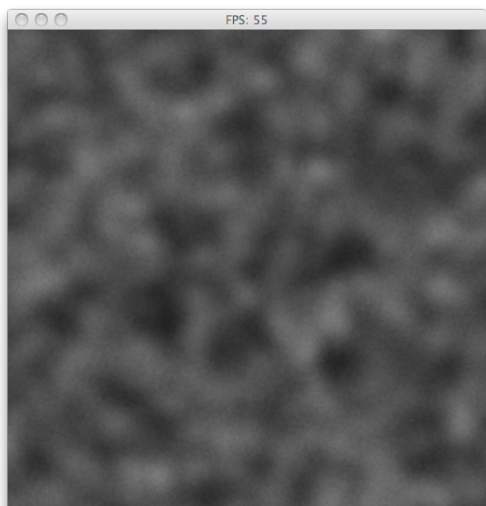


Figure 3: `demo(randomfield)`

`demo(randomfield)` gives a slightly more scientific application of OpenGL and R: Random fields of 512x512 size are generated via blending of 5000 texture mapped 2D gaussian kernels. The counter in the window title bar gives the number of matrices generated per second (see Figure 3). When clicking on the animation window, the current frame and matrix is passed to R and plotted. While several dozens of matrices are computed and drawn per second using OpenGL, it takes several seconds to plot a single matrix in R using `image()`.

Parsing XML using Expat

In the second example, we use the *Expat* XML Parser library (Clark, 2007; Kim, 2001) to implement a stream-oriented XML parser suitable for very large documents. In Expat, custom XML parsers are implemented by defining functions that are registered as callbacks to be invoked on events that occur during parsing, such as the start and end of XML tags. In our second example, we create a simple parser skeleton that prints the start and end tag names.

First we load R bindings for Expat via `dynport`.

```
> dynport(expat)
```

Next we create an abstract parser object via the C function `XML_ParserCreate` that receives one argument of type C string to specify a desired character encoding that overrides the document encoding declaration. We want to pass a null pointer (`NULL`) here. In the `.dyncall` FFI C null pointer values for pointer types are expressed via the R `NULL` value:

```
> p <- XML_ParserCreate(NULL)
```

The C interface for registering start- and end-tag event handler callbacks is given below:

```
/* Language C, from file expat.h: */
typedef void (*XML_StartElementHandler)
  (void *userData, const XML_Char *name,
   const XML_Char **atts);
typedef void (*XML_EndElementHandler)
  (void *userData, const XML_Char *name);
void XML_SetElementHandler(XML_Parser parser,
  XML_StartElementHandler start,
  XML_EndElementHandler end);
```

We implement the callbacks as R functions that print the event and tag name. They are wrapped as C callback pointers via `new.callback` using a matching *callback signature*. The second argument `name` of type C string in both callbacks, `XML_StartElementHandler` and `XML_EndElementHandler`, is of primary interest in this example; this argument passes over the XML tag name. C strings are handled in a special way by the `.dyncall` FFI because they have to be copied as R character objects. The special type signature 'Z' is used to denote a C string type. The other arguments are simply denoted as untyped pointers using 'p':

```

> start <- new.callback("pZp)v",
  function(ignored1,tag,ignored2)
    cat("Start tag:", tag, "\n")
  )
> end <- new.callback("pZ)v",
  function(ignored,tag)
    cat("Stop tag:", tag, "\n")
  )
> XML_SetElementHandler(p, start, end)

```

To test the parser we create a sample document stored in a character object named `text` and pass it to the parse function `XML_Parse`:

```

> text <- "<hello> <world> </world> </hello>"
> XML_Parse(p, text, nchar(text), 1)

```

The resulting output is

```

Start tag: hello
Start tag: world
End tag: world
End tag: hello

```

Expat supports processing of very large XML documents in a chunk-based manner by calling `XML_Parse` several times, where the last argument is used as indicator for the final chunk of the document.

Architecture

The core implementation of the FFI, callback wrapping and loading of code is based on small C libraries of the *DynCall* project (Adler and Philipp, 2011).

The implementation of the FFI is based on the **dyncall** C library, which provides an abstraction for making arbitrary machine-level calls offering a universal C interface for scripting language interpreters. It has support for almost all fundamental C argument/return types⁴ and multiple calling conventions, and is open for extension to other platforms and binary standards. Generic call implementations for the following processor architectures are supported: Intel i386 32-bit, AMD 64-bit, PowerPC 32-bit, ARM (including Thumb extension), MIPS 32/64-bit and SPARC 32/64-bit including support for several platform-, processor- and compiler-specific calling conventions.

The **dyncallback** C library implements generic callback handling. Callback handlers receive calls from C and they forward the call, including conversion of arguments, to a function of a scripting-language interpreter. A subset of architectures from the above is currently supported here: i386, AMD64 and ARM, and partial support for PowerPC 32-bit on Mac OS X/Darwin.

Besides the processor architecture, the libraries support various operating systems such as Linux, Mac OS X, Windows, the BSD family, Solaris, Haiku,

Minix and Plan9. Support for embedded platforms such as Playstation Portable, Nintendo DS and iOS is available as well. FFI implementations for other languages such as Python (van Rossum and Drake, Jr., 2005), Lua (Ierusalimsky et al., 1996) and Ruby (Flanagan and Matsumoto, 2008) are available from the *DynCall* project source repository.

The source tree supports various build tools such as gcc, msvc, SunPro, pcc, llvm and supports several make tools (BSD,C,GNU,N,Sun). A common abstraction layer for assembler dialects helps to develop cross-operating system call kernel. Due to the generic implementation and simple design, the libraries are quite small (the dyncall library for Mac OS X/AMD64 is 24 kb).

To test stability of the libraries, a suite of testing frameworks is available, including test-case generators with support for structured or random case studies and for testing extreme scenarios with large number of arguments. Prior to each release, the libraries and tests are built for a large set of architectures on **DynOS** (Philipp, 2011); a batch-build system using CPU emulators such as **QEmu** (Bellard, 2005) and **GXEmul** (Gavare, 2010), and various operating system images to test the release candidates and to create pre-built binary releases of the library.

Creation of DynPort files

The creation of *DynPort* files from C header files is briefly described next. A tool chain, comprising of freely available components, is applied once on a build machine as depicted in Figure 4.

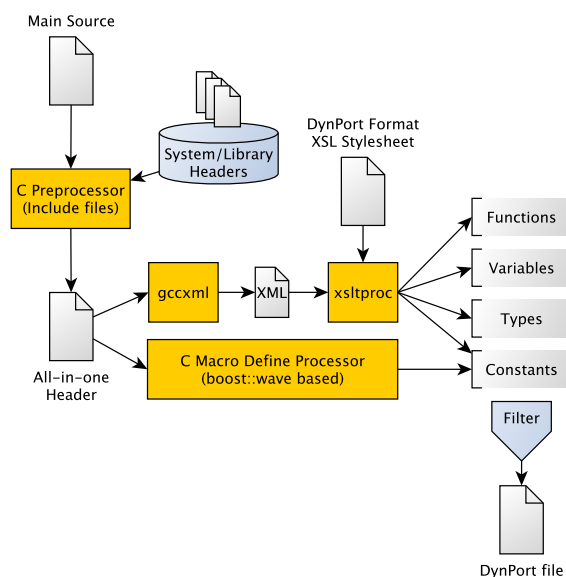


Figure 4: Tool-chain to create *DynPort* files from C headers

At first a main source file references the C header files of the library that should be made accessible via

⁴Passing of long double, struct and union argument/return C value types are currently work in progress.

`dynport`. In a preprocessing phase the **GNU C Macro Processor** is used to process all `#include` statements using standard system search paths to create a concatenated *All-In-One* source file. **GCC-XML** (King, 2004), a modified version of the GNU C compiler, transforms C header declarations to XML. The XML is further transformed to the final type signature format using **xslproc** (Veillard and Reese, 2009), a XSLT (Clark, 2001) processor, and a custom XSL stylesheet that has been implemented for the actual transformation from GCC-XML to the type signature text format.

C Macro `#define` statements are handled separately by a custom C Preprocessor implemented in C++ using the **boost wave** library (Kaiser, 2011). An optional filter stage is used to include only elements with a certain pattern, such as a common prefix usually found in many libraries, e.g. `'SDL_'`. In a last step, the various fragments are assembled into a single text-file that represents the *DynPort* file.

Limitations

During the creation of *DynPort* files, we encountered some cases (mainly for the SDL library) where we had to comment out some symbolic assignments (derived from C macro definitions) manually. These could not be converted as-is into valid R assignments because they consist of complex C expressions such as bit-shift operations. One could solve this problem by integrating a C interpreter within the tool-chain that deduces the appropriate type and value information from the replacement part of each C macro definitions; definitions with incomplete type could be rejected and constant values could be stored in a language-neutral encoding.

In order to use a single *DynPort* for a given C library across multiple platforms, its interface must be constant across platforms. *DynPort* does not support the conditional statements of the C preprocessor. Thus interfaces that use different types for arguments or structure fields depending on the architecture cannot be supported in a universal manner. For example, the *Objective-C Run-Time* C library of MacOS X uses a different number of fields within certain `struct` data types depending on whether the architecture is *i386* or alternative *AMD64* in which case padding fields are inserted in the middle of the structure. We are aware of this problem although we have not encountered a conflict with the given palette of C libraries available via *DynPorts* to R. A possible work around for such cases would be to offer separate *DynPorts* for different architectures.

dyncall and **dyncallback** currently lack support for handling `long double`, `struct` and `union` argument and return value types and architecture-specific vector types. Work is in progress to overcome this limitation. The middleware **BridJ** (Chafik,

2011) for the Java VM and C/C++ libraries, which uses **dyncall**, provides support for passing `struct` value types for a number of *i386* and *AMD64* platforms.

R character strings have a maximum size that can limit the number of library functions per `dynbind` function call. An improved *DynPort* file format and parser are being developed and are already available for **luadyncall**.

This version of *DynPort* does not capture the full range of the C type system. For example array and bit-field types are not supported; the pointer-to-function type in an argument list can only be specified using the void pointer `'*v'` or `'p'` instead of this (more informative) explicit type. An extended version of *DynPort*, that overcomes these inconveniences and that improves type safety, is being developed.

Certain restrictions apply when **rdyncall** is used to work with C libraries. These arise from limitations in R. For example the handling of C `float` pointers/arrays and `char` pointer-to-pointer types are not implemented in R. The functions `.unpack` and `.pack` are powerful helper functions designed to overcome these and some other restrictions. Additional helper functions are included, such as `floatraw` and `floatraw2numeric` that translate numeric R vectors to C `float` arrays and vice versa.

The portable loading of shared libraries via `dynfind` might require fine-tuning the list of short names when using less common R platforms such as BSDs and Solaris.

Related work

Several dynamic languages offer a flexible FFI, e.g. **ctypes** (Heller, 2011) for Python, **alien** (Mascarenhas, 2009) for Lua, **Rffi** (Temple Lang, 2011) for R, **CFFI** (Bielman, 2010) for Common LISP and the **FFI** module for Perl (Moore et al., 2008) and Ruby (Meissner, 2011). These all facilitate similar services such as foreign function calls and handling of foreign data. With the exception of **Rffi**, these also support wrapping of scripting functions as C callbacks. In most cases, the type information is specified in the grammar of the dynamic language. An exception to this is the **Perl FFI** that uses text-based type signatures similar to **rdyncall**.

ctypeslib (Kloss, 2008) is an extension to **ctypes** that comes closest to the idea of *DynPorts* in which Python **ctypes** statements are automatically generated from C library header files, also using **GCC-XML**. In contrast, the *DynPort* framework contributes a *compact text-based* type information format that is also used as the main user-interface for various tasks in **rdyncall**. This software design is applicable across languages and thus type information can be *shared* across platforms and languages at the

same time.

Specific alternatives to **dyncall** include **libffi** (Green, 2011) and **ffcall** (Haible, 2004). These are mature FFI libraries that use a *data-driven* C interface and have support for many platforms. Although not as popular as the first two, the **C/Invoke** library (Weisser, 2007) also offers a similar service with bindings to Lua, Java and Kite. The **dyncall** library offers a *functional* C interface (inspired by the OpenGL API). It includes a comprehensive test suite and detailed documentation of calling conventions on a variety of platforms and compilers. As the framework was developed "de novo" we were free to introduce our own strategy to support open as well as commercial and embedded platforms. For example, the i386 Assembly (except for Plan9) is implemented in a common abstract syntax that translates to GNU and Microsoft Assembler. This makes sense here, because i386-based operating systems use a common C calling convention which we address using a *single* Assembly source. A by-product of this feature is that **dyncall** enables the user to call operating system foreign code on some architectures.

In contrast to the *dynamic zero-compilation* approach of **ctypeslib** and **rdyncall**, the majority of language bindings to libraries use a *compiled* approach in which code (handwritten or auto-generated) is compiled for each platform. **SWIG** (Beazley, 2003) is a development tool for the automatic generation of language bindings. The user specifies the interface for a particular library in a C-like language and then chooses among the several supported languages (including R) to generate C sources that implement the binding for that particular library/language combination. **RGtk2** (Lawrence and Temple Lang, 2011) offers R bindings for the **GTK+** GUI framework consisting of R and C code. These are produced by a custom code generator to offer carefully conceived mappings to the object-oriented **GObject** framework. The generated code includes features such as ownership management of returned objects using human annotations. While custom bindings offer the ability to take into account the features of a particular library and framework to offer very user-friendly mapping schemes, **rdyncall** aims to offer convenient access to C libraries in general but it requires users to know the details of the particular interface of a C library and the R run-time environment.

Summary and Outlook

This paper introduces the **rdyncall** package⁵ that contributes an improved Foreign Function Interface for R. The FFI facilitates *direct* invocation of foreign functions *without* the need to compile wrappers in C. The FFI offers a dynamic cross-platform linkage framework to wrap and access *whole* C inter-

faces of native libraries from R. Instead of *compiling* bindings for every *library/language* combination, R bindings of a library are created dynamically at run-time in a data-driven manner via *DynPort* files, a cross-platform universal type information format. C libraries are made accessible in R as though they were extension packages and the R interface looks very similar to that of C. This enables system-level programming in R and brings a new wave of possibilities to R developers such as direct access to OpenGL across platforms as illustrated in the example. An initial repository of *DynPorts* for standard cross-platform portable C libraries comes with the package. Work is in progress for implementation of callback support on architectures already supported by the **dyncall** C library. The handling of foreign data types, which is currently implemented in R and C, is planned to be reimplemented as a C library and part of the *DynCall* project.

The *DynPort* facility in **rdyncall** constitutes an initial step in building up an infrastructure between scripting languages and C libraries. Analogous to the way in which R users enjoy quick access to the large pool of R software managed by CRAN, we envision an archive network in which C library developers can distribute their work across languages, users could then get quick access to the pool of C libraries from within scripting languages via automatic installation of precompiled components and using universal type information for cross-platform and cross-language dynamic bindings.

Bibliography

- D. Adler and T. Philipp. DynCall project. URL <http://dyncall.org>, November 2011. C library version 0.7.
- D. M. Beazley. Automated scientific software scripting with SWIG. *Future Gener. Comput. Syst.*, 19: 599–609, July 2003. ISSN 0167-739X. doi: 10.1016/S0167-739X(02)00171-1. URL <http://portal.acm.org/citation.cfm?id=860016.860018>.
- F. Bellard. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46. USENIX, 2005. URL <http://www.usenix.org/events/usenix05/tech/freenix/bellard.html>.
- J. Bielman. CFFI - common foreign function interface. URL <http://common-lisp.net/project/cffi/>, August 2010. CL library version 0.10.6.
- O. Chafik. BridJ - Let Java & Scala call C, C++, Objective-C, C#... URL <http://code.google.com/p/bridj/>, Jun 2011. Java package version 0.5.

⁵Version 0.7.4 on CRAN as of this writing.

- J. Clark. XSL transformations (XSLT) version 1.1. W3C working draft, W3C, Aug. 2001. URL <http://www.w3.org/TR/2001/WD-xslt11-20010824/>.
- J. Clark. The Expat XML parser. URL <http://expat.sourceforge.net/>, June 2007. C library version 2.0.1.
- D. Flanagan and Y. Matsumoto. *The Ruby Programming Language*. O'Reilly, Cambridge, 2008.
- A. Gavare. GXEmul: a framework for full-system computer architecture emulation. URL <http://gxemul.sourceforge.net/>, February 2010. Program version 0.6.0.
- A. Green. libffi - a portable foreign function interface library. URL <http://sourceware.org/libffi/>, August 2011. C library version 3.0.10.
- B. Haible. fcall - foreign function call libraries. URL <http://www.gnu.org/s/libffcall/>, June 2004. C library version 1.10.
- T. Heller. ctypes - A foreign function library for Python. URL <http://starship.python.net/crew/theller/ctypes/>, October 2011.
- R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho. Lua – an extensible extension language. *Software – Practice and Experience*, 26(6):635–652, June 1996.
- H. Kaiser. Wave V2.0 - Boost C++ Libraries. URL <http://www.boost.org/doc/libs/release/libs/wave/index.html>, July 2011. Boost C++ Library Version 1.45, Wave C++ Library Version 2.1.0.
- E. E. Kim. A triumph of simplicity: James Clark on markup languages and XML. *Dr. Dobb's Journal of Software Tools*, 26(7):56, 58–60, July 2001. ISSN 1044-789X. URL <http://www.ddj.com/>.
- B. King. GCC-XML. URL <http://www.gccxml.org>, February 2004. Program version 0.6.0.
- G. K. Kloss. Automatic C library wrapping – ctypes from the trenches. *The Python Papers*, 3(3), 2008. ISSN 1834-3147.
- S. Lantinga. libSDL: Simple DirectMedia layer. URL <http://www.libsdl.org/>, October 2009. C library version 1.2.14.
- M. Lawrence and D. Temple Lang. RGtk2: A graphical user interface toolkit for R. *Journal of Statistical Software*, 2011. ISSN 15487660. URL <http://www.jstatsoft.org/v37/i08/paper>.
- F. Mascarenhas. Alien - pure Lua extensions. URL <http://alien.luaforge.net/>, October 2009. Lua module version 0.5.1.
- W. Meissner. Ruby-FFI. URL <https://github.com/ffi/ffi/wiki>, October 2011. Ruby package version 1.0.10.
- P. Moore, G. Yahas, and A. Vorobey. FFI - Perl foreign function interface. URL <http://search.cpan.org/~gaal/FFI/FFI.pm>, September 2008. Perl module version 1.04.
- OpenGL Architecture Review Board, D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2*. Addison Wesley, 2005.
- B. Pendleton. Game programming with the Simple DirectMedia Layer (SDL). *Linux Journal*, 110:42, 44, 46, 48, June 2003. ISSN 1075-3583.
- T. Philipp. DynOS Project. URL <http://dyncall.org/dynos>, May 2011.
- R Development Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2010. URL <http://www.R-project.org>. ISBN 3-900051-11-9.
- D. Temple Lang. Rffi for run-time invocation of arbitrary compiled routines from R. URL <http://www.omegahat.org/Rffi/>, January 2011. R package version 0.3-0.
- L. Tierney. A simple implementation of name spaces for R. URL <http://www.stat.uiowa.edu/~luke/R/namespaces/morenames.pdf>, May 2003.
- G. van Rossum and F. L. Drake, Jr. *Python Language Reference Manual*. Network Theory Ltd., 2005. ISBN 0-9541617-8-5. URL <http://www.network-theory.co.uk/python/language/>.
- D. Veillard and B. Reese. The XSLT C library for GNOME. URL <http://xmlsoft.org/XSLT/>, September 2009. C library version 1.1.26.
- W. Weisser. C/Invoke - version 1.0 - easily call C from any language. URL <http://cinvoke.teegra.net/index.html>, January 2007. C library version 1.0.

Daniel Adler
 Georg-August Universität
 Institute for Statistics and Economics
 Platz der Göttinger Sieben 5
 37079 Göttingen, Germany
 dadler@uni-goettingen.de

Vdgraph: A Package for Creating Variance Dispersion Graphs

by John Lawson

Abstract This article introduces the package **Vdgraph** that is used for making variance dispersion graphs of response surface designs. The package includes functions that make the variance dispersion graph of one design or compare variance dispersion graphs of two designs, which are stored in data frames or matrices. The package also contains several minimum run response surface designs (stored as matrices) that are not available in other R packages.

Introduction

Response surface methods consist of (1) experimental designs for collecting data to fit an approximate relationship between the factors and the response, (2) regression analyses for fitting the model and (3) graphical and numerical techniques for examining the fitted model to identify the optimum. The model normally used in response surface analysis is a second order polynomial, as shown in Equation (1).

$$y = \beta_0 + \sum_{i=1}^k \beta_i x_i + \sum_{i=1}^k \beta_{ii} x_i^2 + \sum_{i < j}^k \beta_{ij} x_i x_j \quad (1)$$

The fitted equation is examined in order to predict the factor coordinates of the maximum or minimum response within the experimental region, or to simply explore the relationship between the factors and response.

Since it is not known in advance what neighborhood will be of most interest in the design space, a desirable response surface design will be one that makes the variance of a predicted value as uniform as possible throughout the experimental region. Standard response surface designs, such as the uniform precision central composite design, are constructed so that the variance of a predicted value will be near constant within a coded radius of one from the center of the design.

One way to visualize the uniformity of the variance of a predicted value for designs with more than two factors is to use the variance dispersion graph proposed by Myers et al. (1992).

Variance of a predicted value

The variance of a predicted value at a point (x_1, \dots, x_k) in the experimental region is given by Equation (2)

$$\text{Var}[\hat{y}(\mathbf{x})] = \sigma^2 \mathbf{x}'(\mathbf{X}'\mathbf{X})^{-1} \mathbf{x} \quad (2)$$

where \mathbf{X} is the design matrix for the quadratic model in Equation (1), σ^2 is the variance of the experimental error, and

$$\mathbf{x} = [1, x_1, \dots, x_k, x_1^2, \dots, x_k^2, x_1 x_2, \dots]$$

is a vector valued function of the coordinates (of the point in the experimental region) whose elements correspond to the columns of the design matrix \mathbf{X} .

Run	x_1	x_2
1	-1	-1
2	1	-1
3	-1	1
4	1	1
5	-1	0
6	1	0
7	0	-1
8	0	1
9	0	0

Table 1: Face Center Cube Design or 3^2 Design

For the face-centered cube design, or 3^2 design shown in Table 1, Figure 1 is a contour plot of the scaled variance of a predicted value in the range of $-1.5 \leq x_1 \leq 1.5, -1.5 \leq x_2 \leq 1.5$. The scaled variance of a predicted value is $N\text{Var}[\hat{y}(\mathbf{x})]/\sigma^2$, where N is the number of points in the experimental design.

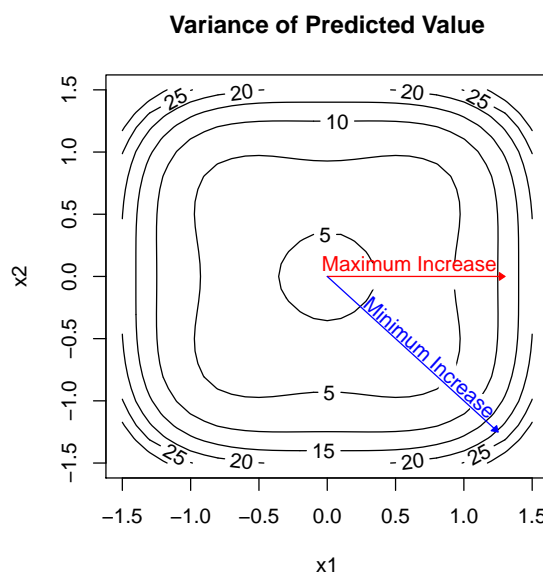


Figure 1: Contour plot of $N\text{Var}[\hat{y}(\mathbf{x})]/\sigma^2$

As seen in Figure 1, the variance of a predicted value increases faster along the line $x_2 = 0$ than along

the line $x_2 = -x_1$. This is easy to visualize in two dimensions, but would be more difficult to see in higher dimensions.

Variance dispersion graphs

A variance dispersion graph allows one to visualize the uniformity of the scaled variance of a predicted value in multidimensional space. It consists of three curves: the maximum, the minimum and the average scaled variance of a predicted value on a hypersphere. Each value is plotted against the radius of the hypersphere. Figure 2 shows the variance dispersion graph of the design shown in Table 1.

In this figure it can be seen that the maximum scaled variance of a predicted value is near 14 at a radius of 1.4 in coded units, while the minimum scaled variance is less than 10 at the same radius. This is the same phenomenon that can be seen in the contour plot of the scaled variance of a predicted value shown in Figure 1. The path of the maximum and minimum variance through the design space will be determined by the design and may not follow straight lines as shown in the specific example in this contour plot.

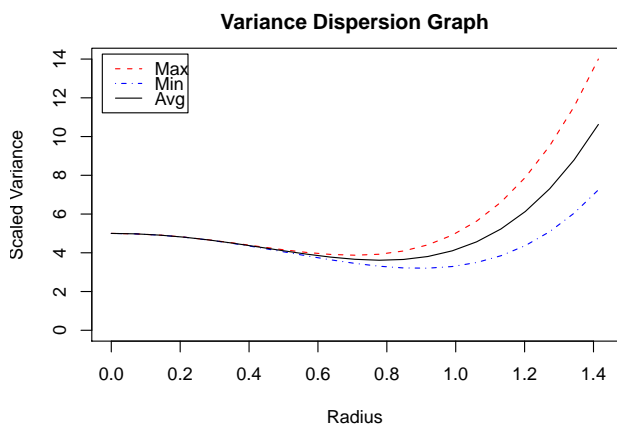


Figure 2: Variance dispersion graph for the design in Table 1.

Unlike the contour plot of the scaled prediction variance, the variance dispersion graph has the same format for a k dimensional response surface design as for a two dimensional design like Table 1.

Recent textbooks such as Montgomery (2005), Myers et al. (2009), and Lawson (2010) illustrate variance dispersion graphs as a tool for judging the merits of a response surface design. These graphs can be produced in commercial software such as SAS ADX (see SAS Institute Inc., 2010) and Minitab (see Minitab Inc., 2010) by using a downloadable macro (see Santiago, 2009).

The Vdgraph package

Vining (1993a), and Vining (1993b) published FORTRAN code for creating variance dispersion graphs. Vining's code obtains the maximum and minimum prediction variance on hyperspheres using a combination of a grid search and Nelder-Mead search as described by Cook and Nachtsheim (1980). The package **Vdgraph** (Lawson, 2011) incorporates this code in R functions that make the graphs.

The package includes the function `Vdgraph` for making a variance dispersion graph of one design and the function `Compare2Vdg` for comparing the variance dispersion graphs of two designs on the same plot. The package also includes several minimum run response surface designs stored as matrices. These include Hartley's small composite design for 2 to 6 factors, Draper and Lin's small composite design for 5 factors, the hexagonal rotatable design for 2 factors and Roquemore's hybrid designs for 3 to 6 factors.

Examples

The first example shown below illustrates the use of the R function `Vdgraph` to make variance dispersion graphs of a three factor Box-Behnken design created by the `bbd` function in the R package **rsm** (see Lenth, 2009).

```
> library(rsm)
> BB.des3 <- bbd(3)
> Vdgraph(BB.des3)
number of design points= 16
number of factors= 3
```

	Radius	Maximum	Minimum	Average
[1,]	0.00000000	4.000000	4.000000	4.00000
[2,]	0.08660254	3.990100	3.990067	3.99008
[3,]	0.17320508	3.961600	3.961067	3.96128
[4,]	0.25980762	3.918100	3.915400	3.91648
[5,]	0.34641016	3.865600	3.857067	3.86048
[6,]	0.43301270	3.812500	3.791667	3.80000
[7,]	0.51961524	3.769600	3.726400	3.74368
[8,]	0.60621778	3.750100	3.670067	3.70208
[9,]	0.69282032	3.769600	3.633067	3.68768
[10,]	0.77942286	3.846100	3.627400	3.71488
[11,]	0.86602540	4.000000	3.666667	3.80000
[12,]	0.95262794	4.254106	3.766067	3.96128
[13,]	1.03923048	4.633600	3.942400	4.21888
[14,]	1.12583302	5.166116	4.214067	4.59488
[15,]	1.21243557	5.881600	4.601067	5.11328
[16,]	1.29903811	6.812500	5.125000	5.80000
[17,]	1.38564065	7.993638	5.809067	6.68288
[18,]	1.47224319	9.462100	6.678067	7.79168
[19,]	1.55884573	11.257600	7.758400	9.15808
[20,]	1.64544827	13.422175	9.078067	10.81568
[21,]	1.73205081	16.000000	10.666667	12.80000

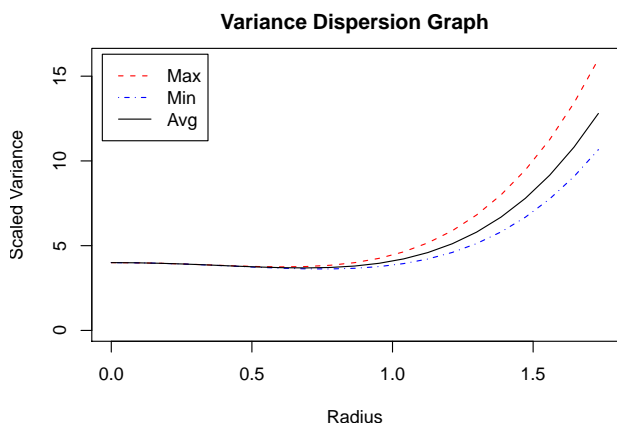


Figure 3: Variance dispersion graph for `BB.des3`.

The result from the first example (shown above) includes a listing of the coordinates of the plot and the graph shown in Figure 3.

The second example illustrates the use of `Compare2Vdg` by comparing the variance dispersion graph of Draper and Lin’s small composite design for 5 factors (`SCDDL5`) (Draper and Lin, 1990) with Hartley’s Small Composite Design (`SCDH5`) (Hartley, 1959). Hartley’s small composite design requires only 28 runs by utilizing a $\frac{1}{2}$ fraction of the factorial portion of the design.

Run	x_1	x_2	x_3	x_4	x_5
1	1	-1	1	1	1
2	1	1	-1	-1	-1
3	-1	1	1	-1	1
4	1	-1	1	-1	1
5	1	1	-1	1	1
6	1	1	1	-1	-1
7	-1	1	1	1	-1
8	-1	-1	1	1	-1
9	-1	-1	-1	-1	1
10	1	-1	-1	1	-1
11	-1	1	-1	1	1
12	-1	-1	-1	-1	-1
13	$-\alpha$	0	0	0	0
14	α	0	0	0	0
15	0	$-\alpha$	0	0	0
16	0	α	0	0	0
17	0	0	$-\alpha$	0	0
18	0	0	α	0	0
19	0	0	0	$-\alpha$	0
20	0	0	0	α	0
21	0	0	0	0	$-\alpha$
22	0	0	0	0	α
23	0	0	0	0	0

Table 2: Draper and Lin’s Small Composite Design for 5 Factors

Although Draper and Lin’s design (shown in Table 2 with $\alpha = 1.86121$) further reduces the number of

runs to 23, by substituting a 12 run Plackett-Burman design in the factorial portion, its variance dispersion graph reveals that the variance of a predicted value is not nearly as uniform as it is for the Hartley’s design.

```
> data(SCDH5)
> data(SCDDL5)
> Compare2Vdg("Hartley's Small Composite-5 fac",SCDH5,
> +"Draper and Lin's Small Composite-5 fac",SCDDL5)
```

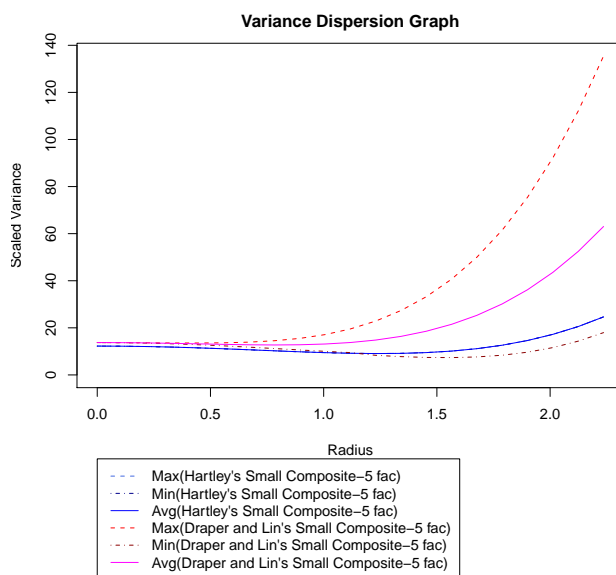


Figure 4: Comparison of Two Variance Dispersion Graphs.

As seen in Figure 4, Hartley’s small composite design for 5 factors is rotatable since the three blue curves for the max, min and average scaled prediction variance coincide. The scaled variance of a predicted value for Hartley’s design is near the minimum scaled variance of a predicted value for Draper and Lin’s design throughout the experimental region.

Acknowledgement

I would like to thank the editor and two reviewers for helpful suggestions that improved this article and the package `Vdgraph`.

Bibliography

R. D. Cook and C. J. Nachtsheim. A comparison of algorithms for constructing d-optimal designs. *Technometrics*, 22:315–324, 1980.

N. R. Draper and D. K. J. Lin. Small response surface designs. *Technometrics*, 32:187–194, 1990.

H. O. Hartley. Smallest composite design for quadratic response surfaces. *Biometrics*, 15:611–624, 1959.

- J. Lawson. *Vdgraph: This package creates variance dispersion graphs for response surface designs*, 2011. URL <http://CRAN.R-project.org/package=Vdgraph>. R package version 1.0-1.
- J. S. Lawson. *Design and Analysis of Experiments with SAS*. CRC Press, Boca Raton, 2010.
- R. V. Lenth. Response surface methods in R, using rsm. *Journal of Statistical Software*, 32(7):1–17, 2009.
- Minitab Inc. Minitab software for quality improvement, 2010. URL <http://www.minitab.com>.
- D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, sixth edition, 2005.
- R. H. Myers, G. Vining, A. Giovannitti-Jensen, and S. L. Myers. Variance dispersion properties of second order response surface designs. *Journal of Quality Technology*, 24:1–11, 1992.
- R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. John Wiley & Sons, New York, 2009.
- E. Santiago. Macro: VDG.MAC, 2009. URL www.minitab.com/support/macros/default.aspx.
- SAS Institute Inc. Getting started with the SAS 9.2 ADX interface for design of experiments, 2010. URL <http://support.sas.com/documentation/cdl/en/adxgs/60376/PDF/default/adxgs.pdf>.
- G. G. Vining. A computer program for generating variance dispersion graphs. *Journal of Quality Technology*, 25:45–58, 1993a.
- G. G. Vining. Corrigenda: A computer program for generating variance dispersion graphs. *Journal of Quality Technology*, 25:333–335, 1993b.

John Lawson
Department of Statistics
Brigham Young University
Provo, UT (USA)
lawson@byu.edu

xgrid and R: Parallel Distributed Processing Using Heterogeneous Groups of Apple Computers

by Sarah C. Anoke, Yuting Zhao, Rafael Jaeger and Nicholas J. Horton

Abstract The Apple Xgrid system provides access to groups (or grids) of computers that can be used to facilitate parallel processing. We describe the **xgrid** package which facilitates access to this system to undertake independent simulations or other long-running jobs that can be divided into replicate runs within R. Detailed examples are provided to demonstrate the interface, along with results from a simulation study of the performance gains using a variety of grids. Use of the grid for “embarrassingly parallel” independent jobs has the potential for major speedups in time to completion. Appendices provide guidance on setting up the workflow, utilizing add-on packages, and constructing grids using existing machines.

Introduction

Many scientific computations can be sped up by dividing them into smaller tasks and distributing the computations to multiple systems for simultaneous processing. Particularly in the case of *embarrassingly parallel* (Wilkinson and Allen, 1999) statistical simulations, where the outcome of any given simulation is independent of others, parallel computing on existing *grids* of computers can dramatically increase computation speed. Rather than waiting for the previous simulation to complete before moving on to the next, a *grid controller* can distribute *tasks* to *agents* (also known as *nodes*) as quickly as they can process them in parallel. As the number of nodes in the grid increases, the total computation time for a given job will generally decrease. Figure 1 provides a conceptual model of this framework.

Several solutions exist to facilitate parallel computation within R. Wegener et al. (2007) developed **GridR**, a condor-based environment for settings where one can connect directly to agents in a grid. The **Rmpi** package (Yu, 2002) is an R wrapper for the popular Message Passing Interface (MPI) protocol and provides extremely low-level control over grid functionality. The **rpvm** package (Li and Rossini, 2001) provides a connection to a Parallel Virtual Machine (Geist et al., 1994). The **snow** package (Rossini et al., 2007) provides a simpler implementation of **Rmpi** and **rpvm**, using a low-level socket functionality. The **multicore** package (Urbanek, 2011) pro-

vides several functions to divide work between a single machine’s multiple cores. Starting with release 2.14.0, **snow** and **multicore** are available as slightly revised copies within the **parallel** package in base R.

The Apple Xgrid (Apple Inc., 2009) technology is a parallel computing environment. Many Apple Xgrids already exist in academic settings, and are straightforward to set up. As loosely organized clusters, Apple Xgrids provide *graceful degradation*, where agents can easily be added to or removed from the grid without disrupting its operation. Xgrid supports heterogeneous agents (also a plus in many settings, where a single grid might include a lab, classroom, individual computers, as well as more powerful dedicated servers) and provides automated housekeeping and cleanup. The Xgrid Admin program provides a graphical overview of the controller, agents and jobs that are being managed (instructions on downloading and installing this tool can be found in Appendix C).

We created the **xgrid** package (Horton and Anoke, 2012) to provide a simple interface to this distributed computing system. The package facilitates use of an Apple Xgrid for distributed processing of a simulation with many independent repetitions, by simplifying job submission (or *grid stuffing*) and collation of results. It provides a relatively thin but useful layer between R and Apple’s ‘xgrid’ shell command, where the user constructs input scripts to be run remotely. A similar set of routines, optimized for parallel estimation of JAGS (just another Gibbs sampler) models is available within the **runjags** package (Denwood, 2010). However, with the exception of **runjags**, none of the previously mentioned packages support parallel computation over an Apple Xgrid.

We begin by describing the **xgrid** package interface to the Apple Xgrid, detailing two examples which utilize this setup, summarizing simulation studies that characterize the performance of a variety of tasks on different grid configurations, then close with a summary. We also include a glossary of terms and provide three appendices detailing how to access a grid using R (Appendix A), how to utilize add-on packages within R (Appendix B), and how to construct a grid using existing machines (Appendix C).

Controlling an Xgrid using R

To facilitate use of an Apple Xgrid using R, we created the **xgrid** package, which contains support rou-

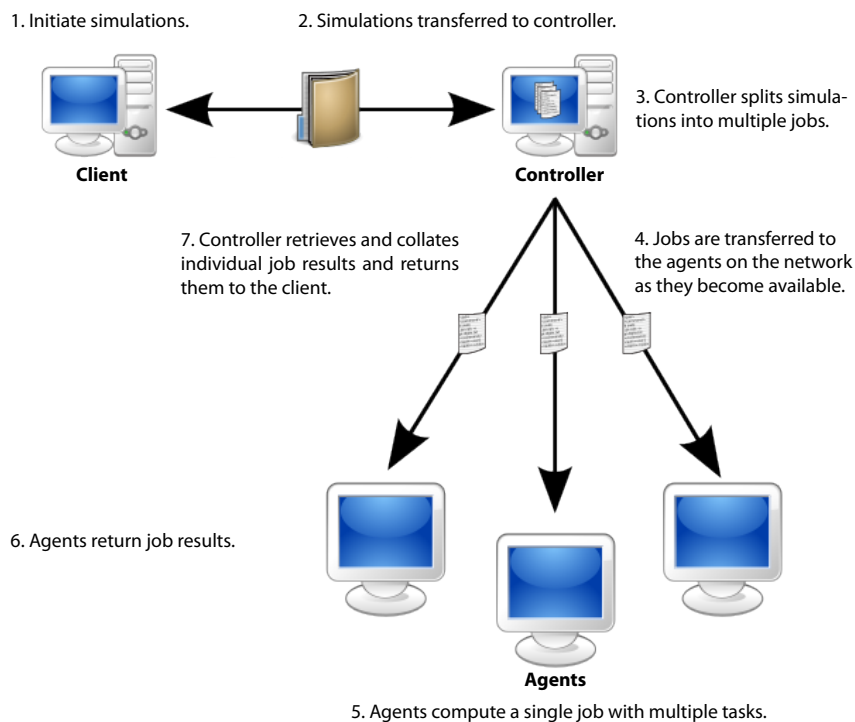


Figure 1: Conceptual model of the Apple Xgrid framework (derived from graphic by F. Loxy)

tines to split up, submit, monitor, then retrieve results from a series of simulation studies.

The `xgrid()` function connects to the grid by repeated calls to the `xgrid` command at the Mac OS X shell level on the client. Table 1 displays some of the actions supported by the `xgrid` command and their analogous routines in the `xgrid` package. While users will ordinarily not need to use these routines, they are helpful in understanding the workflow. These routines are designed to call a specified R script with suitable environment (packages, input files) on a remote machine. The remote job is given arguments as part of a call to `R CMD BATCH`, which allow it to create a unique location to save results, which are communicated back to the client by way of R object files created with the `R saveRDS()` function. Much of the work involves specifying a naming structure for the jobs, to allow the results to be automatically collated.

The `xgrid()` function is called to start a series of simulations. This function takes as arguments the R script to run on the grid (by default set to `'job.R'`), the directory containing input files (by default set to `'input'`), the directory to save output created within R on the agent (by default set to `'output'`), and a name for the results file (by default set to `'RESULTS.rds'`). In addition, the total number of iterations in the simulation (`numsim`) and number of tasks per job (`ntask`) can be specified. The `xgrid()` function divides the total number of iterations into `numsim/ntask` individual jobs, where each job is responsible for calculating the specified number of tasks on a single agent (see Figure 1). For example, if 2,000 iterations are desired,

these could be divided into 200 jobs each running 10 of the tasks. The number of active jobs on the grid can be controlled using the `throttle` option (by default, all jobs are submitted then queued until an agent is available). The `throttle` option helps facilitate sharing a large grid between multiple users (since by default the Apple Xgrid system provides no load balancing amongst users).

The `xgrid()` function checks for errors in specification, then begins to repeatedly call the `xgridsubmit()` function for each job that needs to be created. The `xgrid()` function also calls `xgridsubmit()` to create a properly formatted `'xgrid -job submit'` command using Mac OS X through the `R system()` function. This has the effect of executing a command of the form `'R CMD BATCH file.R'` on the grid, with appropriate arguments (the number of repetitions to run, parameters to pass along and the name of the unique filename to save results). The results of the `system()` call are saved to be able to determine the identification number for that particular job. This number can be used to check the status of the job as well as retrieve its results and delete it from the system once it has completed.

Once all of the jobs have been submitted, `xgrid()` then periodically polls the list of active jobs until they are completed. This function makes a call to `xgridattr()` and determines the value of the `jobStatus` attribute. The function waits (sleeps) between each poll, to lessen load on the grid.

When a job has completed, its results are retrieved using `xgridresults()` then deleted from the

Action	R Function	Description
submit	xgridsubmit()	submit a job to the grid controller
attributes	xgridattr()	check on the status of a job
results	xgridresults()	retrieve the results from a completed job
delete	xgriddelete()	delete the job

Table 1: Job actions supported by the `xgrid` command and their analogous functions in the `xgrid` package

system using `xgriddelete()`. This capability relies on the properties of the Apple Xgrid, which can be set up to have all files created by the agent when running a given job copied to the ‘output’ directory on the client computer. When all jobs have completed, the individual result files are combined into a single data frame in the current directory. The ‘output’ directory has a complete listing of the individual results as well as the R output from the remote agents. This directory can be useful for debugging in case of problems and the contents are typically accessed only in those cases.

To help demonstrate how to access an existing Xgrid, we provide two detailed examples: one involving a relatively straightforward computation assessing the robustness of the one-sample t-test and the second requiring use of add-on packages to undertake simulations of a latent class model. These examples are provided as vignettes within the package. In addition, the example files are available for download from <http://www.math.smith.edu/xgrid>.

Examples

Example 1: Assessing the robustness of the one-sample t-test

The t-test is remarkably robust to violations of its underlying assumptions (Sawilowsky and Blair, 1992). However, as Hesterberg (2008) argues, not only is it possible for the total non-coverage to exceed α , the asymmetry of the test statistic causes one tail to account for more than its share of the overall α level. Hesterberg found that sample sizes in the thousands were needed to get symmetric tails.

In this example, we demonstrate how to utilize an Apple Xgrid cluster to investigate the robustness of the one-sample t-test, by looking at how the α level is split between the two tails. When the number of simulation iterations is small ($< 100,000$), this study runs very quickly as a loop in R. Here, we provide the computation of a study consisting of 10^6 iterations. A more efficient alternative would be to use `pvec()` or `mclapply()` of the `multicore` package to distribute this study over the cores of a single machine. However for the purposes of illustration, we conduct this simple statistical investigation over an Xgrid to demonstrate package usage, and to compare the results and computation time to the same study

run with a `for` loop on a local machine.

Our first step is to set up an appropriate directory structure for our simulation (see Figure 5; Appendix A provides an overview of requirements). The first item is the folder ‘input’, which contains two files that will be run on the remote agents. The first of these files, ‘job.R’ (Figure 2), defines the code to run a particular task, `ntask` times.

In this example, the `job()` function begins by generating a sample of `param` exponential random variables with mean 1. A one-sample t-test is conducted on this sample and logical (TRUE/FALSE) values denoting whether the test rejected in that tail are saved in the vectors `leftreject` and `rightreject`. This process is repeated `ntask` times, after which `job()` returns a data frame with the rejection results and the corresponding sample size.

```
# Assess the robustness of the one-sample
# t-test when underlying data are exponential.
# This function returns a data frame with
# number of rows equal to the value of "ntask".
# The option "param" specifies the sample size.
job <- function(ntask, param) {
  alpha <- 0.05 # how often to reject under null
  leftreject <- logical(ntask) # placeholder
  rightreject <- logical(ntask) # for results
  for (i in 1:ntask) {
    dat <- rexp(param) # generate skewed data
    left <- t.test(dat, mu = 1,
                  alternative = "less")
    leftreject[i] <- left$p.value <= alpha/2
    right <- t.test(dat, mu = 1,
                   alternative = "greater")
    rightreject[i] <- right$p.value <= alpha/2
  }
  return(data.frame(leftreject, rightreject,
                   n = rep(param, ntask)))
}
```

Figure 2: Contents of ‘job.R’

The folder ‘input’ also contains ‘runjob.R’ (Figure 3), which retrieves command line arguments generated by `xgrid()` and passes them to `job()`. The results from the completed job are saved as `res0`, which is subsequently saved to the ‘output’ folder.

The folder ‘input’ may also contain other files (including add-on packages or other files needed for the simulation; see Appendix B for details).

```

source("job.R")
# commandArgs() is expecting three arguments:
# 1) number of tasks to run within this job
# 2) parameter to pass to the function
# 3) place to stash the results when finished
args    <- commandArgs(trailingOnly = TRUE)
ntask1  <- as.numeric(args[1])
param1  <- args[2]
resfile <- args[3]
res0    <- job(ntask = ntask1, param = param1)
# stash the results
saveRDS(res0, file = resfile)

```

Figure 3: Contents of 'runjob.R'

The next item in the directory structure is 'simulation.R' (Figure 4), which contains R code to be run on the client machine using `source()`. The call to `xgrid()` submits the simulation to the grid for calculation, which includes passing `param` and `ntask` to `job()` within 'job.R'. Results from all jobs are returned as one data frame, `res`. The call to `with()` summarizes all results in a table and prints them to the console.

```

require(xgrid)
# run the simulation
res <- xgrid(Rcmd = "runjob.R", param = 30,
             numsim = 10^6, ntask = 5*10^4)
# analyze the results
with(res, table(leftreject, rightreject))

```

Figure 4: Contents of 'simulation.R'

Here we specify a total of 10^6 simulation iterations, to be split into twenty jobs of 5×10^4 simulations each. Note that the number of jobs is calculated as the total number of simulation iterations (`numsim`) divided by the number of tasks per job (`ntask`). Each simulation iteration has a sample size of `param`.

The final item in the directory structure is 'output'. Initially empty, results returned from the grid are saved here (this directory is automatically created if it does not already exist).

Figure 5 displays the file structure within the directory used to access the Xgrid.

Jobs are submitted to the grid by running 'simulation.R'. In this particular simulation, twenty jobs are submitted. As jobs are completed, the results are saved in the 'output' folder then removed from the grid.

Figures 6 and 7 display management tools available using the Xgrid Admin interface.

In addition to returning a data frame (10^6 rows and 3 columns) with the collated results, the `xgrid()` function saves this object as a file (by default as `res` in the file 'RESULTS.rds').

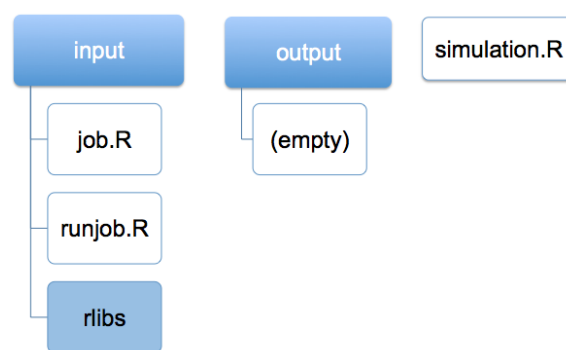


Figure 5: File structure needed on the client to access the Xgrid (the 'rlibs' folder contains any add-on packages required by the remote agent)



Figure 6: Monitoring overall grid status using the Xgrid Admin application

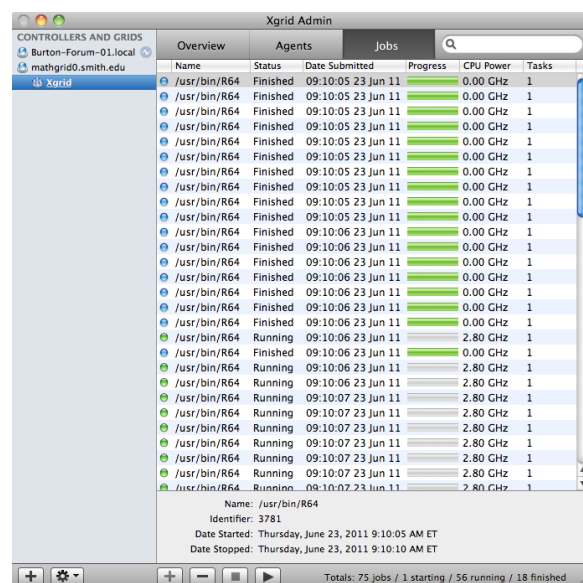


Figure 7: Job management using Xgrid Admin

Criteria	Acronym	Class				
		2	3	4	5	6+
Bayesian Information Criterion	BIC	49%	44%	7%	0%	0%
Akaike Information Criterion	AIC	0%	0%	53%	31%	16%
Consistent Akaike Information Criterion	cAIC	73%	25%	2%	0%	0%
Sample Size Adjusted Bayesian Information Criterion	aBIC	0%	5%	87%	6%	2%

Table 2: Percentage of times (out of 100 simulations) that a particular number of classes was selected as the best model (where the true data derive from 4 distinct and equiprobable classes, simulation sample size 300), reprinted from Swanson et al. (2011). Note that a perfect criterion would have “100%” under class 4.

In terms of our motivating example, when the underlying data are normally distributed, we would expect to reject the null hypothesis 2.5% of the time on the left and 2.5% on the right. The simulation yielded rejection rates of 6.5% and 0.7% for left and right, respectively. This confirms Hesterberg’s argument regarding lack of robustness for both the overall α -level as well as the individual tails.

Regarding computation time, this simulation took 48.2 seconds (standard deviation of 0.7 seconds) when run on a heterogeneous mixture of twenty iMacs and Mac Pros. When run locally on a single quad-core Intel Xeon Mac Pro computer, this simulation took 592 seconds (standard deviation of 0.4 seconds).

Example 2: Fitting latent class models using add-on packages

Our second example is more realistic, as it involves simulations that would ordinarily take days to complete (as opposed to seconds for the one-sample t -test). It involves study of the properties of latent class models, which are used to determine better schemes for classification of eating disorders (Keel et al., 2004). The development of an empirically-created eating disorder classification system is of public health interest as it may help identify individuals who would benefit from diagnosis and treatment.

As described by Collins and Lanza (2009), latent class analysis (LCA) is used to identify subgroups in a population. There are several criteria used to evaluate the fit of a given model, including the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), the Consistent Akaike Information Criterion (cAIC), and the Sample Size Adjusted Bayesian Information Criterion (aBIC). These criteria are useful, but further guidance is needed for researchers to choose between them, as well as better understand how their accuracy is affected by methodological factors encountered in eating disorder classification research, such as unbalanced class size, sample size, missing data, and under- or over-specification of the model. Swanson et al. (2011) undertook a comprehensive review of these model cri-

teria, including a full simulation study to generate hypothetical data sets, and investigated how each criterion behaved in a variety of statistical environments. For this example, we replicate some of their simulations using an Apple Xgrid to speed up computation time.

Following the approach of Swanson et al., we generated “true models” where we specified an arbitrary four-class structure (with balanced number of observations in each class). This structure was composed of 10 binary indicators in a simulated data set of size 300. The model was fitted using the `poLCA()` function of the **poLCA** (polytomous latent class analysis) package (Linzer and Lewis, 2011). Separate latent class models were fitted specifying the number of classes, ranging from two to six. For each simulation, we determined the lowest values of BIC, AIC, cAIC and aBIC and recorded the class structure associated with that value.

Swanson and colleagues found that for this set of parameter values, the AIC and aBIC picked the correct number of classes more than half the time (see Table 2).

This example illustrates the computational burden of undertaking simulation studies to assess the performance of modern statistical methods, as several minutes are needed to undertake each of the single iterations of the simulation (which may explain why Swanson and colleagues only ran 100 simulations for each of their scenarios).

A complication of this example is that fitting LCA models in R requires the use of add-on packages. For instance, we use **poLCA** and its supporting packages with an Apple Xgrid to conduct our simulation. When a simulation requires add-on packages, they must be preloaded within the job and shipped over to the Xgrid. The specific steps of utilizing add-on packages are provided in Appendix B.

Simulation results

To better understand the potential performance gains when running simulations on the grid, we assessed the relative performance of a variety of grids with different characteristics and differing numbers of tasks per job (since there is some overhead to the

Grid description	ntask	numjobs	mean	sd
Mac Pro (1 processor)	1,000	1	43.98	0.27
Mac Pro (8 processors)	20	50	9.97	0.04
Mac Pro (8 processors)	10	100	9.65	0.05
Mac Pro (8 processors)	4	250	9.84	0.59
grid of 11 machines (66 processors)	1	1,000	1.02	0.003
grid of 11 machines (77 processors)	1	1,000	0.91	0.003
grid of 11 machines (88 processors)	20	50	1.28	0.01
grid of 11 machines (88 processors)	10	100	1.20	0.04
grid of 11 machines (88 processors)	8	125	1.04	0.03
grid of 11 machines (88 processors)	4	250	0.87	0.01
grid of 11 machines (88 processors)	1	1,000	0.87	0.004

Table 3: Mean and standard deviation of elapsed timing results (in hours) to run 1,000 simulations (each with three repetitions) using `poLCA()`, by grid and number of tasks per job. Note that the Mac Pro processors were rated at 3 GHz while the grid CPUs were rated at 2.8 GHz.

Apple Xgrid system). One grid was a single quad-core Intel Xeon Mac Pro computer (two quad-core processors, for total of 8 processors, each rated at 3 GHz, total 24 GHz). The other was a grid of eleven rack-mounted quad-core Intel servers (a total of 88 processors, each rated at 2.8 GHz, total 246.4 GHz). For comparison, we also ran the job directly within R on the Mac Pro computer (equivalent to `'ntask = 1000, numjobs = 1'`), as well as on the 88 processor grid but restricting the number of active jobs to 66 or 77 using the `throttle` option within `xgrid()`. For each setup, we varied the number of tasks (`ntask`) while maintaining 1,000 simulations for each scenario (`ntask * numjobs = 1000`).

For simulations with a relatively small number of tasks per job and a large number of jobs, the time to submit them to the grid controller can be measured in minutes; this overhead may be nontrivial. Also, for simulations on a quiescent grid where the number of jobs is not evenly divisible by the number of available processors, some part of the grid may be idle near the end of the simulation and the elapsed time longer than necessary.

Each simulation was repeated three times and the mean and standard deviation of total elapsed time (in hours) was recorded. Table 3 displays the results.

As expected, there was considerable speedup by using the grid. When moving from a single processor to 8, we were able to speed up our simulation by 34 hours – a 78% decrease in elapsed time $[(43.98 - 9.84)/43.98]$. We saw an even larger decrease of 98% when moving to 88 (somewhat slower) processors. There were only modest differences in the elapsed time comparing different configurations of the number of tasks per job and number of jobs, indicating that the overhead of the system imposes a relatively small cost. Once the job granularity is sufficiently fine, performance change is minimal.

Summary

We have described an implementation of a “grid stuffer” that can be used to access an Apple Xgrid from within R. Xgrid is an attractive platform for scientific computing because it can be easily set up, and it handles tedious housekeeping and collation of results from large simulations with relatively minimal effort. Core technologies in Mac OS X and easily available extensions simplify the process of creating a grid. An Xgrid may be configured with little knowledge of advanced networking technologies or disruption of networking activities.

Another attractive feature of this environment is that the Xgrid degrades gracefully. If an individual agent fails, then the controller will automatically resubmit the job to another agent. Once all of the submitted jobs are queued up on a controller, the `xgrid()` function can handle temporary controller failures (if the controller restarts then all pending and running jobs will be restarted automatically). If the client crashes, the entire simulation must be restarted. It may be feasible to keep track of this state to allow more fault tolerance in a future release.

A limitation of the Apple Xgrid system is that it is proprietary, which may limit its use. A second limitation is that it requires some effort on the part of the user to structure their program in a manner that can be divided into chunks then reassembled. Because of the relatively thin communication connections between the client and controller, the user is restricted to passing configuration via command line arguments in R, which must be parsed by the agent. Results are then passed back via the filesystem. In addition to the need for particular files to be created that can be run on the client as well as on the controller, use of the Apple Xgrid system may also require separate installation of additional packages needed for the simulation (particularly if the user does not have administrative access or control of the individual agents).

However, as demonstrated by our simulation studies, the Apple Xgrid system is particularly well-suited for embarrassingly parallel problems (e.g. simulation studies with multiple independent runs). It can be set up on a heterogeneous set of machines (such as a computer classroom) and configured to run when these agents are idle. For such a setup, the overall process load tends to balance if the simulation is broken up into sufficiently fine grained jobs, as faster agents will take on more than their share of jobs.

The Xgrid software is available as an additional download for Apple users as part of Mac OS X Server, and it is straightforward to set up a grid using existing computers (instructions are provided in Appendix C). In our tests, it took approximately 30 minutes to set up an Apple Xgrid on a simple network consisting of three computers.

Our implementation provides for three distinct authentication schemes (Kerberos, clear text password, or none). While the Xgrid system provides full support for authentication using Kerberos, this requires more configuration and is beyond the scope of this paper. The system documentation (Apple Inc., 2009) provides a comprehensive review of administration using this mechanism. If the `'auth = "Password"'` option is used, then the `XGRID_CONTROLLER_PASSWORD` environment variable must be set by the user to specify the password. As with any shared networking resource, care should be taken when less secure approaches are used to access the controller and agents. Firewalling the Xgrid controller and agents from outside networks may be warranted.

There are a number of areas of potential improvement for this interface. It may be feasible to create a single XML file to allow all of the simulations to be included as multiple tasks within a single job. This has the potential to decrease input/output load and simplify monitoring.

As described previously, the use of parallel computing to speed up scientific computation using R by use of multiple cores, tightly connected clusters, and other approaches remains an active area of research. Other approaches exist, though address different issues. For example, the **GridR** package does not support access through the `xgrid` command and requires access to individual agents. The **runjags** package provides an alternative interface that is particularly well suited for Bayesian estimation using Gibbs sampling.

Our setup focuses on a simple (but common) problem: embarrassingly parallel computations that can be chopped into multiple tasks or jobs. The ability to dramatically speed up such simulations within R by running them on a designated grid (or less formally on a set of idle machines) may be an attractive option in many settings with Apple computers.

Glossary

Agent A member of a grid that performs tasks distributed to it by the controller (also known as a node). An agent can be its own controller.

Apple Remote Desktop (ARD) An application that allows a user to monitor or control networked computers remotely.

Bonjour networking Apple's automatic network service discovery and configuration tool. Built on a variety of commonly used networking protocols, Bonjour allows Mac computers to easily communicate with each other and with a large number of compatible devices, with little to no user configuration required.

Client A computer that submits jobs to the controller for grid processing.

Controller A computer (running OS X Server or XgridLite) that accepts jobs from clients and distributes them to agents. A controller can also be an agent within the grid.

Embarrassingly parallel A computation that can be divided into a series of independent tasks where little or no communication is required.

Grid More general than an Apple Xgrid, a group of machines (agents), managed by a controller, which accept and perform computational tasks.

Grid stuffer An application that submit jobs and retrieves their results through the `xgrid` command. The `xgrid()` function within the **xgrid** package implements a grid stuffer in R.

Job A group of one or more tasks submitted by a client to the controller.

Mac OS X Server A Unix server operating system from Apple, that provides advanced features for operating an Xgrid controller (for those without Mac OS X Server, the XgridLite panel provides similar functionality for grid computing).

plist file Short for "property list file". On Mac OS computers, this is an XML file that stores user- or system-defined settings for an application or extension. For example, the "Preferences" pane in an application is linked to a plist file such that when Preferences are changed, their values are recorded in the plist file, to be read by other parts of the application when running.

Preference pane Any of the sections under the Mac OS X System Preferences, each controlling a different aspect of the operating system (desktop background, energy saving preferences, etc.) or of user-installed programs. XgridLite is a preference pane, since it is simply a GUI for

core system tools (such as the `xgridctl` command).

Processor core Independent processors in the same CPU. Most modern computers have multi-core CPUs, usually with 2 or 4 cores. Multi-core systems can execute more tasks simultaneously (in our case, multiple R runs at the same time). Multi-core systems are often able to manage tasks more efficiently, since less time is spent waiting for a core to be ready to process data.

Task Sub-components of jobs. A job may contain many tasks, each of which can be performed individually, or may depend on the results of other tasks or jobs.

Xgrid Software and distributed computing protocol developed by Apple that allows networked computers to contribute to distributed computations.

XgridLite Preference pane authored by Ed Baskerville. A free GUI to access Xgrid functionality that is built into all Mac OS X computers. Also facilitates configuration of controllers (similar functionality is provided by Mac OS X Server).

Appendix A: Accessing the Apple Xgrid using R

This section details the steps needed to access an existing Xgrid using R. All commands are executed on the client.

Install the package from CRAN

The first step is to install the package from CRAN on the client computer (needs to be done only once)

```
install.packages("xgrid")
```

Determine hostname and access information for the Apple Xgrid

The next step is to determine the access procedures for your particular grid. These include the hostname (specified as the `grid` option) and authorization scheme (specified as the `auth` option).

Create directory and file structure

An overview of the required directory structure is provided in Figure 5.

The 'input' folder contains two files, 'job.R' and 'runjob.R'. The file 'job.R' contains the definition for the function `job()`. This function expects two arguments: `ntask`, which specifies the number of tasks

per job, and `param`, used to define any parameter of interest. Note that `job()` must have `ntask` and `param` as arguments, but can be modified to take more. This function returns a data frame, where each row is the result of one task. The file 'runjob.R' (running on the agent) receives three arguments when called by `xgrid()` (running on the client): `ntask`, `param`, and `resfile` (see description of 'simulation.R' below). The arguments `ntask` and `param` are passed to `job()`, and the data frame returned by `job()` is saved to the folder 'output', with filename specified by `resfile`.

The folder 'output' will contain the results from the simulations. If created manually, it should be empty. If not created manually, the function `xgrid()` will create it.

The script 'simulation.R' accesses the controller by calling `xgrid()` and allows the user to override any default arguments.

Call `xgrid()` within 'simulation.R'

After loading the `xgrid` package and calling the `xgrid()` function as described in 'simulation.R', results from all `numsim` simulations are collated and returned as the object `res`. This object is also saved as the file 'RESULTS.rds', at the root of the directory structure. This can be analyzed as needed.

Appendix B: Using additional packages with the Apple Xgrid

One of the strengths of R is the large number of add-on packages that extend the system. If the user of the grid has administrative privileges on the individual machines then any needed packages can be installed in the usual manner.

However, if no administrative access is available, it is possible to utilize such packages within this setup by manually installing them in the 'input/rlibs' directory and loading them within a given job.

This appendix details how to make a package available to a job running on a given agent.

1. Install the appropriate distribution file from CRAN into the directory 'input/rlibs'. This can be done by using the `lib` argument of `install.packages()`. For instance, in Example 2 we can install `poLCA` by using the call `install.packages("poLCA", lib = "~/.../input/rlibs")`, where '~/...' emphasizes use of the absolute path to the 'rlibs' directory.
2. Access the add-on package within a job running on an agent. This is done by using the `lib.loc` option within the standard invocation of `library()`. For instance, in Example 2 this can be done by using the call `library(poLCA, lib.loc="./rlibs")`.

It should be noted that the package will need to be shipped over to the agent for each job, which may be less efficient than installing the package once per agent in the usual manner (but the latter option may not be available unless the grid user has administrative access to the individual agents).

Appendix C: Xgrid setup

In this section we discuss the steps required to set up a new Xgrid using XgridLite under Mac OS X 10.6 or 10.7. We presume some background in system administration and note that installation of a grid will likely require the assistance of technical staff.

Depending on the nature of the network and machines involved, different Xgrid-related tools can be used, including Mac OS X Server and XgridLite. Mac OS X Server has several advantages, but for the purposes of this paper, we restrict our attention to XgridLite.

Anatomy of an Apple Xgrid

It is important to understand the flow of information through an Apple Xgrid and the relationship between the different computers involved. The center-point of an Apple Xgrid is the controller (Figure 1). When clients submit jobs to the grid, the controller distributes them to agents, which make themselves available to the controller on the local network (using Apple's Bonjour network discovery). That is, rather than the controller keeping a master list of which agents are part of the grid, agents detect the presence of a controller on the network and, depending on their configuration, make themselves available for job processing.

This loosely coupled grid structure means that if an agent previously involved in computation is unavailable, the controller simply passes jobs to other agents. This functionality is known as graceful degradation. Once an agent has completed its computation, the results are made available to the controller, where the client can retrieve them. As a result, the client never communicates directly with agents, only with the controller.

Downloading XgridLite

XgridLite (<http://edbaskerville.com/software/xgridlite>) is free and open-source software, released under the GNU GPL (general public license), and installed as a Mac OS X preference pane. Once it is installed, it is found in the "Other" section (at the bottom) of the System Preferences (Apple menu → System Preferences).

Downloading server tools

Although not necessary to install and operate an Apple Xgrid, the free Apple software suite called Mac OS X Server Tools is extremely useful. In particular, the Xgrid Admin application (see Figure 6) allows for the monitoring of any number of Apple Xgrids, including statistics on the grid such as the number of agents and total processing power, as well as a list of jobs which can be paused, stopped, restarted, and deleted. Most "grid stuffers" (our system included) are designed to clean up after themselves, so the job management functions in Xgrid Admin are primarily useful for monitoring the status of jobs.

Setting up a controller

Figure 8 displays the process of setting up a grid controller within XgridLite. To start the Xgrid controller service, simply click the "Start" button. Authentication settings are configured with the "Set Client Password..." and "Set Agent Password..." buttons. From the XgridLite panel, you can also reset the controller or turn it off.

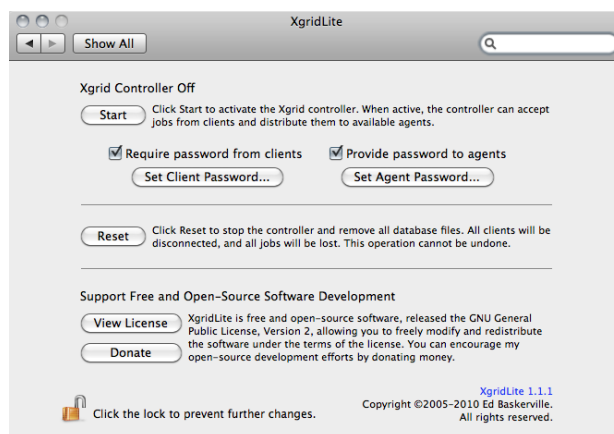


Figure 8: Setting up a controller

Unlike many other client-server relationships, the Xgrid controller authenticates to the client, rather than the other way around. That is, the password entered in the XgridLite "Set Agent Password..." field is *provided* to agents, not required of them. Individual agents must be configured to require that particular password (or none at all) in order to join the Xgrid.

Setting up agents

To set up an agent to join the Xgrid, open the Sharing preferences pane ("Internet & Wireless" section in OS 10.6) in System Preferences (see Figures 9 and 10). Select the "Xgrid Sharing" item in the list on the left. Click the "Configure..." button. From here, you can choose whether the agent should join the first available Xgrid, or a specific one. The latter of these

options is usually best – the drop-down menu of controllers will auto-populate with all that are currently running.

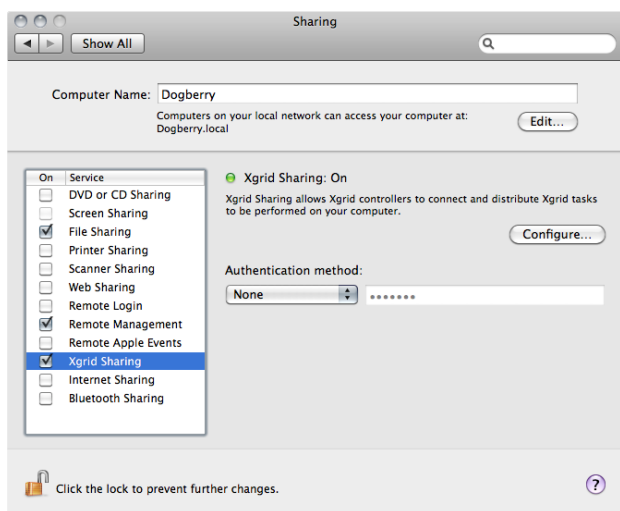


Figure 9: Sharing setup to configure an agent for Xgrid

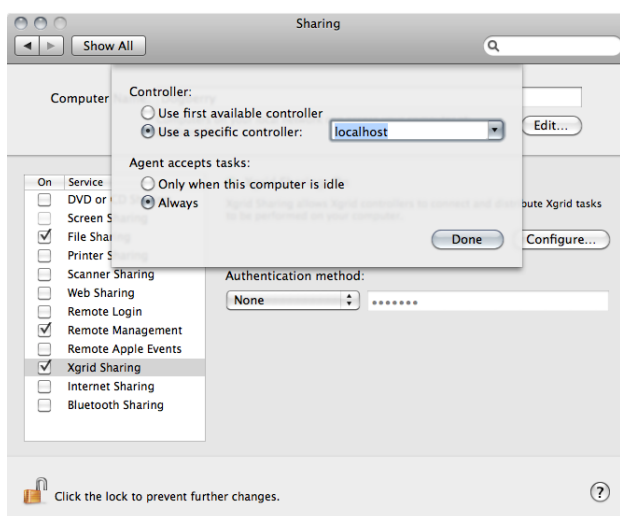


Figure 10: Specifying the controller for an agent

If there are multiple controllers on a network, you may want to choose one specifically so that the agent does not default to a different one. You may also choose whether or not the agent should accept tasks when idle. Then click “Done” and choose the desired authentication method (most likely “Password” or [less advisable] “None”) in the main window. If applicable, enter the password that will be required of the controller. Then, select the checkbox next to the “Xgrid Sharing” item on the left. The agent needs to be restarted to complete the process.

Note that a controller can also be an agent within its grid. However, it is important to consider that if the controller is also an agent, this has the potential to slow a simulation.

Automation

For those with more experience in system administration and access to Apple Remote Desktop (ARD) or secure shell (SSH) on the agent machines (or work with an administrator who does), the process of setting up agents can be automated. R can be installed remotely (allowing you to be sure that it is installed in the same location on each machine), the file that contains Xgrid agent settings can be pushed to all agents and, finally, the agent service can be activated. We will not discuss the ARD remote package installation process in detail here, but a full explanation can be found in the Apple Remote Desktop Administrator Guide at http://images.apple.com/remotedesktop/pdf/ARD_Admin_Guide_v3.3.pdf.

The settings for the Xgrid agent service are stored in the plist file located at `/Library/Preferences/com.apple.xgrid.agent.plist`. These settings include authentication options, controller binding, and the number of physical processor cores the agent should report to the controller (see the note below for a discussion of this value, which is significant). The simplest way to automate the creation of an Apple Xgrid is to configure the agent service on a specific computer with the desired settings and then push the above plist file to the same relative location on all agents. Keep the processor core settings in mind; you may need to push different versions of the file to different agents in a heterogeneous grid (see Note 3 below). Once all agents have the proper plist file, run the commands `xgridctl agent enable` and `xgridctl agent on` on the agents.

Notes

1. Mac OS X Server provides several additional options for setting up an Xgrid controller. These include Kerberos authentication and customized Xgrid cache files location. Depending on your network setup, these features may bolster security, but we will not explore them further here.
2. R **must** be installed in the same location on each agent. If it is installed locally by an end-user, it should default to the proper location, but it is worth verifying this before running tasks, otherwise they may fail. As mentioned previously, installing R remotely using ARD is an easy way to ensure an identical instantiation on each agent.
3. The reported number of processor cores (the `ProcessorCount` attribute) is significant because the controller considers it the maximum number of tasks the agent can execute simultaneously. On Mac OS X 10.6, agents by default report the number of cores (not processors) that they have (see discussion at <http://lists.>

apple.com/archives/Xgrid-users/2009/Oct/msg00001.html). The `ProcessorCount` attribute can be modified depending on the nature of the grid. Since R uses only 1 processor per core, this may leave many processors idle if the default behavior is not modified.

4. Because Xgrid jobs are executed as the user “nobody” on grid agents, they are given lower priority if the CPU is not idle.

Acknowledgements

Thanks to Tony Caldanaro and Randall Pruiem for technical assistance, as well as Molly Johnson, two anonymous reviewers and the associate editor for comments and useful suggestions on an earlier draft. This material is based in part upon work supported by the National Institute of Mental Health (5R01MH087786-02) and the US National Science Foundation (DUE-0920350, DMS-0721661, and DMS-0602110).

Bibliography

- Apple Inc. *Mac OS X Server: Xgrid Administration and High Performance Computing (Version 10.6 Snow Leopard)*. Apple Inc., 2009.
- L. M. Collins and S. T. Lanza. *Latent Class and Latent Transition Analysis: With Applications in the Social, Behavioral, and Health Sciences*. Wiley, 2009.
- M. J. Denwood. *runjags: Run Bayesian MCMC Models in the BUGS syntax from Within R*, 2010. URL <http://cran.r-project.org/web/packages/runjags/>. R package version 0.9.9-1.
- A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994. URL <http://www.csm.ornl.gov/pvm/>.
- T. Hesterberg. It's time to retire the $n \geq 30$ rule. *Proceedings of the Joint Statistical Meetings*, 2008. URL <http://home.comcast.net/~timhesterberg/articles/>.
- N. Horton and S. Anoke. *xgrid: Access Apple Xgrid using R*, 2012. URL <http://CRAN.R-project.org/package=xgrid>. R package version 0.2-5.
- P. K. Keel, M. Fichter, N. Quadflieg, C. M. Bulik, M. G. Baxter, L. Thornton, K. A. Halmi, A. S. Kaplan, M. Strober, D. B. Woodside, S. J. Crow, J. E. Mitchell, A. Rotondo, M. Mauri, G. Cassano, J. Treasure, D. Goldman, W. H. Berrettini, and W. H. Kaye. Application of a latent class analysis to empirically define eating disorder phenotypes. *Archives of General Psychiatry*, 61(2):192–200, February 2004.
- N. M. Li and A. J. Rossini. rpvm: Cluster statistical computing in R. *R News*, 1(3):4–7, 2001.
- D. A. Linzer and J. B. Lewis. poLCA: An R package for polytomous variable latent class analysis. *Journal of Statistical Software*, 42(10):1–29, 2011. URL <http://www.jstatsoft.org/v42/i10/>.
- A. J. Rossini, L. Tierney, and N. Li. Simple parallel statistical computing in R. *Journal of Computational and Graphical Statistics*, 16(2):399–420, 2007.
- S. S. Sawilowsky and R. C. Blair. A more realistic look at the robustness and Type II error properties of the t test to departures from population normality. *Psychological Bulletin*, 111(2):352–360, 1992.
- S. A. Swanson, K. Lindenberg, S. Bauer, and R. D. Crosby. A Monte Carlo investigation of factors influencing latent class analysis: An application to eating disorder research. *Journal of Abnormal Psychology*, 121(1):225–231, 2011.
- S. Urbanek. *multicore: Parallel processing of R code on machines with multiple cores or CPUs*, 2011. URL <http://CRAN.R-project.org/package=multicore>. R package version 0.1-7.
- D. Wegener, T. Sengstag, S. Sfakianakis, and A. A. S. Rüping. GridR: An R-based grid-enabled tool for data analysis in ACGT Clinico-Genomic Trials. *Proceedings of the 3rd International Conference on e-Science and Grid Computing (eScience 2007)*, 2007.
- B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 1999.
- H. Yu. Rmpi: Parallel statistical computing in R. *R News*, 2(2):10–14, 2002.

Sarah Anoke
Department of Mathematics and Statistics, Smith College
Clark Science Center, 44 College Lane
Northampton, MA 01063-0001 USA
sanoke527@gmail.com

Yuting Zhao
Department of Mathematics and Statistics, Smith College
Clark Science Center, 44 College Lane
Northampton, MA 01063-0001 USA
yuzhao@smith.edu

Rafael Jaeger
Brown University
Providence, RI 02912 USA
rafael_jaeger@brown.edu

Nicholas J. Horton
Department of Mathematics and Statistics, Smith College
Clark Science Center, 44 College Lane
Northampton, MA 01063-0001 USA
nhorton@smith.edu

maxent: An R Package for Low-memory Multinomial Logistic Regression with Support for Semi-automated Text Classification

by Timothy P. Jurka

Abstract `maxent` is a package with tools for data classification using multinomial logistic regression, also known as maximum entropy. The focus of this maximum entropy classifier is to minimize memory consumption on very large datasets, particularly sparse document-term matrices represented by the `tm` text mining package.

Introduction

The information era has provided political scientists with a wealth of data, yet along with this data has come the need to make sense of it all. Researchers have spent the past two decades manually classifying, or "coding" data according to their specifications—a monotonous task often assigned to undergraduate research assistants.

In recent years, supervised machine learning has become a boon in the social sciences, supplementing assistants with a computer that can classify documents with comparable accuracy. One of the main programming languages used for supervised learning in political science is R, which contains a plethora of machine learning add-ons via its package repository, CRAN.

Multinomial logistic regression

Multinomial logistic regression, or maximum entropy, has historically been a strong contender for text classification via supervised learning. When compared to the naive Bayes algorithm, a common benchmark for text classification, maximum entropy generally classifies documents with higher accuracy (Nigam, Lafferty, and McCallum, 1999).

Although packages for multinomial logistic regression exist on CRAN (e.g. `multinom` in package `nnet`, package `mlogit`), none handle the data using compressed sparse matrices, and most use a formula interface to define the predictor variables, approaches which tend to be inefficient. For most text classification applications, these approaches do not provide the level of optimization necessary to train and classify maximum entropy models using large

corpora. Typically, this problem manifests as the error "cannot allocate vector of size N" that terminates execution of the script.

Reducing memory consumption

`maxent` seeks to address this issue by utilizing an efficient C++ library based on an implementation written by Yoshimasa Tsuruoka at the University of Tokyo (Tsuruoka, 2011). Tsuruoka reduces memory consumption by using three efficient algorithms for parameter estimation: limited-memory Broyden-Fletcher-Goldfarb-Shanno method (L-BFGS), orthant-wise limited-memory quasi-newton optimization (OWLQN), and stochastic gradient descent optimization (SGD). Typically maximum entropy models grow rapidly in size when trained on large text corpora, and minimizing the number of parameters in the model is key to reducing memory consumption. In this respect, these algorithms outperform alternative optimization techniques such as the conjugate gradient method (Nocedal, 1990). Furthermore, the SGD algorithm provides particularly efficient parameter estimation when dealing with large-scale learning problems (Bottou and Bousquet, 2008)

After modifying the C++ implementation to make it work with R and creating an interface to its functions using the `Rcpp` package (Eddelbuettel and Francois, 2011), the `maxent` package yielded impressively efficient memory usage on large corpora created by `tm` (Feinerer, Hornik, and Meyer, 2008), as can be seen in Figure 1. Additionally, `maxent` includes the `as.compressed.matrix()` function for converting various matrix representations including "DocumentTermMatrix" found in the `tm` package, "Matrix" found in the `Matrix` package, and "simple_triplet_matrix" found in package `slam` into the "matrix.csr" representation from package `SparseM` (Koenker and Ng, 2011). In the case of document-term matrices generated by `tm`, this function eliminates the intermediate step of converting the "DocumentTermMatrix" class to a standard "matrix", which consumes significant amounts of memory when representing large datasets. Memory consumption was measured using a shell script that tracked the peak memory usage reported by the UNIX `top` command during execution of the

`maxent()` and `predict()` functions in a sterile R environment.

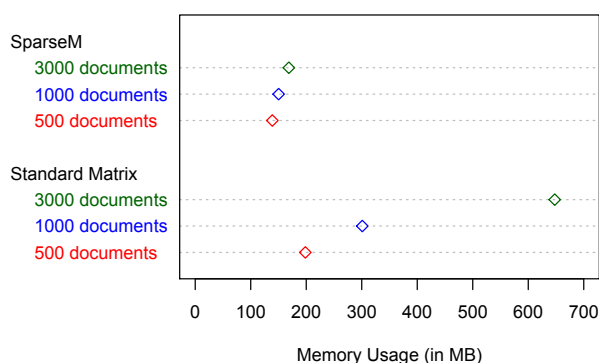


Figure 1: Memory consumption of a **SparseM** "matrix.csr" representation compared with a standard "matrix" representation for 500, 1000, and 3000 documents from *The New York Times* dataset.

Furthermore, the standard "DocumentTermMatrix" represents i indices as repeating values, which needlessly consumes memory. The i indices link a document to its features via an intermediate j index. In the example below, an eight-digit series of 1's signifies that the first eight j indices correspond to the first document in the "DocumentTermMatrix".

```
> matrix$i
1 1 1 1 1 1 1 2 2 2 2 2 2 3
3 3 3 3 3 3 3 4 4 4 4 4 4 4
4 4 4 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 7 7 7 7 7 8 8 8 8
8 8
```

Conversely, the **SparseM** package used in **maxent** consolidates repeating values into a range of values. In the next example, j indices 1 through 8 correspond to the first document, 9 through 14 to the second document, and so forth. This removes excess integers in the "DocumentTermMatrix" representation of the sparse triplet matrix, further reducing the memory footprint of the sparse matrix representation. The matrix representation below has been reduced from 62 integers down to 9, or approximately 1/7th the original size.

```
> sparse@ia
1 9 15 24 34 46 52 57 63
```

Example

We read in our dataset using input/output functions, in this case `read.csv()`. The `NYTimes.csv.gz` dataset is bundled with the **maxent** package, and contains 4000 manually coded headlines from *The New York Times* (Boydston, 2008).

```
data <- read.csv(
```

```
system.file("data/NYTimes.csv.gz",
package = "maxent"))
```

Next, we use **tm** to transform our data into a corpus, and subsequently a "DocumentTermMatrix" or its transpose, the "TermDocumentMatrix". We then convert our "DocumentTermMatrix" into a compressed "matrix.csr" representation using the `as.compressed.matrix()` function.

```
corpus <- Corpus(VectorSource(data$Title))
matrix <- DocumentTermMatrix(corpus)
sparse <- as.compressed.matrix(matrix)
```

We are now able to specify a training set and a classification set of data, and start the supervised learning process using the `maxent()` and `predict()` functions. In this case we have specified a training set of 1000 documents and a classification set of 500 documents.

```
model <- maxent(sparse[1:1000,],
               data$Topic.Code[1:1000])
results <- predict(model,
                  sparse[1001:1500,])
```

Although **maxent**'s primary appeal is to those users needing low-memory multinomial logistic regression, the process can also be run using standard matrices. However, the input matrix must be an `as.matrix()` representation of a "DocumentTermMatrix"—class "TermDocumentMatrix" is not currently supported.

```
model <- maxent(as.matrix(matrix)[1:1000,],
               as.factor(data$Topic.Code)[1:1000])
results <- predict(model,
                  as.matrix(matrix)[1001:1500,])
```

To save time during the training process, we can save our model to a file using the `save.model()` function. The saved maximum entropy model is accessible using the `load.model()` function.

```
model <- maxent(sparse[1:1000,],
               data$Topic.Code[1:1000])
save.model(model, "myModel")
saved_model <- load.model("myModel")
results <- predict(saved_model,
                  sparse[1001:1500,])
```

Algorithm performance

In political science, supervised learning is heavily used to categorize textual data such as legislation, television transcripts, and survey responses. To demonstrate the accuracy of the **maxent** package in a research setting, we will train the multinomial logistic regression classifier using 4,000 summaries of legislation proposed in the United States Congress and then test accuracy on 400 new summaries. The summaries have been manually partitioned into 20 topic codes corresponding to policy issues such as health,

agriculture, education, and defense (Adler and Wilkerson, 2004).

We begin by loading the `maxent` package and reading in the bundled `USCongress.csv.gz` dataset.

```
data <- read.csv(
  system.file("data/USCongress.csv.gz",
    package = "maxent"))
```

Using the built-in `sample()` function in R and the `tm` text mining package, we then randomly sample 4,400 summaries and generate a "TermDocumentMatrix". To reduce noise in the data, we make all characters lowercase and remove punctuation, numbers, stop words, and whitespace.

```
indices <- sample(1:4449, 4400, replace=FALSE)
sample <- data[indices,]
vector <- VectorSource(as.vector(sample$text))
corpus <- Corpus(vector)
matrix <- TermDocumentMatrix(corpus,
  control=list(weighting = weightTfIdf,
    language = "english",
    tolower = TRUE,
    stopwords = TRUE,
    removeNumbers = TRUE,
    removePunctuation = TRUE,
    stripWhitespace = TRUE))
```

Next, we pass this matrix into the `maxent()` function to train the classifier, partitioning the first 4,000 summaries as the training data. Similarly, we partition the last 400 summaries as the new data to be classified. In this case, we use stochastic gradient descent for parameter estimation and set a held-out sample of 400 summaries to account for model overfitting.

```
model <- maxent(matrix[,1:4000],
  sample$major[1:4000],
  use_sgd = TRUE,
  set_heldout = 400)
results <- predict(model, matrix[,4001:4400])
```

When we compare the predicted value of the results to the true value assigned by a human coder, we find that 80% of the summaries are accurately classified. Considering we only used 4,000 training samples and 20 labels, this result is impressive, and the classifier could certainly be used to supplement human coders. Furthermore, the algorithm is very fast; `maxent` uses only 135.4 megabytes of RAM and finishes in 53.3 seconds.

Model tuning

The `maxent` package also includes the `maxent.tune()` function to determine parameters that will maximize the recall of the results during the training process. The function tests the algorithm

across a sample space of 18 parameter configurations including varying the regularization terms for L1 and L2 regularization, using SGD optimization, and setting a held-out portion of the data to detect overfitting.

We can use the tuning function to optimize a model that will predict the species of an iris given the dimensions of petals and sepals. Using Anderson's Iris data set that is bundled with R, we will use n-fold cross-validation to test the model using several parameter configurations.

```
data(iris)

x <- subset(iris, select = -Species)
y <- iris$Species

f <- tune.maxent(x, y, nfold=3, showall=TRUE)
```

L1	L2	SGD	Held-out	Accuracy	Pct. Fit
0	0	0	0	0.948	0.975
0.2	0	0	0	0.666	0.685
0.4	0	0	0	0.666	0.685
0.6	0	0	0	0.666	0.685
0.8	0	0	0	0.666	0.685
1	0	0	0	0.666	0.685
0	0	0	25	0.948	0.975
0	0.2	0	0	0.966	0.993
0	0.4	0	0	0.966	0.993
0	0.6	0	0	0.972	1
0	0.8	0	0	0.972	1
0	1	0	0	0.966	0.993
0	0	1	0	0.966	0.993
0.2	0	1	0	0.666	0.685
0.4	0	1	0	0.666	0.685
0.6	0	1	0	0.666	0.685
0.8	0	1	0	0.666	0.685
1	0	1	0	0.666	0.685

Table 1: Output of the `maxent.tune()` function when run on Anderson's Iris data set, with each row corresponding to a unique parameter configuration. The first two columns are L1/L2 regularization parameters bounded by 0 and 1. SGD is a binary variable indicating whether stochastic gradient descent should be used. The fourth column represents the number of training samples held-out to test for overfitting. Accuracy is measured via n-fold cross-validation and percent fit is the accuracy of the configuration relative to the optimal parameter configuration.

As can be seen in Table 1, the results of the tuning function show that using L2 regularization will provide the best results, but using stochastic gradient descent without any regularization will perform nearly as well. We can also see that we would not want to use L1 regularization for this application as

it drastically reduces accuracy. The held-out parameter is used without L1/L2 regularization and SGD because tests showed a decrease in accuracy when using them together. However, when L1/L2 regularization and SGD do not yield an increase in accuracy, using held-out data can improve performance. Using this information, we can improve the accuracy of our classifier without adding or manipulating the training data.

Summary

The `maxent` package provides a fast, low-memory maximum entropy classifier for a variety of classification tasks including text classification and natural language processing. The package integrates directly into the popular `tm` text mining package, and provides sample datasets and code to get started quickly. Additionally, a number of advanced features are available to prevent model overfitting and provide more accurate results.

Acknowledgements

This project was made possible through financial support from the University of California at Davis, University of Antwerp, and Sciences Po Bordeaux. I am indebted to Professor Yoshimasa Tsuruoka for taking the time to answer questions about his excellent C++ maximum entropy implementation, as well as Professor Amber E. Boydstun for her *New York Times* dataset and comments on this paper. This paper also benefited from the feedback of two anonymous reviewers.

Bibliography

- E. Adler, J. Wilkerson. *Congressional Bills Project*. University of Washington, 2004. URL <http://www.congressionalbills.org/>.
- L. Bottou, O. Bousquet. The tradeoffs of large scale learning. *Advances in Neural Information Processing Systems*, pp. 161–168, 2008. URL <http://leon.bottou.org/publications/pdf/nips-2007.pdf>.
- A. Boydstun. How Policy Issues Become Front-Page News. *Under review*. URL <http://psfaculty.ucdavis.edu/boydstun/>.
- D. Eddelbuettel, R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8), pp. 1–18, 2011. URL <http://www.jstatsoft.org/v40/i08/>.
- I. Feinerer, K. Hornik, D. Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25(5), pp. 1–54, 2008. URL <http://www.jstatsoft.org/v25/i05/>.
- R. Koenker, P. Ng. SparseM: A Sparse Matrix Package for R. *CRAN Package Archive*, 2011. URL <http://cran.r-project.org/package=SparseM>.
- R. Nocedal. The performance of several algorithms for large scale unconstrained optimization. *Large Scale Numerical Optimization*, pp. 138–151, 1990. Society for Industrial & Applied Mathematics.
- K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. *IJCAI-99 Workshop on Machine Learning for Information Filtering*, pp. 61–67, 1999. URL <http://www.kamalnigam.com/papers/maxent-ijcaiws99.pdf>.
- Y. Tsuruoka. A simple C++ library for maximum entropy classification. *University of Tokyo Department of Computer Science (Tsujii Laboratory)*, 2011. URL <http://www-tsujii.is.s.u-tokyo.ac.jp/~tsuruoka/maxent/>.

Timothy P. Jurka
 Department of Political Science
 University of California, Davis
 Davis, CA 95616-8682 USA
 tpjurka@ucdavis.edu

Sumo: An Authenticating Web Application with an Embedded R Session

by Timothy T. Bergsma and Michael S. Smith

Abstract Sumo is a web application intended as a template for developers. It is distributed as a Java ‘war’ file that deploys automatically when placed in a Servlet container’s ‘webapps’ directory. If a user supplies proper credentials, Sumo creates a session-specific Secure Shell connection to the host and a user-specific R session over that connection. Developers may write dynamic server pages that make use of the persistent R session and user-specific file space. The supplied example plots a data set conditional on preferences indicated by the user; it also displays some static text. A companion server page allows the user to interact directly with the R session. Sumo’s novel feature set complements previous efforts to supply R functionality over the internet.

Despite longstanding interest in delivering R functionality over the internet (Hornik, 2011), embedding R in a web application can still be technically challenging. Anonymity of internet users is a compounding problem, complicating the assignment of persistent sessions and file space. Sumo uses Java servlet technology to minimize administrative burden and requires authentication to allow unambiguous assignment of resources. Sumo has few dependencies and may be readily adapted to create other applications.

Deployment

Use of Sumo requires an internet server with the following:

- users with passworded accounts (Sumo defers to the server for all authentication)
- a system-wide installation of R (<http://cran.r-project.org>)
- a running instance of a Java servlet container, e.g. Tomcat 6 (<http://tomcat.apache.org>)
- Secure Shell¹, e.g. OpenSSH (<http://www.openssh.com>) or freeSSHd (<http://www.freesshd.com>) for Windows.

Tomcat, the reference implementation for the Java Servlet and Java Server Pages (JSP) specifications,

¹OS X includes OpenSSH. Check the box: System Preferences | Sharing | remote login. In ‘/etc/sshd_config’, add “PasswordAuthentication yes”. With freeSSHd for Windows, uncheck: settings | SSH | Use new console engine.

²Host is configurable in the file ‘sumo.xml’.

³Command to start R is configurable in the file ‘sumo.xml’.

is widely available and surprisingly easy to install. While frequently run behind Apache (<http://www.apache.org>), it can function as a stand-alone web server. In our experience, it requires mere minutes to install under Linux (Ubuntu) or Mac OS X, and a little longer to install for Windows. Tomcat is open source.

Sumo is maintained at <http://sumo.googlecode.com> and is distributed under the GPL v3 open source license as a Java ‘war’ file. The current distribution, considered complete, is ‘sumo-007.war’ as of this writing. It can be downloaded, optionally renamed (e.g. ‘sumo.war’), and saved in Tomcat’s ‘webapps’ directory. Tomcat discovers, installs, and launches the web application. By default, Tomcat listens on port 8080. The application can be accessed with a web browser, e.g. at <http://localhost:8080/sumo>.

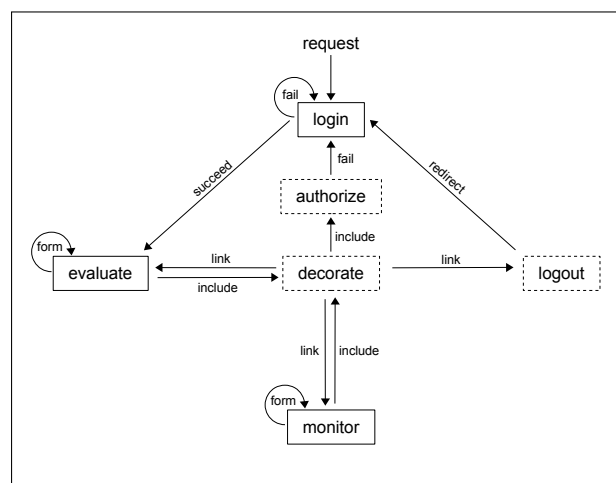


Figure 1: Sumo architecture: flow control among page requests. Solid boxes: visible JSPs. Dashed boxes: JSPs with no independent content.

Architecture

An http request for the web application Sumo defaults to the ‘login.jsp’ page (Figure 1). Tomcat creates a Java servlet session for the particular connection. ‘login.jsp’ forwards the supplied credentials to a servlet (not shown) that tries to connect to the local host² using Secure Shell. If successful, it immediately opens an R session using `R --vanilla`³, stores pointers to the R session, and forwards the user to ‘evaluate.jsp’: the core of the application (see Figure 2). The only other viewable page is ‘monitor.jsp’,

which allows direct interaction with the R command line (Figure 3).

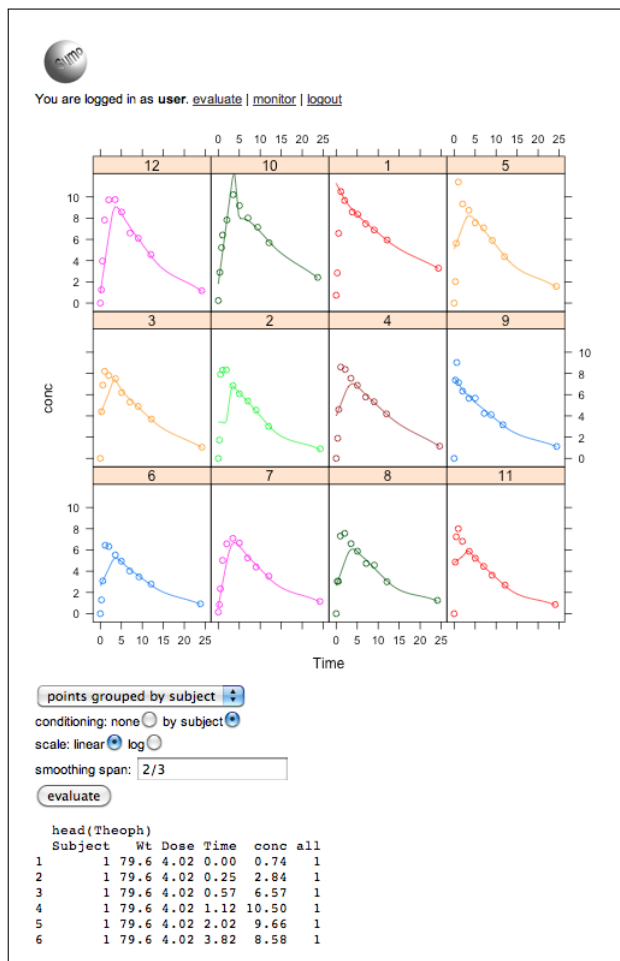


Figure 2: Sumo display: sample output of 'evaluate.jsp'.

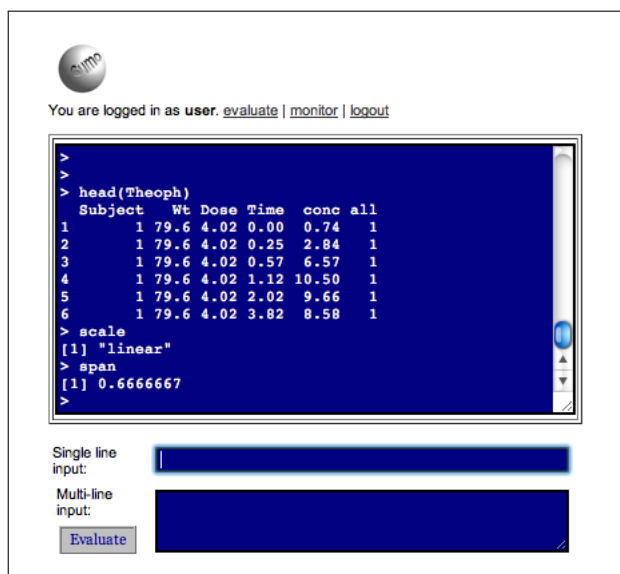


Figure 3: Sumo display: sample output of 'monitor.jsp'.

Both 'evaluate.jsp' and 'monitor.jsp' use optional Sitemesh technology (<http://www.sitemesh.org>) to include 'decorate.jsp', which itself includes the header, the footer, and navigation links to 'evaluate.jsp', 'monitor.jsp', and 'logout.jsp'. 'decorate.jsp' also includes 'authorize.jsp', which redirects to 'login.jsp' for non-authenticated requests. Thus decorated, 'evaluate.jsp' and 'monitor.jsp' always include links to each other and are only available to authenticated users.

The key to embedding R in Sumo is the exchange of information between the Java session and the R session. The exchange occurs primarily in 'evaluate.jsp'. Similar to Sweave (Leisch, 2002), Java Server Pages support alternation between two languages: in this case, html and Java. The R session can be accessed using either language. With respect to Java, 'evaluate.jsp' defines a reference to an object called R that evaluates R code and returns a string result (with or without echo).

```

<%
String directory = R.evaluate(
  "writeLines(getwd())", false);
%>
    
```

With respect to html, 'evaluate.jsp' includes Sumo's RTags library (a JSP Tag Library), which allows blocks of R code to be inserted directly within html-like tags.

```

<%@
taglib prefix="r" uri="/WEB-INF/RTags.tld"
%>
<r:R silent="true">
  library(lattice)
  Theoph$all <- 1
</r:R>

<pre>
<r:R>
  head(Theoph)
</r:R>
</pre>
    
```

Also included in 'evaluate.jsp' is 'params2r.jsp', which assures that parameters supplied by the user are automatically defined in the R session as either numeric (if possible) or character objects.

Note that 'evaluate.jsp' has a single web form (Figure 2) whose target is 'evaluate.jsp' itself. Interaction with a user consists of repeated calls to the page, with possibly modified parameters. The example given illustrates table and figure display, emphasizing a variety of input styles (pull-down, radio button, text box). The data frame Theoph is plottable eight different ways (grouped or ungrouped, conditioned or unconditioned, log or linear) with an adjustable smoothing span. Internally, 'evaluate.jsp' follows a well-defined sequence:

1. Capture any parameters specified by the user.

2. Supply defaults for parameters not user-specified.
3. Echo parameters to the R session as necessary.
4. Create and display a figure.
5. Present a form with current parameters as the defaults.

Additionally, static text is included to illustrate a very simple way of presenting tabular data.

Development

Sumo is intended as a template for web application developers. There are at least three ways to adapt Sumo. For experimental purposes, one can edit deployed JSPs in place (see ‘webapps/sumo/’); Tomcat notices the changes and recompiles corresponding servlets. For better source control, one can unzip ‘sumo.war’⁴, edit files of interest, re-zip and redeploy. For a formal development environment, one can install Apache Ant (<http://ant.apache.org>) and check out Sumo’s source:

```
svn checkout
http://sumo.googlecode.com/svn/trunk/ sumo
```

Then at the command prompt in the ‘sumo’ directory⁵ use:

```
ant -f build.xml
```

Security

Sumo inherits most of its security attributes from the infrastructure on which it builds. Since the R session is created with a Secure Shell connection, it has exactly the file access configured at the machine level for the particular user. The JSPs and related servlets, which are not user-modifiable, verify the user before proceeding. ‘SpecificImageServlet’ (called from ‘evaluate.jsp’) can be used to request arbitrary files, but it defers to the R session’s verdict on read permission. Over http, login credentials may be vulnerable to discovery. Where necessary, administrators may configure Tomcat or Apache to use the secure http protocol (i.e. https).

Persistence

While it is possible to evaluate R expressions in isolation, more functionality is available if results can persist in session memory or in hardware memory. Sumo uses both forms of persistence. A session is created for each user at login, and persists across

page requests until logout. Files created on behalf of the user are stored in the server file space associated the user’s account. The application developer can take advantage of both.

In some cases, it is possible to create R sessions that persist, not just *within* server sessions, but also *across* them. On systems supporting ‘screen’ (e.g. Linux, OS X), the command used to start R can be configured in ‘sumo.xml’ as

```
screen -d -r || screen R -- vanilla
```

An R session will be attached if one exists, else a new one will be created. On logout, the R session persists, and is reattached at next login.

Discussion

While Sumo is not the first attempt at an R-enabled web application, it offers simple deployment and standard architecture. Deployment consists of dropping the Sumo archive into the ‘webapps’ directory of a running instance of Tomcat. Development can be as simple as editing the resulting text files, even while the application is running.

Like **Rpad** (Short and Grosjean, 2007), Sumo accesses R functionality through the user’s web browser. **Rpad** is arguably easier to deploy on a local machine, requiring only the **Rpad** package, but Sumo is easier to deploy on a server, as it requires no customization of Tomcat or Apache.

Relative to Sumo, rApache (Horner, 2012b) is a more sophisticated solution for the development of R-enabled web applications. It uses the Apache web server rather than Tomcat. Sumo is an example application, whereas rApache is a module. As such, rApache remains generally unprejudiced with respect to design choices like those for authentication and persistence. Like **Rpad**, rApache requires server configuration. rApache is currently only available for Linux and Mac OS X, whereas Tomcat (and therefore Sumo) is available for those platforms as well as Windows.

Of broader utility than just for web pages, the packages **brew** (Horner, 2011) and **R.rsp** (Bengtsson, 2012) independently provide functions for text pre-processing. Expressions within delimiters are evaluated in R, substituting the resulting values into the original text. The delimiters are similar or identical to those used in Java Server Pages. Sumo retains such delimiters to evaluate Java expressions and introduces alternative delimiters to evaluate R expressions.

Rook (Horner, 2012a) is a specification that defines an interface between a web server and R. Applications written against the specification can be run unmodified with any supporting server, such as R’s

⁴‘war’ files have ‘zip’ file architecture.

⁵The value of `tomcat.dir` in ‘build.xml’ may need modification.

internal web server or an rApache instance. As a convenience, **Rook** also supplies classes for writing Rook-compliant applications and for working with R's built-in web server. For comparison, Sumo is compliant with the Java Servlet specification and runs on supporting servers such as Tomcat.

Rserve (Urbanek, 2003) can be adapted for web applications, but is intended for applications where bandwidth is more important and security less so, relative to Sumo.

While Sumo does provide some access to the R command line, those interested primarily in running R remotely should favor the server version of RStudio (<http://rstudio.org>). RStudio provides some graphical interaction via the **manipulate** package, but the technique does not seem easy to formalize for an independent application.

Display of text is only briefly treated by Sumo; **R2HTML** (Lecoutre, 2003) seems a natural complement.

Summary

Sumo is a web application designed to be easily deployed and modified. It relies on industry-standard software, such as Tomcat and Secure Shell, to enlist the web browser as an interface to R functionality. It has few dependencies, few limitations, good persistence and good security. Enforced authentication of users allows unambiguous assignment of sessions and file space for full-featured use of R. The learning curve is shallow: while some Java is required, most interested parties will already know enough html and R to produce derivative applications quickly.

Acknowledgments

We thank Henrik Bengtsson, Jeffrey Horner, James Rogers, Joseph Hebert, and an anonymous reviewer for valuable comments.

Development and maintenance of Sumo is funded by Metrum Research Group LLC (<http://www.metrumrg.com>).

Bibliography

H. Bengtsson. *R.rsp: Dynamic generation of scientific reports*, 2012. URL <http://CRAN.R-project.org/package=R.rsp>. R package version 0.7.1.

J. Horner. *brew: Templating framework for report generation*, 2011. URL <http://CRAN.R-project.org/package=brew>. R package version 1.0-6.

J. Horner. *Rook: a web server interface for R*, 2012a. URL <http://CRAN.R-project.org/package=Rook>. R package version 1.0-3.

J. Horner. *rApache: Web application development with R and Apache*, 2012b. URL <http://www.rapache.net>.

K. Hornik. R FAQ: Frequently asked questions on R, 2011. URL <http://cran.r-project.org/doc/FAQ/R-FAQ.html#R-Web-Interfaces>. Version 2.13.2011-07-05, ISBN 3-900051-08-9.

E. Lecoutre. The R2HTML package. *R News*, 3(3):33–36, December 2003. URL http://cran.r-project.org/doc/Rnews/Rnews_2003-3.pdf.

F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. URL <http://www.stat.uni-muenchen.de/~leisch/Sweave>. ISBN 3-7908-1517-9.

T. Short and P. Grosjean. *Rpad: Workbook-style, web-based interface to R*, 2007. URL <http://www.rpad.org/Rpad>. R package version 1.3.0.

S. Urbanek. Rserve: A fast way to provide R functionality to applications. In K. Hornik, F. Leisch, and A. Zeileis, editors, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, Vienna, Austria, March 20-22 2003. URL <http://www.ci.tuwien.ac.at/Conferences/DSC-2003/>. ISSN 1609-395X.

Timothy T. Bergsma
Metrum Research Group LLC
2 Tunxis Road Suite 112
Tariffville CT 06081
USA
timb@metrumrg.com

Michael S. Smith
Consultant
26 Harris Fuller Road
Preston, CT 06365
USA
michael_s_smith@yahoo.com

Who Did What? The Roles of R Package Authors and How to Refer to Them

by Kurt Hornik, Duncan Murdoch and Achim Zeileis

Abstract Computational infrastructure for representing persons and citations has been available in R for several years, but has been restructured through enhanced classes "person" and "bibentry" in recent versions of R. The new features include support for the specification of the roles of package authors (e.g. maintainer, author, contributor, translator, etc.) and more flexible formatting/printing tools among various other improvements. Here, we introduce the new classes and their methods and indicate how this functionality is employed in the management of R packages. Specifically, we show how the authors of R packages can be specified along with their roles in package 'DESCRIPTION' and/or 'CITATION' files and the citations produced from it.

R packages are the result of scholarly activity and as such constitute scholarly resources which must be clearly identifiable for the respective scientific communities and, more generally, today's information society. In particular, packages published by standard repositories can be regarded as *reliable sources* which can and should be referenced (i.e. cited) by scientific works such as articles or other packages. This requires conceptual frameworks and computational infrastructure for describing bibliographic resources, general enough to encompass the needs of communities with an interest in R. These needs include support for exporting bibliographic metadata in standardized formats such as `BIBTEX` (Berry and Patashnik, 2010), but also facilitating bibliometric analyses and investigations of the social fabric underlying the creation of scholarly knowledge.

The latter requires a richer vocabulary than commonly employed by reference management software such as `BIBTEX`, identifying *persons* and their *roles* in relation to bibliographic resources. For example, a thesis typically has an author and advisors. Software can have an (original) author and a translator to another language (such as from S to R). The maintainer of an R package is not necessarily an author.

In this paper, we introduce the base R infrastructure (as completely available in R since version 2.14.0) for representing and manipulating such scholarly data: objects of class "person" (hereafter, *person objects*) hold information about persons, possibly including their roles; objects of class "bibentry" (hereafter, *bibentry objects*) hold bibliographic information in enhanced `BIBTEX` style, ideally using person objects when referring to persons (such as au-

thors or editors). Furthermore, we indicate how this functionality is employed in the management of R packages, in particular in their 'CITATION' and 'DESCRIPTION' files.

Persons and their roles

Person objects can hold information about an arbitrary positive number of persons. These can be obtained by one call to `person()` with list arguments, or by first creating objects representing single persons and combining these via `c()`.

Every person has at least one *given* name, and natural persons typically (but not necessarily, Wikipedia, 2012) have a *family* name. These names are specified by arguments `given` and `family`, respectively. For backward compatibility with older versions, there are also arguments `first`, `middle` and `last` with obvious (Western-centric) meaning; however, these are deprecated as not universally applicable: some cultures place the given after the family name, or use no family name.

An optional `email` argument allows specifying persons' email addresses, which may serve as unique identifiers for the persons.

Whereas person names by nature may be arbitrary non-empty character strings, the state of the art in library science is to employ standardized vocabularies for person roles. R uses the "Relator and Role" codes and terms (<http://www.loc.gov/marc/relators/relaterm.html>) from MARC (MACHINE-Readable Cataloging, Library of Congress, 2012), one of the leading collections of standards used by librarians to encode and share bibliographic information. Argument `role` specifies the roles using the MARC codes; if necessary, the `comment` argument can be used to provide (human-readable) character strings complementing the standardized role information.

As an example (see Figure 1 and also `?person`), consider specifying the authors of package `boot` (Canty and Ripley, 2012): Angelo Canty is the first author ("aut") who wrote the S original; Brian D. Ripley is the second author ("aut"), who translated the code to R ("trl"), and maintains the package ("cre").

Note that the MARC definition of the relator term "Creator" is "Use for a person or organization responsible for the intellectual or artistic content of a work.", which maps nicely to the notion of the maintainer of an R package: the maintainer creates the package as a bibliographic resource for possible redistribution (e.g. via the CRAN package repository), and is the person (currently) responsible for

```
R> ## specifying the authors of the "boot" package
R> p <- c(
+   person("Angelo", "Canty", role = "aut", comment =
+     "S original, http://statwww.epfl.ch/davison/BMA/library.html"),
+   person(c("Brian", "D."), "Ripley", role = c("aut", "trl", "cre"),
+     comment = "R port, author of parallel support", email = "ripley@stats.ox.ac.uk")
R> p

[1] "Angelo Canty [aut] (S original, http://statwww.epfl.ch/davison/BMA/library.html)"
[2] "Brian D. Ripley <ripley@stats.ox.ac.uk> [aut, trl, cre] (R port, author of parallel support)"

R> ## subsetting
R> p[-1]

[1] "Brian D. Ripley <ripley@stats.ox.ac.uk> [aut, trl, cre] (R port, author of parallel support)"

R> ## extracting information (as list for multiple persons)
R> p$given

[[1]]
[1] "Angelo"

[[2]]
[1] "Brian" "D."

R> ## extracting information (as vector for single person)
R> p[2]$given

[1] "Brian" "D."

R> ## flexible formatting
R> format(p, include = c("family", "given", "role"),
+   braces = list(family = c("", "", " "), role = c("[Role(s): ", " ]")),
+   collapse = list(role = "/"))

[1] "Canty, Angelo [Role(s): aut]"          "Ripley, Brian D. [Role(s): aut/trl/cre]"

R> ## formatting for BibTeX
R> toBibtex(p)

[1] "Angelo Canty and Brian D. Ripley"

R> ## alternative BibTeX formatting "by hand"
R> paste(format(p, include = c("family", "given"),
+   braces = list(family = c("", "", " "))), collapse = " and ")

[1] "Canty, Angelo and Ripley, Brian D."
```

Figure 1: Example for `person()`, specifying the authors of the `boot` package.

the package, being the “interface” between what is inside the package and the outside world, and warranting in particular that the license and representation of intellectual property rights are appropriate. For the persons who originally “created” the package code, typically author roles should be employed.

In general, while all MARC relator codes are supported, the following usage is suggested when giving the roles of persons in the context of authoring R packages:

"aut" (Author): Full authors who have made substantial contributions to the package and should show up in the package citation.

"com" (Compiler): Persons who collected code (potentially in other languages) but did not make further substantial contributions to the package.

"ctb" (Contributor): Authors who have made smaller contributions (such as code patches

etc.) but should not show up in the package citation.

"cph" (Copyright holder): Copyright holders.

"cre" (Creator): Package maintainer.

"ths" (Thesis advisor): Thesis advisor, if the package is part of a thesis.

"trl" (Translator): Translator to R, if the R code is a translation from another language (typically S).

Person objects can be subscripted by field (using `$`) or by position (using `[]`), and allow for flexible formatting and printing. By default, email, role and comment entries are suitably “embraced” by enclosing with angle brackets, square brackets and parentheses, respectively, and suitably collapsed. There is also a `toBibtex()` method which creates a suitable $\text{BIB}\text{\TeX}$ representation. Finally, the default method for `as.person()` is capable of inverting the default person formatting, and also can handle formatted person entries collapsed by comma or ‘and’ (with appropriate white space). See Figure 1 and `?person` for various illustrative usages of these methods.

R code specifying person objects can also be used in the `Authors@R` field of ‘DESCRIPTION’ files. This information is used by the R package management system (specifically, R CMD `build` and R CMD `INSTALL`) to create the `Author` and `Maintainer` fields from `Authors@R`, unless these are still defined (statically) in ‘DESCRIPTION’. Also, the person information is used for reliably creating the author information in auto-generated package citations (more on this later). Note that there are no default roles for `Authors@R`: all authors must be given an author role (“aut”), and the maintainer must be identified by the creator role (“cre”).

Bibliographic information

Bibentry objects can hold information about an arbitrary positive number of bibliography entries. These can be obtained by one call to `bibentry()` with list arguments, or by first creating objects representing single bibentries and combining these via `c()`.

Bibentry objects represent bibliographic information in “enhanced $\text{BIB}\text{\TeX}$ style”, using the same entry types (such as ‘Article’ or ‘Book’) and field names (such as ‘author’ and ‘year’) as $\text{BIB}\text{\TeX}$. A single bibentry is created by calling `bibentry()` with arguments `bibtype` (typically used positionally) and `key` (optionally) giving the type of the entry and a key for it, and further fields given in `tag = value` form, with `tag` and `value` the name and value of the field, respectively. See `?bibentry` for details on types and

fields. The author and editor field values can be person objects as discussed above (and are coerced to such using `as.person()` if they are not). In addition, it is possible to specify header and footer information for the individual entries (arguments `header` and `footer`) and the collection of entries (arguments `mheader` and `mfooter`).

Bibentry objects can be subscripted by field (using `$`) or by position (using `[]`), and have `print()` and `format()` methods providing a choice between at least six different styles (as controlled by a `style` argument): plain text, HTML, $\text{L}\text{\TeX}$, $\text{BIB}\text{\TeX}$, R (for formatting bibentries as (high-level) R code), and citation information (including headers/footers and a mixture of plain text and $\text{BIB}\text{\TeX}$). The first three make use of a `.bibstyle` argument using the `bibstyle()` function in package `tools` which defines the display style of the citation by providing a collection of functions to format the individual parts of the entry. Currently the default style “JSS”, which corresponds to the bibliography style employed by the *Journal of Statistical Software* (<http://www.jstatsoft.org/>), is the only available style, but it can be modified by replacing the component functions, or replaced with a completely new style. A style is implemented as an environment, and is required to contain functions to format each type of bibliographic entry (`formatArticle()`, `formatBook()`, etc.) as well a function `sortKeys()` to sort entries. In addition to these required functions, the “JSS” style includes several dozen auxiliary functions that are employed by the required ones. Users modifying it should read the help page `?tools:bibstyle`, which includes an example that changes the sort order. Those implementing a completely new style should consult the source code.

Figure 2 shows an illustrative example for using `bibentry()`: A vector with two bibentries is created for the `boot` package and the book whose methods it implements. In the first bibentry the person object `p` from Figure 1 is reused while the second bibentry applies `as.person()` to a string with author and role information. Subsequently, the bibentry is printed in “JSS” style and transformed to $\text{BIB}\text{\TeX}$. Note that the roles of the persons in the author field are *not* employed in `toBibtex()` as $\text{BIB}\text{\TeX}$ does not support that concept. (Hence, the role information in the `as.person()` call creating `b[2]` would not have been necessary for producing $\text{BIB}\text{\TeX}$ output.)

Currently, the primary use of bibentry objects is in package ‘CITATION’ files, which contain R code for defining such objects.¹ Function `citation()` collects all of the defined bibentry objects into a single object for creating package citations, primarily for visual inspection and export to $\text{BIB}\text{\TeX}$. (Technically, `citation()` creates an object of class “citation” inheriting from “bibentry”, for which the `print()`

¹Typically, the R code should contain `bibentry()` calls but special constructs for reusing the information from the ‘DESCRIPTION’ are also available as is some legacy functionality (`citEntry()` etc.). More details are provided in the following.

```
R> b <- c(
+   bibentry(
+     bibtype = "Manual",
+     title = "{boot}: Bootstrap {R} ({S-Plus}) Functions",
+     author = p,
+     year = "2012",
+     note = "R package version 1.3-4",
+     url = "http://CRAN.R-project.org/package=boot",
+     key = "boot-package"
+   ),
+
+   bibentry(
+     bibtype = "Book",
+     title = "Bootstrap Methods and Their Applications",
+     author = "Anthony C. Davison [aut], David V. Hinkley [aut]",
+     year = "1997",
+     publisher = "Cambridge University Press",
+     address = "Cambridge",
+     isbn = "0-521-57391-2",
+     url = "http://statwww.epfl.ch/davison/BMA/",
+     key = "boot-book"
+   )
+ )
R> b
```

Canty A and Ripley BD (2012). `_boot: Bootstrap R (S-Plus) Functions_`. R package version 1.3-4, <URL: <http://CRAN.R-project.org/package=boot>>.

Davison AC and Hinkley DV (1997). `_Bootstrap Methods and Their Applications_`. Cambridge University Press, Cambridge. ISBN 0-521-57391-2, <URL: <http://statwww.epfl.ch/davison/BMA/>>.

```
R> ## formatting for BibTeX
R> toBibtex(b[1])
```

```
@Manual{boot-package,
  title = {boot}: Bootstrap {R} ({S-Plus}) Functions},
  author = {Angelo Canty and Brian D. Ripley},
  year = {2012},
  note = {R package version 1.3-4},
  url = {http://CRAN.R-project.org/package=boot},
}
```

Figure 2: Example for `bibentry()`, specifying citations for the `boot` package (i.e. the package itself and the book which it implements).

method uses a different default style.)

Package citations can also be auto-generated from the package ‘DESCRIPTION’ metadata, see Figure 3 for an example: The `bibentry` title field is generated from the package metadata fields `Package` and `Title` which should be capitalized (in title case), and not use markup or end in a period. (Hence, it may need manual touch-ups, e.g. protecting `{S}` or `{B}`ayesian etc.) The year is taken from `Date/Publication` (for CRAN packages) or `Date` (if available), and the note field is generated from the

package’s `Version`. If the package was installed from CRAN, the official stable CRAN package URL (here: <http://CRAN.R-project.org/package=boot>) is given in the `url` field.² Finally, the `bibentry` author field is generated from the description `Author` field (unless there is an `Authors@R`, see below). As the `Author` field in ‘DESCRIPTION’ is a plain text field intended for human readers, but not necessarily for automatic processing, it may happen that the auto-generated BIBTEX is incorrect (as in Figure 3). Hence, `citation()` provides an ATTENTION note to this

² Note that the current physical URLs, e.g. <http://CRAN.R-project.org/web/packages/boot/index.html> are not guaranteed to be stable and may change in the future (as they have in the past).

```
R> citation("boot", auto = TRUE)
```

To cite package 'boot' in publications use:

```
Angelo Canty and Brian Ripley (2012). boot: Bootstrap Functions (originally by Angelo Canty for S). R package version 1.3-4. http://CRAN.R-project.org/package=boot
```

A BibTeX entry for LaTeX users is

```
@Manual{,
  title = {boot: Bootstrap Functions (originally by Angelo Canty for S)},
  author = {Angelo Canty and Brian Ripley},
  year = {2012},
  note = {R package version 1.3-4},
  url = {http://CRAN.R-project.org/package=boot},
}
```

Figure 3: Citation for the **boot** package auto-generated from its 'DESCRIPTION' file. (Note that this is not the recommended citation of the package authors, see `citation("boot")` without `auto = TRUE` for that.)

end. One can overcome this problem by providing an `Authors@R` field in 'DESCRIPTION', from which the names of all persons with author roles can reliably be determined and included in the author field. Note again that author roles ("aut") are not "implied" as defaults, and must be specified explicitly.

As illustrated in Figure 3, one can inspect the auto-generated package citation bibentry by giving `auto = TRUE` when calling `citation()` on a package – even if the package provides a 'CITATION' file in which case `auto` defaults to `FALSE`. The motivation is that it must be possible to refer to the package *itself*, even if the preferred way of citing the package in publications is *via* the references given in the package 'CITATION' file. Of course, both approaches might also be combined as in the **boot** package where one of the bibentries in 'CITATION' corresponds essentially to the auto-generated package bibentry (see `citation("boot")`). To facilitate incorporating the auto-generated citations into 'CITATION', one can include a `citation(auto = meta)` entry: when evaluating the code in the 'CITATION' file, `citation()` automatically sets `meta` to the package metadata. This eliminates the need for more elaborate programmatic constructions of the package citation bibentry (or even worse, manually duplicating the relevant package metadata), provided that auto-generation is reliable (i.e. correctly processes author names and does not need additional L^AT_EX markup or protection in the title). To check whether this is the case for a particular package's metadata and hence `citation(auto = meta)` can conveniently be used in 'CITATION', one can employ constructs such as `citation(auto = packageDescription("boot"))`.

³Thus, the information from the 'DESCRIPTION' can be reused in the 'CITATION'. However, the reverse is not possible as the 'DESCRIPTION' needs to be self-contained.

Actions for package maintainers

Package maintainers should provide an `Authors@R` entry in their 'DESCRIPTION' files to allow computations on the list of authors, in particular to ensure that auto-generation of bibliographic references to their packages works reliably. Typically, the `Author` and `Maintainer` entries in 'DESCRIPTION' can then be omitted in the sources (as these are automatically added by R CMD `build`).

Packages wanting to employ the auto-generated citation as (one of) the officially recommended reference(s) to their package can use the new simplified `citation(auto = meta)` to do so.³ Note that if this is used then the package's citation depends on R at least 2.14.0, suggesting to declare this requirement in the `Depends` field of 'DESCRIPTION'.

As of 2012-04-02, CRAN had 3718 packages in total, with about 20% (734) having 'CITATION' files, but less than 2.5% (92) already providing an `Authors@R` entry in their package metadata: so clearly, there is substantial room for future improvement.

Conclusions and outlook

R has provided functionality for representing persons and generating package citations for several years now. Earlier versions used 'CITATION' files with calls to `citEntry()` (and possibly `citHeader()` and `citFooter()`). R 2.12.0 introduced the new bibentry functionality for improved representation and manipulation of bibliographic information (the old mechanism is still supported and implemented by the new one). R 2.12.0 also added the new per-

son functionality (changing to the more appropriate given/family scheme and adding roles). R 2.14.0 added more functionality, including the `Authors@R` field in package ‘DESCRIPTION’ files from which `Author` and `Maintainer` fields in this file as well as package bibentries can reliably be auto-generated.

Having reliable comprehensive information on package “authors” and their roles which can be processed automatically is also highly relevant for the purpose of understanding the roles of persons in the social fabric underlying the remarkable growth of the R package multiverse, opening up fascinating possibilities for research on R (of course, with R), e.g. using social network analysis and related methods.

Since R 2.10.0, R help files have been able to contain R code in `\Sexpr` macros to be executed before the help text is displayed. In the future, it is planned to use this facility to auto-generate `\references` sections based on citations in the text, much as \LaTeX and \BibTeX can automatically generate bibliographies.

These enhancements will clearly make dealing with bibliographic information in Rd files much more convenient and efficient, and eliminate the need to “manually” (re)format entries in the `\references` sections. Of course, this should not come at the price of needing to manually generate bibentries for references already available in other bibliographic formats. For \BibTeX format ‘.bib’ databases (presumably the format typically employed by R users), conversion is straightforward: one can read the entries in ‘.bib’ files into bibentry objects using `read.bib()` in CRAN package **bibtex** (François, 2011), use `format()` with `style = "R"` to obtain a character vector with `bibentry()` calls for each bibentry, and use `writeLines()` to write (or add) this to a bibentry database file (see `?bibentry` for examples of the last two steps). For other formats, one could first transform to \BibTeX format: e.g. **Bibutils** (Putnam, 2010) can convert between the bibliography formats COPAC, EndNote refer, EndNote XML, Pubmed XML, ISI Web of Science, US Library of Congress MODS XML, RIS, and Word 2007. However, this may lose useful information: e.g. the MODS format can provide role data which (as discussed above) will be dropped when converting to \BibTeX format. We are thus planning to provide a **bibutils** package which instead converts to the MODS XML format first, and creates bibentry objects from this. However, as this functionality requires both external software and package **XML** (Temple Lang, 2012), it cannot be made available within base R.

With these functionalities in place, managing bibliographic information in R documentation files will become much more convenient, and R will become a much sharper knife for cutting-edge bibliometric

and scientometric analyses. In particular, it will become possible to programmatically analyze package citations, and use these to learn about the processes of knowledge dissemination driving the creation of R packages, and the relatedness of package authors and concepts.

Bibliography

- K. Berry and O. Patashnik. \BibTeX , 2010. URL <http://www.ctan.org/tex-archive/biblio/bibtex/base>.
- A. Canty and B. D. Ripley. *boot: Bootstrap R (S-Plus) Functions*, 2012. URL <http://CRAN.R-project.org/package=boot>. R package version 1.3-4.
- R. François. *bibtex: \BibTeX Parser*, 2011. URL <http://CRAN.R-project.org/package=bibtex>. R package version 0.3-0.
- Library of Congress. MARC standards. URL <http://www.loc.gov/marc/>, accessed 2012-04-02, 2012.
- C. Putnam. *Bibutils: Bibliography Conversion Utilities*. The Scripps Research Institute, 2010. URL <http://sourceforge.net/p/bibutils/>.
- D. Temple Lang. *XML: Tools for Parsing and Generating XML within R and S-Plus*, 2012. URL <http://CRAN.R-project.org/package=XML>. R package version 3.9-4.
- Wikipedia. Personal name – Wikipedia, the free encyclopedia. URL http://en.wikipedia.org/wiki/Personal_name, accessed 2012-04-02, 2012.

Kurt Hornik
 Department of Finance, Accounting and Statistics
 WU Wirtschaftsuniversität Wien
 Augasse 2–6, 1090, Wien
 Austria
Kurt.Hornik@R-project.org

Duncan Murdoch
 Department of Statistical and Actuarial Sciences
 University of Western Ontario
 London, Ontario
 Canada
murdoch@stats.uwo.ca

Achim Zeileis
 Faculty of Economics and Statistics
 Universität Innsbruck
 Universitätsstr. 15, 6020 Innsbruck
 Austria
Achim.Zeileis@R-project.org

Changes in R

From version 2.14.1 to version 2.15.1

by the R Core Team

CHANGES IN R VERSION 2.15.1

NEW FEATURES

- `source()` now uses `withVisible()` rather than `.Internal(eval.with.vis)`. This sometimes alters tracebacks slightly.
- `install.packages("pkg_version.tgz")` on Mac OS X now has sanity checks that this is actually a binary package (as people have tried it with incorrectly named source packages).
- `splineDesign()` and `spline.des()` in package **splines** have a new option `sparse` which can be used for efficient construction of a sparse B-spline design matrix (*via Matrix*).
- `norm()` now allows `type = "2"` (the 'spectral' or 2-norm) as well, mainly for didactical completeness.
- `pmin()` and `pmax()` now also work when one of the inputs is of length zero and others are not, returning a zero-length vector, analogously to, say, `+`.
- `colorRamp()` (and hence `colorRampPalette()`) now also works for the boundary case of just one color when the ramp is flat.
- `qqline()` has new optional arguments `distribution`, `probs` and `qtype`, following the example of **lattice**'s `panel.qqmathline()`.
- `.C()` gains some protection against the misuse of character vector arguments. (An all too common error is to pass `character(N)`, which initializes the elements to "", and then attempt to edit the strings in-place, sometimes forgetting to terminate them.)
- Calls to the new function `globalVariables()` in package **utils** declare that functions and other objects in a package should be treated as globally defined, so that `CMD` check will not note them.
- `print(packageDescription(*))` trims the `Collate` field by default.
- The included copy of `zlib` has been updated to version 1.2.7.

- A new option `"show.error.locations"` has been added. When set to `TRUE`, error messages will contain the location of the most recent call containing source reference information. (Other values are supported as well; see `?options`.)
- The NA warning messages from e.g. `pchisq()` now report the call to the closure and not that of the `.Internal`.
- Added Polish translations by Łukasz Daniel.

PERFORMANCE IMPROVEMENTS

- In package **parallel**, `makeForkCluster()` and the multicore-based functions use native byte-order for serialization (deferred from 2.15.0).
- `lm.fit()`, `lm.wfit()`, `glm.fit()` and `lsfit()` do less copying of objects, mainly by using `.Call()` rather than `.Fortran()`.
- `.C()` and `.Fortran()` do less copying: arguments which are raw, logical, integer, real or complex vectors and are unnamed are not copied before the call, and (named or not) are not copied after the call. Lists are no longer copied (they are supposed to be used read-only in the C code).
- `tabulate()` makes use of `.C(DUP = FALSE)` and hence does not copy `bin`. (Suggested by Tim Hesterberg.) It also avoids making a copy of a factor argument `bin`.
- Other functions (often or always) doing less copying include `cut()`, `dist()`, the complex case of `eigen()`, `hclust()`, `image()`, `kmeans()`, `loess()`, `stl()` and `svd(LINPACK = TRUE)`.
- There is less copying when using primitive replacement functions such as `names()`, `attr()` and `attributes()`.

DEPRECATED AND DEFUNCT

- The converters for use with `.C()` (see `?getCCConverterDescriptions`) are deprecated: use the `.Call()` interface instead. There are no known examples (they were never fully documented).

UTILITIES

- For R `CMD` check, a few people have reported problems with junctions on Windows (although they were tested on Windows 7,

XP and Server 2008 machines and it is unknown under what circumstances the problems occur). Setting the environment variable `R_WIN_NO_JUNCTIONS` to a non-empty value (e.g. in `'~/R/check.Renviron'`) will force copies to be used instead.

INSTALLATION

- R CMD INSTALL with `_R_CHECK_INSTALL_DEPENDS_` set to a true value (as done by R CMD check --as-cran) now restricts the packages available when lazy-loading as well as when test-loading (since packages such as **ETLUtils** and **agsemisc** had top-level calls to `library()` for undeclared packages).

This check is now also available on Windows.

C-LEVEL FACILITIES

- C entry points `mkChar` and `mkCharCE` now check that the length of the string they are passed does not exceed $2^{31} - 1$ bytes: they used to overflow with unpredictable consequences.
- C entry points `R_GetCurrentSrcRef` and `R_GetSrcFilename` have been added to the API to allow debuggers access to the source references on the stack.

WINDOWS-SPECIFIC CHANGES

- Windows-specific changes will now be announced in this file ('NEWS'). Changes up and including R 2.15.0 remain in the 'CHANGES' file.
- There are two new environment variables which control the defaults for command-line options.

If `R_WIN_INTERNET2` is set to a non-empty value, it is as if `'--internet2'` was used.

If `R_MAX_MEM_SIZE` is set, it gives the default memory limit if `'--max-mem-size'` is not specified: invalid values being ignored.

BUG FIXES

- `lsfit()` lost the names from the residuals.
- More cases in which `merge()` could create a data frame with duplicate column names now give warnings. Cases where names specified in `by` match multiple columns are errors.
- Nonsense uses such as `seq(1:50, by = 5)` (from package **plotrix**) and `seq.int(1:50, by = 5)` are now errors.

- The residuals in the 5-number summary printed by `summary()` on an "lm" object are now explicitly labelled as weighted residuals when non-constant weights are present. (Wish of PR#14840.)

- `tracemem()` reported that all objects were copied by `.C()` or `.Fortran()` whereas only some object types were ever copied.

It also reported and marked as copies *some* transformations such as `rexp(n, x)`: it no longer does so.

- The `plot()` method for class "stepfun" only used the optional `xval` argument to compute `xlim` and not the points at which to plot (as documented). (PR#14864)

- Names containing characters which need to be escaped were not deparsed properly. (PR#14846)

- Trying to update (recommended) packages in `'R_HOME/library'` without write access is now dealt with more gracefully. Further, such package updates may be skipped (with a warning), when a newer installed version is already going to be used from `.libPaths()`. (PR#14866)

- `hclust()` is now fast again (as up to end of 2003), with a different fix for the "median"/"centroid" problem. (PR#4195).

- `get_all_vars()` failed when the data came entirely from vectors in the global environment. (PR#14847)

- R CMD check with `_R_CHECK_NO_RECOMMENDED_` set to a true value (as done by the `--as-cran` option) could issue false errors if there was an indirect dependency on a recommended package.

- `formatC()` uses the C entry point `str_signif` which could write beyond the length allocated for the output string.

- Missing default argument added to implicit S4 generic for `backsolve()`. (PR#14883)

- Some bugs have been fixed in handling load actions that could fail to export assigned items or generate spurious warnings in `CMD check` on loading.

- For `tiff(type = "windows")`, the numbering of per-page files except the last was off by one.

- On Windows, loading package **stats** (which is done for a default session) would switch line endings on `'stdout'` and `'stderr'` from CRLF to LF. This affected `Rterm` and `R CMD BATCH`.

- On Windows, the compatibility function `x11()` had not kept up with changes to `windows()`, and issued warnings about bad parameters. (PR#14880)
- On Windows, the `Sys.glob()` function did not handle UNC paths as it was designed to try to do. (PR#14884)
- In package **parallel**, `clusterApply()` and similar failed to handle a (pretty pointless) length-1 argument. (PR#14898)
- Quartz Cocoa display reacted asynchronously to `dev.flush()` which means that the re-draw could be performed after the plot has been already modified by subsequent code. The redraw is now done synchronously in `dev.flush()` to allow animations without sleep cycles.
- Source locations reported in `traceback()` were incorrect when byte-compiled code was on the stack.
- `plogis(x, lower = FALSE, log.p = TRUE)` no longer underflows early for large x (e.g. 800).
- ?Arithmetic's "1 ^ y and y ^ 0 are 1, always" now also applies for integer vectors y .
- X11-based pixmap devices like `png(type = "Xlib")` were trying to set the cursor style, which triggered some warnings and hangs.
- Code executed by the built-in HTTP server no longer allows other HTTP clients to re-enter R until the current worker evaluation finishes, to prevent cascades.
- The `plot()` and `Axis()` methods for class "table" now respect graphical parameters such as `cex.axis`. (Reported by Martin Becker.)
- Under some circumstances `package.skeleton()` would give out progress reports that could not be translated and so were displayed by question marks. Now they are always in English. (This was seen for CJK locales on Windows, but may have occurred elsewhere.)
- The evaluator now keeps track of source references outside of functions, e.g. when `source()` executes a script.
- The replacement method for `window()` now works correctly for multiple time series of class "mts". (PR#14925)
- `is.unsorted()` gave incorrect results on non-atomic objects such as data frames. (Reported by Matthew Dowle.)
- The value returned by `tools::psnice()` for invalid pid values was not always NA as documented.
- Closing an `X11()` window while `locator()` was active could abort the R process.
- `getMethod(f, sig)` produced an incorrect error message in some cases when f was not a string.
- Using a string as a "call" in an error condition with `options(showErrorCalls=TRUE)` could cause a segfault. (PR#14931)
- The string "infinity" allowed by C99 was not accepted as a numerical string value by e.g. `scan()` and `as.character()`. (PR#14933)
- In `legend()`, setting some entries of `lwd` to NA was inconsistent (depending on the graphics device) in whether it would suppress those lines; now it consistently does so. (PR#14926)
- `by()` failed for a zero-row data frame. (Reported by Weiqiang Qian)
- Yates correction in `chisq.test()` could be bigger than the terms it corrected, previously leading to an infinite test statistic in some corner cases which are now reported as NaN.
- `xgettext()` and related functions sometimes returned items that were not strings for translation. (PR#14935)
- `plot(<lm>, which=5)` now correctly labels the factor level combinations for the special case where all h_{ij} are the same. (PR#14837)

CHANGES IN R VERSION 2.15.0

SIGNIFICANT USER-VISIBLE CHANGES

- The behaviour of `unlink(recursive = TRUE)` for a symbolic link to a directory has changed: it now removes the link rather than the directory contents (just as `rm -r` does).

On Windows it no longer follows reparse points (including junctions and symbolic links).

NEW FEATURES

- Environment variable `RD2DVI_INPUTENC` has been renamed to `RD2PDF_INPUTENC`.
- `.Deprecated()` becomes a bit more flexible, getting an `old` argument.
- Even data-only packages without R code need a namespace and so may need to be installed under R 2.14.0 or later.

- `assignInNamespace()` has further restrictions on use apart from at top-level, as its help page has warned. Expect it to be disabled from programmatic use in the future.
- `system()` and `system2()` when capturing output report a non-zero status in the new "status" attribute.
- `kronecker()` now has an S4 generic in package **methods** on which packages can set methods. It will be invoked by `X %x% Y` if either `X` or `Y` is an S4 object.
- `pdf()` accepts forms like `file = "|lpr"` in the same way as `postscript()`.
- `pdf()` accepts `file = NULL`. This means that the device does NOT create a PDF file (but it can still be queried, e.g., for font metric info).
- `format()` (and hence `print()`) on "bibentry" objects now uses `options("width")` to set the output width.
- `legend()` gains a `text.font` argument. (Suggested by Tim Paine, PR#14719.)
- `nchar()` and `nzchar()` no longer accept factors (as integer vectors). (Wish of PR#6899.)
- `summary()` behaves slightly differently (or more precisely, its `print()` method does). For numeric inputs, the number of NAs is printed as an integer and not a real. For dates and datetimes, the number of NAs is included in the printed output (the latter being the wish of PR#14720).

The "data.frame" method is more consistent with the default method: in particular it now applies `zapsmall()` to numeric/complex summaries.

- The number of items retained with `options(warn = 0)` can be set by `options(nwarnings=)`.
- A new function `assignInMyNamespace()` uses the namespace of the function it is called from.
- `attach()` allows the default name for an attached file to be overridden.
- `bxp()`, the work horse of `boxplot()`, now uses a more sensible default `xlim` in the case where `at` is specified differently from `1:n`, see the discussion on R-devel, <https://stat.ethz.ch/pipermail/r-devel/2011-November/062586.html>.
- New function `paste0()`, an efficient version of `paste(*, sep="")`, to be used in many places for more concise (and slightly more efficient) code.

- Function `setClass()` in package **methods** now returns, invisibly, a generator function for the new class, slightly preferred to calling `new()`, as explained on the `setClass` help page.
- The "dendrogram" method of `str()` now takes its default for `last.str` from option `str.dendrogram.last`.
- New simple `fitted()` method for "kmeans" objects.
- The `traceback()` function can now be called with an integer argument, to display a current stack trace. (Wish of PR#14770.)
- `setGeneric()` calls can be simplified when creating a new generic function by supplying the default method as the `def` argument. See `?setGeneric`.
- `serialize()` has a new option `xdr = FALSE` which will use the native byte-order for binary serializations. In scenarios where only little-endian machines are involved (these days, close to universal) and (un)serialization takes an appreciable amount of time this may speed up noticeably transferring data between systems.
- The internal (un)serialization code is faster for long vectors, particularly with XDR on some platforms. (Based on a suggested patch by Michael Spiegel.)
- For consistency, circles with zero radius are omitted by `points()` and `grid.circle()`. Previously this was device-dependent, but they were usually invisible.
- `NROW(x)` and `NCOL(x)` now work whenever `dim(x)` looks appropriate, e.g., also for more generalized matrices.
- PCRE has been updated to version 8.30.
- The internal `R_Sceref` variable is now updated before the browser stops on entering a function. (Suggestion of PR#14818.)
- There are 'bare-bones' functions `.colSums()`, `.rowSums()`, `.colMeans()` and `.rowMeans()` for use in programming where ultimate speed is required.
- The function `.package_dependencies()` from package **tools** for calculating (recursive) (reverse) dependencies on package databases, which was formerly internal, has been renamed to `package_dependencies()` and is now exported.
- There is a new function `optimHess()` to compute the (approximate) Hessian for an `optim()` solution if `hessian = TRUE` was forgotten.

- `.filled.contour()` is a 'bare-bones' function to add a filled-contour rectangular plot to an already prepared plot region.
- The stepping in debugging and single-step browsing modes has changed slightly: now left braces at the start of the body are stepped over for `if` statements as well as for `for` and `while` statements. (Wish of PR#14814.)
- `library()` no longer warns about a conflict with a function from `package:base` if the function has the same code as the base one but with a different environment. (An example is `Matrix::det()`.)
- When deparsing very large language objects, `as.character()` now inserts newlines after each line of approximately 500 bytes, rather than truncating to the first line.
- New function `rWishart()` generates Wishart-distributed random matrices.
- Packages may now specify actions to be taken when the package is loaded (`setLoadActions()`).
- `options(max.print = Inf)` and similar now give an error (instead of warnings later).
- The "difftime" replacement method of `units` tries harder to preserve other attributes of the argument. (Wish of PR#14839.)
- `poly(raw = TRUE)` no longer requires more unique points than the degree. (Requested by John Fox.)

PACKAGE parallel

- There is a new function `mcmapply()`, a parallel version of `mapply()`, and a wrapper `mcMap()`, a parallel version of `Map()`.
- A default cluster can be registered by the new function `setDefaultCluster()`: this will be used by default in functions such as `parLapply()`.
- `clusterMap()` has a new argument `.scheduling` to allow the use of load-balancing.
- There are new load-balancing functions `parLapplyLB()` and `parSapplyLB()`.
- `makePSOCKcluster()` has a new option `useXDR = FALSE` which can be used to avoid byte-shuffling for serialization when all the nodes are known to be little-endian (or all big-endian).

PACKAGE INSTALLATION

- Non-ASCII vignettes without a declared encoding are no longer accepted.
- C/C++ code in packages is now compiled with `-NDEBUG` to mitigate against the C/C++ function `assert` being called in production use. Developers can turn this off during package development with `PKG_CPPFLAGS = -UNDEBUG`.
- R CMD INSTALL has a new option `'--dsym'` which on Mac OS X (Darwin) dumps the symbols alongside the `'so'` file: this is helpful when debugging with `valgrind` (and especially when installing packages into `'R.framework'`). [This can also be enabled by setting the undocumented environment variable `PKG_MAKE_DSYM`, since R 2.12.0.]
- R CMD INSTALL will test loading under all installed sub-architectures even for packages without compiled code, unless the flag `'--no-multiarch'` is used. (Pure R packages can do things which are architecture-dependent: in the case which prompted this, looking for an icon in a Windows R executable.)
- There is a new option `install.packages(type = "both")` which tries source packages if binary packages are not available, on those platforms where the latter is the default.
- The meaning of `install.packages(dependencies = TRUE)` has changed: it now means to install the essential dependencies of the named packages plus the 'Suggests', but only the essential dependencies of dependencies. To get the previous behaviour, specify dependencies as a character vector.
- R CMD INSTALL `--merge-multiarch` is now supported on OS X and other Unix-alikes using multiple sub-architectures.
- R CMD INSTALL `--libs-only` now by default does a test load on Unix-alikes as well as on Windows: suppress with `'--no-test-load'`.

UTILITIES

- R CMD check now gives a warning rather than a note if it finds inefficiently compressed datasets. With `bzip2` and `xz` compression having been available since R 2.10.0, it only exceptionally makes sense to not use them.

The environment variable `_R_CHECK_COMPACT_DATA2_` is no longer consulted: `check` is always done if `_R_CHECK_COMPACT_DATA_` has a true value (its default).

- Where multiple sub-architectures are to be tested, R CMD check now runs the examples and tests for all the sub-architectures even if one fails.
- R CMD check can optionally report timings on various parts of the check: this is controlled by environment variable `_R_CHECK_TIMINGS_` documented in ‘Writing R Extensions’. Timings (in the style of R CMD BATCH) are given at the foot of the output files from running each test and the R code in each vignette.
- There are new options for more rigorous testing by R CMD check selected by environment variables – see the ‘Writing R Extensions’ manual.
- R CMD check now warns (rather than notes) on undeclared use of other packages in examples and tests: increasingly people are using the metadata in the ‘DESCRIPTION’ file to compute information about packages, for example reverse dependencies.
- The defaults for some of the options in R CMD check (described in the ‘R Internals’ manual) have changed: checks for unsafe and `.Internal()` calls and for partial matching of arguments in R function calls are now done by default.
- R CMD check has more comprehensive facilities for checking compiled code and so gives fewer reports on entry points linked into ‘.so’/‘.dll’ files from libraries (including C++ and Fortran runtimes).

Checking compiled code is now done on FreeBSD (as well as the existing supported platforms of Linux, Mac OS X, Solaris and Windows).

- R CMD build has more options for ‘--compact-vignettes’: see R CMD build --help.
- R CMD build has a new option ‘--md5’ to add an ‘MD5’ file (as done by CRAN): this is used by R CMD INSTALL to check the integrity of the distribution.

If this option is not specified, any existing (and probably stale) ‘MD5’ file is removed.

DEPRECATED AND DEFUNCT

- R CMD Rd2dvi is now defunct: use R CMD Rd2pdf.
- Options such ‘--max-nszie’, ‘--max-vsize’ and the function `mem.limits()` are now defunct. (Options ‘--min-nszie’ and ‘--min-vsize’ remain available.)

- Use of `library.dynam()` without specifying all the first three arguments is now disallowed.

Use of an argument `chname` in `library.dynam()` including the extension ‘.so’ or ‘.dll’ (which was never allowed according to the help page) is defunct. This also applies to `library.dynam.unload()` and to `useDynLib` directives in ‘NAMESPACE’ files.

- The internal functions `.readRDS()` and `.saveRDS()` are now defunct.
- The off-line help() types “postscript” and “ps” are defunct.
- `Sys.putenv()`, replaced and deprecated in R 2.5.0, is finally removed.
- Some functions/objects which have been defunct for five or more years have been removed completely. These include `.Alias()`, `La.chol()`, `La.chol2inv()`, `La.eigen()`, `Machine()`, `Platform()`, `Version`, `codes()`, `delay()`, `format.char()`, `getenv()`, `httpClient()`, `loadURL()`, `machine()`, `parse.dcf()`, `printNoClass()`, `provide()`, `read.table.url()`, `restart()`, `scan.url()`, `symbol.C()`, `symbol.For()` and `unix()`.
- The `ENCODING` argument to `.C()` is deprecated. It was intended to smooth the transition to multi-byte character strings, but can be replaced by the use of `iconv()` in the rare cases where it is still needed.

INSTALLATION

- Building with a positive value of ‘--with-valgrind-instrumentation’ now also instruments logical, complex and raw vectors.

C-LEVEL FACILITIES

- Passing R objects other than atomic vectors, functions, lists and environments to `.C()` is now deprecated and will give a warning. Most cases (especially NULL) are actually coding errors. NULL will be disallowed in future.

`.C()` now passes a pairlist as a SEXP to the compiled code. This is as was documented, but pairlists were in reality handled differently as a legacy from the early days of R.

- `call_R` and `call_S` are deprecated. They still exist in the headers and as entry points, but are no longer documented and should not be used for new code.

BUG FIXES

- `str(x,width)` now obeys its `width` argument also for function headers and other objects `x` where `deparse()` is applied.
- The convention for `x %% 0L` for integer-mode `x` has been changed from `0L` to `NA_integer_`. (PR#14754)
- The `exportMethods` directive in a 'NAMESPACE' file now exports S4 generics as necessary, as the extensions manual said it does. The manual has also been updated to be a little more informative on this point.
It is now required that there is an S4 generic (imported or created in the package) when methods are to be exported.
- Reference methods cannot safely use non-exported entries in the namespace. We now do not do so, and warn in the documentation.
- The namespace import code was warning when identical S4 generic functions were imported more than once, but should not (reported by Brian Ripley, then Martin Morgan).
- `merge()` is no longer allowed (in some ways) to create a data frame with duplicate column names (which confused PR#14786).
- Fixes for rendering raster images on X11 and Windows devices when the `x`-axis or `y`-axis scale is reversed.
- `getAnywhere()` found S3 methods as seen from the `utils` namespace and not from the environment from which it was called.
- `selectMethod(f,sig)` would not return inherited group methods when caching was off (as it is by default).
- `dev.copy2pdf(out.type = "cairo")` gave an error. (PR#14827)
- Virtual classes (e.g., class unions) had a `NULL` prototype even if that was not a legal subclass. See `?setClassUnion`.
- The C prototypes for `zdotc` and `zdotu` in 'R_ext/BLAS.h' have been changed to the more modern style rather than that used by `f2c`. (Patch by Berwin Turlach.)
- `isGeneric()` produced an error for primitives that can not have methods.
- `.C()` or `.Fortran()` had a lack-of-protection error if the registration information resulted in an argument being coerced to another type.

- `boxplot(x=x,at=at)` with non finite elements in `x` and non integer `at` could not generate a warning but failed.
- `heatmap(x,symm=TRUE,RowSideColors=*)` no longer draws the colors in reversed order.
- `predict(<ar>)` was incorrect in the multivariate case, for `p >= 2`.
- `print(x,max=m)` is now consistent when `x` is a "Date"; also the "reached ... max.print .." messages are now consistently using single brackets.
- Closed the '' tag in pages generated by `Rd2HTML()`. (PR#14841.)
- Axis tick marks could go out of range when a log scale was used. (PR#14833.)
- Signature objects in methods were not allocated as S4 objects (caused a problem with `trace()` reported by Martin Morgan).

CHANGES IN R VERSION 2.14.2

NEW FEATURES

- The internal `untar()` (as used by default by R CMD INSTALL) now knows about some `pax` headers which `bsdtar` (e.g., the default `tar` for Mac OS `>= 10.6`) can incorrectly include in `tar` files, and will skip them with a warning.
- PCRE has been upgraded to version 8.21: as well as bug fixes and greater Perl compatibility, this adds a JIT pattern compiler, about which PCRE's news says 'large performance benefits can be had in many situations'. This is supported on most but not all R platforms.
- Function `compactPDF()` in package `tools` now takes the default for argument `gs_quality` from environment variable `GS_QUALITY`: there is a new value "none", the ultimate default, which prevents GhostScript being used in preference to `qpdf` just because environment variable `R_GSCMD` is set. If `R_GSCMD` is unset or set to "", the function will try to find a suitable GhostScript executable.
- The included version of `zlib` has been updated to 1.2.6.
- For consistency with the `logLik()` method, `nobs()` for "nls" files now excludes observations with zero weight. (Reported by Berwin Turlach.)

UTILITIES

- R CMD check now reports by default on licenses not according to the description in ‘Writing R Extensions’.
- R CMD check has a new option ‘--as-cran’ to turn on most of the customizations that CRAN uses for its incoming checks.

PACKAGE INSTALLATION

- R CMD INSTALL will now no longer install certain file types from ‘inst/doc’: these are almost certainly mistakes and for some packages are wasting a lot of space. These are ‘Makefile’, files generated by running LaTeX, and unless the package uses a ‘vignettes’ directory, PostScript and image bitmap files.

Note that only PDF vignettes have ever been supported: some of these files come from DVI/PS output from the Sweave defaults prior to R 2.13.0.

BUG FIXES

- R configured with ‘--disable-openmp’ would mistakenly set HAVE_OPENMP (internal) and SUPPORT_OPENMP (in ‘Rconfig.h’) even though no OpenMP flags were populated.
- The getS3method() implementation had an old computation to find an S4 default method.
- readLines() could overflow a buffer if the last line of the file was not terminated. (PR#14766)
- R CMD check could miss undocumented S4 objects in packages which used S4 classes but did not ‘Depends: methods’ in their ‘DESCRIPTION’ file.
- The HTML Help Search page had malformed links. (PR#14769)
- A couple of instances of lack of protection of SEXPs have been squashed. (PR#14772, PR#14773)
- image(x,useRaster=TRUE) misbehaved on single-column x. (PR#14774)
- Negative values for options("max.print") or the max argument to print.default() caused crashes. Now the former are ignored and the latter trigger an error. (PR#14779)
- The text of a function body containing more than 4096 bytes was not properly saved by the parser when entered at the console.
- Forgetting the #endif tag in an Rd file could cause the parser to go into a loop. (Reported by Hans-Jorg Bibiko.)
- str(*,, strict.width="cut") now also obeys list.len = n. (Reported by Sören Vogel.)
- Printing of arrays did not have enough protection (C level), e.g., in the context of capture.output(). (Reported by Hervé Pagès and Martin Morgan.)
- pdf(file = NULL) would produce a spurious file named ‘NA’. (PR#14808)
- list2env() did not check the type of its envir argument. (PR#14807)
- svg() could segfault if called with a non-existent file path. (PR#14790)
- make install can install to a path containing ‘+’ characters. (PR#14798)
- The edit() function did not respect the options("keep.source") setting. (Reported by Cleridy Lennert.)
- predict.lm(*, type="terms", terms=*, se.fit=TRUE) did not work. (PR#14817)
- There is a partial workaround for errors in the TRE regular-expressions engine with named classes and repeat counts of at least 2 in a MBCS locale (PR#14408): these are avoided when TRE is in 8-bit mode (e.g. for useBytes = TRUE and when all the data are ASCII).
- The C function R_Rep1DLLdol() did not call top-level handlers.
- The Quartz device was unable to detect window sessions on Mac OS X 10.7 (Lion) and higher and thus it was not used as the default device on the console. Since Lion any application can use window sessions, so Quartz will now be the default device if the user’s window session is active and R is not run via ssh which is at least close to the behavior in prior OS X versions.
- mclapply() would fail in code assembling the translated error message if some (but not all) cores encountered an error.
- format.POSIXlt(x) raised an arithmetic exception when x was an invalid object of class "POSIXlt" and parts were empty.
- installed.packages() has some more protection against package installs going on in parallel.
- .Primitive() could be mis-used to call .Internal() entry points.

CHANGES IN R VERSION 2.14.1

NEW FEATURES

- `parallel::detectCores()` is now able to find the number of physical cores (rather than CPUs) on Sparc Solaris.
It can also do so on most versions of Windows; however the default remains `detectCores(logical = TRUE)` on that platform.
- Reference classes now keep a record of which fields are locked. `$lock()` with no arguments returns the names of the locked fields.
- `HoltWinters()` reports a warning rather than an error for some optimization failures (where the answer might be a reasonable one).
- `tools::dependsOnPkg()` now accepts the shorthand dependencies = "all".
- `parallel::clusterExport()` now allows specification of an environment from which to export.
- The `quartz()` device now does tilde expansion on its file argument.
- `tempfile()` on a Unix-alike now takes the process ID into account. This is needed with **multicore** (and as part of **parallel**) because the parent and all the children share a session temporary directory, and they can share the C random number stream used to produce the unique part. Further, two children can call `tempfile()` simultaneously.
- Option `print` in Sweave's `RweaveLatex()` driver now emulates auto-printing rather than printing (which can differ for an S4 object by calling `show()` rather than `print()`).
- `filled.contour()` now accepts infinite values: previously it might have generated invalid graphics files (e.g. containing NaN values).

INSTALLATION

- On 64-bit Linux systems, `configure` now only sets `'LIBnn'` to `lib64` if `'/usr/lib64'` exists. This may obviate setting `'LIBnn'` explicitly on Debian-derived systems.
It is still necessary to set `'LIBnn = lib'` (or `'lib32'`) for 32-bit builds of R on a 64-bit OS on those Linux distributions capable for supporting that concept.
- `configure` looks for `'inconsolata.sty'`, and if not found adjusts the default `R_RD4PDF` to not use it (with a warning, since it is needed for high-quality rendering of manuals).

PACKAGE INSTALLATION

- R CMD `INSTALL` will now do a test load for all sub-architectures for which code was compiled (rather than just the primary sub-architecture).

UTILITIES

- When checking examples under more than one sub-architecture, R CMD `check` now uses a separate directory `'examples_arch'` for each sub-architecture, and leaves the output in file `'pkgname-Ex_arch.Rout'`. Some packages expect their examples to be run in a clean directory

BUG FIXES

- `stack()` now gives an error if no vector column is selected, rather than returning a 1-column data frame (contrary to its documentation).
- `summary.mlm()` did not handle objects where the formula had been specified by an expression. (Reported by Helios de Rosario Martinez).
- `tools::deparseLatex(dropBraces=TRUE)` could drop text as well as braces.
- `colormodel = "grey"` (new in R 2.14.0) did not always work in `postscript()` and `pdf()`.
- `file.append()` could return `TRUE` for failures. (PR#14727)
- `gzcon()` connections are no longer subject to garbage collection: it was possible for this to happen when unintended (e.g. when calling `load()`).
- `nobs()` does not count zero-weight observations for `glm()` fits, for consistency with `lm()`. This affects the `BIC()` values reported for such `glm()` fits. (Spotted by Bill Dunlap.)
- `options(warn = 0)` failed to end a (C-level) context with more than 50 accumulated warnings. (Spotted by Jeffrey Horner.)
- The internal `plot.default()` code did not do sanity checks on a `cex` argument, so invalid input could cause problems. (Reported by Ben Bolker.)
- `anyDuplicated(<array>, MARGIN=0)` no longer fails. (Reported by Hervé Pagès.)
- `read.dcf()` removes trailing blanks: unfortunately on some platforms this included `\xa0` (non-breaking space) which is the trailing byte of a UTF-8 character. It now only considers ASCII space and tab to be 'blank'.

- There was a sign error in part of the calculations for the variance returned by `KalmanSmooth()`. (PR#14738)
- `pbinom(10, 1e6, 0.01, log.p = TRUE)` was NaN thanks to the buggy fix to PR#14320 in R 2.11.0. (PR#14739)
- `RweaveLatex()` now emulates auto-printing rather than printing, by calling `methods::show()` when auto-printing would.
- `duplicated()` ignored `fromLast` for a one-column data frame. (PR#14742)
- `source()` and related functions did not put the correct timestamp on the source references; `srcfilecopy()` has gained a new argument `timestamp` to support this fix. (PR#14750)
- `srcfilecopy()` has gained a new argument `isFile` and now records the working directory, to allow debuggers to find the original source file. (PR#14826)
- LaTeX conversion of Rd files did not correctly handle preformatted backslashes. (PR#14751)
- HTML conversion of Rd files did not handle markup within tabular cells properly. (PR#14708)
- `source()` on an empty file with `keep.source = TRUE` tried to read from `stdin()`, in R 2.14.0 only. (PR#14753)
- The code to check Rd files in packages would abort if duplicate description sections were present.

Changes on CRAN

2011-12-01 to 2012-06-08

by Kurt Hornik and Achim Zeileis

New CRAN task views

DifferentialEquations Topic: Differential Equations.
Maintainer: Karline Soetaert and Thomas Petzoldt. Packages: *CollocInfer*, *FME*, *GillespieSSA*, *PBSddesolve*, *PKfit*, *PKtools*, *PSM*, *ReacTran*, *bvpSolve**, *deSolve**, *deTestSet*, *ecolMod*, *mkin*, *nlmeODE*, *pomp*, *pracma*, *rootSolve**, *scaRabee*, *sde**, *simecol*.

(* = core package)

New packages in CRAN task views

Bayesian *AtelierR*, *Runuran*, *iterLap*.

ChemPhys *FITSio*, *cda*, *dielectric*, *planar*, *ringscale*, *stellaR*.

ClinicalTrials *AGSDest*, *CRTSize**, *FrF2*, *MSToolkit**, *PIPS**, *PowerTOST**, *TEQR**, *ThreeGroups*, *adaptTest**, *asd**, *bcrm**, *clin-sig*, *conf.design*, *crossdes**, *dfcrm**, *long-power*, *medAdherence*, *nppbib*, *pwr**, *qtlDesign**, *samplesize**, *tdm*.

Cluster *CoClust*, *bgmm*, *pmclust*, *psychomix*.

Distributions *LambertW*, *TSA*, *agricolae*, *e1071*, *emdbook*, *modeest*, *moments*, *npde*, *s20x*.

Econometrics *geepack*, *intReg*, *lfe*, *mhurdle*, *pcse*, *spdep*, *sphet*.

Environmetrics *Interpol.T*, *RMAWGEN*, *anm*, *boussinesq*, *fast*, *hydroPSO*, *nsRFA*, *qualV*, *rconifers*, *sensitivity*, *tiger*.

ExperimentalDesign *dynaTree*, *mixexp*, *plgp*, *tgp*.

Finance *AmericanCallOpt*, *BurStFin*, *FinAsym*, *VarSwapPrice*, *crp.CSFP*, *frmqa*.

HighPerformanceComputing *BatchExperiments*, *BatchJobs*, *WideLM*, *cloudRmpi*, *cloudRmpi-Jars*, *harvestr*, *pmclust*, *rreval*.

MachineLearning *GMMBoost*, *RPM*, *bst*, *evtree*, *gamboostLSS*, *gbev*, *longRPart*, *oblique.tree*, *obliqueRF*, *partykit*, *rattle*, *rpartOrdinal*.

MedicalImaging *DATforDCEMRI**, *arf3DS4**, *neuRosim**, *occ**.

NaturalLanguageProcessing *KoNLP*, *RcmdrPlugin.TextMining*, *TextRegression*, *koRpus*, *maxent*, *tau*, *textcat*, *textir*, *tm.plugin.dc*, *tm.plugin.mail*, *topicmodels*, *wordcloud*.

OfficialStatistics *FFD*, *pxR*.

Optimization *GenSA*, *Rmallschains*, *clpAPI*, *cplexAPI*, *crs*, *dfoptim*, *glpkAPI*, *hydroPSO*, *mcga*, *nloptr*, *powell*.

Phylogenetics *SYNCSA*, *auteur*, *paleotree*, *pmc*.

Psychometrics *CDM*, *MPTinR*, *MplusAutomation*, *betareg*, *irtrees*, *pathmox*, *plspm*.

ReproducibleResearch *RExcelInstaller*, *SWordInstaller*, *knitr*, *rtf*, *tables*.

Spatial *McSpatial*, *geospt*, *rangeMapper*, *vec2dtrans*.

Survival *BaSTA*, *CPE*, *CPHshape*, *OIsurv*, *RobustAFT*, *SurvGini*, *ahaz*, *asbio*, *bwsurvival*, *dcens*, *dynpred*, *dynsurv*, *flexsurv*, *kaps*, *log-concens*, *parfm*, *riskRegression*, *rtv*, *survC1*, *tlmec*.

TimeSeries *Interpol.T*, *RMAWGEN*, *deseasonalize*, *quantspec*, *tempdisagg*.

gR *abn*, *gRim*.

(* = core package)

New contributed packages

AGD Analysis of Growth Data. Author: Stef van Buuren.

ARTIVA Time-varying DBN inference with the ARTIVA (Auto Regressive Time Varying) model. Authors: S. Lebre and G. Lelandais.

Ace Assay-based Cross-sectional Estimation of incidence rates. Authors: Brian Claggett, Weiliang Qiu, Rui Wang.

AdaptFitOS Adaptive Semiparametric Regression with Simultaneous Confidence Bands. Authors: Manuel Wiesenfarth and Tatyana Krivobokova.

AmericanCallOpt This package includes pricing function for selected American call options with underlying assets that generate payouts. Author: Paolo Zagaglia. In view: *Finance*.

AssotesteR Statistical Tests for Genetic Association Studies. Author: Gaston Sanchez.

BBmisc Miscellaneous helper functions for B. Bischl. Authors: Bernd Bischl, Michel Lang, Olaf Mersmann.

BCEA Bayesian Cost Effectiveness Analysis. Author: Gianluca Baio.

- BINCO** Bootstrap Inference for Network COstruction. Authors: Shuang Li, Li Hsu, Jie Peng, Pei Wang.
- BVS** Bayesian Variant Selection: Bayesian Model Uncertainty Techniques for Genetic Association Studies. Author: Melanie Quintana.
- BaSAR** Bayesian Spectrum Analysis in R. Authors: Emma Granqvist, Matthew Hartley and Richard J Morris.
- BatchExperiments** Statistical experiments on batch computing clusters. Authors: Bernd Bischl, Michel Lang, Olaf Mersmann. In view: *HighPerformanceComputing*.
- BatchJobs** Batch computing with R. Authors: Bernd Bischl, Michel Lang, Olaf Mersmann. In view: *HighPerformanceComputing*.
- BayesLCA** Bayesian Latent Class Analysis. Authors: Arthur White and Brendan Murphy.
- BayesLogit** Logistic Regression. Authors: Nicolas Polson, James G. Scott, and Jesse Windle.
- BayesSingleSub** Computation of Bayes factors for interrupted time-series designs. Authors: Richard D. Morey, Rivka de Vries.
- BayesXsrc** R Package Distribution of the BayesX C++ Sources. Authors: Daniel Adler [aut, cre], Thomas Kneib [aut], Stefan Lang [aut], Nikolaus Umlauf [aut], Achim Zeileis [aut].
- BerkeleyEarth** Data Input for Berkeley Earth Surface Temperature. Author: Steven Moshier.
- BrailleR** Improved access for blind useRs. Author: Jonathan Godfrey.
- BurStFin** Burns Statistics Financial. Author: Burns Statistics. In view: *Finance*.
- CALINE3** R interface to Fortran 77 implementation of CALINE3. Author: David Holstius.
- CARBayes** Spatial areal unit modelling. Author: Duncan Lee.
- CARE1** Statistical package for population size estimation in capture-recapture models. Author: T.C. Hsieh.
- CARramps** Reparameterized and marginalized posterior sampling for conditional autoregressive models. Authors: Kate Cowles and Stephen Bonett; with thanks to Juan Cervantes, Dong Liang, Alex Sawyer, and Michael Sedorff.
- CAScaling** CA scaling in R. Author: J. H. Straat.
- CPHshape** Find the maximum likelihood estimator of the shape constrained hazard baseline and the effect parameters in the Cox proportional hazards model. Authors: Rihong Hui and Hanna Jankowski. In view: *Survival*.
- CPMCGLM** Correction of the pvalue after multiple coding. Authors: Jeremie Riou, Amadou Diakite, and Benoit Liquet.
- CUMP** Analyze Multivariate Phenotypes by Combining Univariate results. Authors: Xuan Liu and Qiong Yang.
- CePa** Centrality-based pathway enrichment. Author: Zuguang Gu.
- ChoiceModelR** Choice Modeling in R. Authors: Ryan Sermas, assisted by John V. Colias.
- CoClust** Copula based cluster analysis. Authors: Francesca Marta Lilja Di Lascio, Simone Gianerini. In view: *Cluster*.
- Compounding** Computing Continuous Distributions. Authors: Bozidar V. Popovic, Saralees Nadarajah, Miroslav M. Ristic.
- ConjointChecks** A package to check the cancellation axioms of conjoint measurement. Author: Ben Domingue.
- CpGassoc** Association between Methylation and a phenotype of interest. Authors: Barfield, R., Conneely, K., Kilaru, V.
- CrypticIBDcheck** Identifying cryptic relatedness in genetic association studies. Authors: Annick Joelle Nembot-Simo, Jinko Graham and Brad McNeney.
- DDD** Diversity-dependent diversification. Author: Rampal S. Etienne & Bart Haegeman.
- DIRECT** Bayesian Clustering of Multivariate Data Under the Dirichlet-Process Prior. Authors: Audrey Qiuyan Fu, Steven Russell, Sarah J. Bray and Simon Tavaré.
- DSL** Distributed Storage and List. Authors: Ingo Feinerer [aut], Stefan Theussl [aut, cre], Christian Buchta [ctb].
- DandEFA** Dandelion Plot for R-mode Exploratory Factor Analysis. Authors: Artur Manukyan, Ahmet Sedef, Erhan Cene, Ibrahim Demir (Advisor).
- DeducerSpatial** Deducer for spatial data analysis. Authors: Ian Fellows and Alex Rickett with contributions from Neal Fultz.
- DetSel** A computer program to detect markers responding to selection. Author: Renaud Vitalis.

- DiscreteLaplace** Discrete Laplace distribution. Authors: Alessandro Barbiero, Riccardo Inchin-golo.
- Distance** A simple way to fit detection functions to distance sampling data and calculate abundance/density for biological populations. Author: David L. Miller.
- ETLUtils** Utility functions to execute standard ETL operations (using package **ff**) on large data. Author: Jan Wijffels.
- EcoTroph** EcoTroph modelling support. Authors: J. Guitton and M. Colleter, D. Gascuel.
- EffectStars** Visualization of Categorical Response Models. Author: Gunther Schauburger.
- EpiEstim** EpiEstim: a package to estimate time varying reproduction numbers from epidemic curves. Author: Anne Cori.
- EstSimPDMP** Estimation of the jump rate and simulation for piecewise-deterministic Markov processes. Author: Romain Azais.
- Exact** Exact Unconditional Tests for 2x2 Tables. Author: Peter Calhoun.
- ExomeDepth** Calls CNV from exome sequence data. Author: Vincent Plagnol.
- ExpDes** Experimental Designs package. Authors: Eric Batista Ferreira, Portya Piscitelli Cavalcanti, Denismar Alves Nogueira.
- FacPad** Bayesian Sparse Factor Analysis model for the inference of pathways responsive to drug treatment. Author: Haisu Ma.
- FactMixtAnalysis** Factor Mixture Analysis with covariates. Author: Cinzia Viroli.
- Familias** Probabilities for Pedigrees given DNA data. Authors: Petter Mostad and Thore Ege-land.
- FastImputation** Learn from training data then quickly fill in missing data. Author: Stephen R. Haptonstahl.
- FinAsym** Classifies implicit trading activity from market quotes and computes the probability of informed trading. Author: Paolo Zagaglia. In view: *Finance*.
- ForeCA** ForeCA — Forecastable Component Analysis. Author: Georg M. Goerg.
- G2Sd** Grain-size Statistics and Description of Sediment. Authors: Regis K. Gallon, Jerome Fournier.
- GOGANPA** GO-Functional-Network-based Gene-Set-Analysis. Author: Billy Chang.
- GPvam** Maximum Likelihood Estimation of the Generalized Persistence Value-Added Model. Authors: Andrew Karl, Yan Yang, and Sharon Lohr.
- GUniFrac** Generalized UniFrac distances. Author: Jun Chen.
- GenOrd** Simulation of ordinal and discrete variables with given correlation matrix and marginal distributions. Authors: Alessandro Barbiero, Pier Alda Ferrari.
- GeoLight** Analysis of light based geolocator data. Authors: Simeon Lisovski, Silke Bauer, Tamara Emmenegger.
- Grid2Polygons** Convert Spatial Grids to Polygons. Author: Jason C. Fisher.
- Hlest** Hybrid index estimation. Author: Ben Fitzpatrick.
- HMMmix** The HMMmix (HMM with mixture of gaussians as emission distribution) package. Authors: Stevonn Volant and Caroline Berard.
- Holidays** Holiday and halfday data, for use with the TimeWarp package. Author: Lars Hansen.
- IC2** Inequality and Concentration Indices and Curves. Author: Didier Plat.
- IPMpack** Builds and analyses Integral Projection Models (IPMs). Authors: CJE Metcalf, SM McMahon, R Salguero-Gomez, E Jongejans.
- ISBF** Iterative Selection of Blocks of Features — ISBF. Author: Pierre Alquier.
- IgorR** Read binary files saved by Igor Pro (including Neuromatic data). Author: Greg Jefferis.
- Interpol.T** Hourly interpolation of multiple temperature daily series. Author: Emanuele Eccel & Emanuele Cordano. In views: *Environmetrics*, *TimeSeries*.
- JMLSD** Joint Modeling of Longitudinal and Survival Data — Power Calculation. Authors: Emil A. Cornea, Liddy M. Chen, Bahjat F. Qaqish, Haitao Chu, and Joseph G. Ibrahim.
- JPSurv** Methods for population-based cancer survival analysis. Author: Yongwu Shao.
- Jmisc** Julian Miscellaneous Function. Author: TszKin Julian Chan.
- JohnsonDistribution** Johnson Distribution. Authors: A.I. McLeod and Leanna King.
- LCFdata** Data sets for package **LMERConvenience-Functions**. Authors: Antoine Tremblay.

- LIHNPSD** Poisson Subordinated Distribution. Author: Stephen Horng-Twu Lihn.
- LOST** Missing morphometric data simulation and estimation. Authors: J. Arbour and C. Brown.
- LTR** Perform LTR analysis on microarray data. Author: Paul C. Boutros.
- Lambda4** Estimation techniques for the reliability estimate: Maximized Lambda4. Author: Tyler Hunt.
- LeafAngle** Fits, plots, and summarizes leaf angle distributions. Author: Remko Duursma.
- LinearizedSVR** Linearized Support Vector Regression. Authors: Sriharsha Veeramachaneni and Ken Williams.
- Luminescence** Package for Luminescence Dating data analysis. Authors: Sebastian Kreutzer [aut, cre], Christoph Schmidt [aut, ctb], Margret C. Fuchs [aut, ctb], Michael Dietze [aut, ctb], Manfred Fischer [aut, ctb], Markus Fuchs [ths].
- MAMS** Designing Multi-Arm Multi-Stage Studies. Authors: Thomas Jaki and Dominic Magirr.
- MAVTgsa** Ordinary least square test and Multivariate Analysis Of Variance test with n contrasts. Authors: Chih-Yi Chien, Chen-An Tsai, Ching-Wei Chang, and James J. Chen.
- MImix** Mixture summary method for multiple imputation. Authors: Russell Steele, Naisyin Wang, and Adrian Raftery. In view: *Official-Statistics*.
- MLEP** Maximum likelihood estimate of penetrance parameters. Author: Yuki Sugaya.
- MM** The multiplicative multinomial distribution. Authors: Robin K. S. Hankin and P. M. E. Altham.
- MRCE** Author: Adam J. Rothman.
- McSpatial** Nonparametric spatial data analysis. Author: Daniel McMillen. In view: *Spatial*.
- MetaDE** Meta analysis of multiple microarray data. Authors: Jia Li and Xingbin Wang.
- MetaPath** Perform the Meta-Analysis for Pathway Enrichment analysis (MAPE). Authors: Kui Shen and Geroge Tseng.
- MixMod** Analysis of Mixed Models. Authors: Alexandra Kuznetsova, Per Bruun Brockhoff.
- Mobilize** Mobilize plots and functions. Author: Jeroen Ooms.
- Momocs** Shape Analysis of Outlines. Authors: Vincent Bonhomme, Sandrine Picq, Julien Claude.
- MorseGen** Simple raw data generator based on user-specified summary statistics. Author: Brendan Morse.
- MultiOrd** Generation of multivariate ordinal variates. Authors: Anup Amatya and Hakan Demirtas.
- NHPoisson** Modelling and validation of non homogeneous Poisson processes. Author: Ana C. Cebrian.
- NPMPM** tertiary probabilistic model in predictive microbiology for use in food manufacture. Author: Nadine Schoene.
- NbClust** An examination of indices for determining the number of clusters. Authors: Malika Charrad and Nadia Ghazzali and Veronique Boiteau and Azam Niknafs.
- NetComp** Network Generation and Comparison. Authors: Shannon M. Bell, Lyle D. Burgoon.
- NetPreProc** NetPreProc: Network Pre-Processing and normalization. Authors: Giorgio Valentini.
- NlsyLinks** Utilities and kinship information for Behavior Genetics and Developmental research using the NLSY. Authors: Will Beasley, Joe Rodgers, David Bard, and Kelly Meredith.
- NormalGamma** Normal-gamma convolution model. Authors: S. Plancade and Y. Rozenholc.
- Oidata** Data sets and supplements (OpenIntro). Authors: Andrew P Bray and David M Diez.
- OIsurv** Survival analysis supplement to OpenIntro guide. Author: David M Diez. In view: *Survival*.
- OLScurve** OLS growth curve trajectories. Authors: Phil Chalmers and Carrie Smith and Matthew Sigal.
- OOmisc** Ozgur-Ozlem Miscellaneous. Authors: Ozgur Asar, Ozlem Ilk.
- Ohmage** R Client for Mobilize/Andwellness server. Author: Jeroen Ooms.
- OneHandClapping** Prediction of condition-specific transcription factor interactions. Author: Sebastian Dümcke.
- OpenStreetMap** Access to open street map raster images. Authors: Ian Fellows, using the JMapView library by Jan Peter Stotz.
- OptimalCutpoints** Computing optimal cutpoints in diagnostic tests. Authors: Monica Lopez-Raton, Maria Xose Rodriguez-Alvarez.
- OptionsPdf** This package estimates a mix of log-normal distributions from interest-rate option data. Author: Paolo Zagaglia.

- PEIP** Functions for Aster Book on Inverse Theory. Author: Jonathan M. Lees.
- PIPS** Predicted Interval Plots. Authors: Daniel G. Muenz, Ray Griner, Huichao Chen, Lijuan Deng, Sachiko Miyahara, and Scott R. Evans, with contributions from Lingling Li, Hajime Uno, and Laura M. Smeaton. In view: *ClinicalTrials*.
- PKPDmodels** Pharmacokinetic/pharmacodynamic models. Authors: Anne Dubois, Julie Bertand, France Mentre and Douglas Bates.
- PairedData** Paired Data Analysis. Author: Stephane Champely.
- ParamHelpers** Helpers for parameters in black-optimization, tuning and machine learning. Authors: Bernd Bischl, Patrick Koch.
- PopGenome** Population genetic analysis. Author: Bastian Pfeifer.
- PrivateLR** Differentially private regularized logistic regression. Author: Staal A. Vinterbo.
- QRM** Provides R-language code to examine Quantitative Risk Management concepts. Author: Bernhard Pfaff.
- QUIC** Regularized sparse inverse covariance matrix estimation. Authors: Cho-Jui Hsieh [aut], Matyas A. Sustik [aut, cre], Inderjit S. Dhillon [aut], Pradeep Ravikumar [aut].
- QuasiSeq** Author: Steve Lund.
- R.devices** Enhanced methods for handling graphical devices. Author: Henrik Bengtsson.
- R2BayesX** Estimate Structured Additive Regression Models with BayesX. Authors: Nikolaus Umlauf [aut, cre], Thomas Kneib [aut], Stefan Lang [aut], Achim Zeileis [aut].
- R2G2** Converting R CRAN outputs into Google Earth. Author: Nils Arrigo.
- R2OpenBUGS** Running OpenBUGS from R. Authors: originally written as R2WinBUGS by Andrew Gelman; changes and packaged by Sibylle Sturtz and Uwe Ligges. With considerable contributions by Gregor Gorjanc and Jouni Kerman. Adapted to **R2OpenBUGS** from **R2WinBUGS** by Neal Thomas.
- R2admb** ADMB to R interface functions. Authors: Ben Bolker, Hans Skaug.
- R330** An R package for Stats 330. Authors: Alan Lee, Blair Robertson.
- RAFM** Admixture F-model. Authors: Markku Karhunen, Uni. Helsinki.
- RCALI** Calculation of the Integrated Flow of Particles between Polygons. Authors: Annie Bouvier, Kien Kieu, Kasia Adamczyk, and Herve Monod.
- RCassandra** R/Cassandra interface. Author: Simon Urbanek.
- RForcecom** RForcecom provides the connection to Force.com (Salesforce.com) from R. Author: Takekatsu Hiramura.
- RGIFT** Create quizzes in GIFT Format. Authors: María José Haro-Delicado, Virgilio Gómez-Rubio and Francisco Parreño-Torres.
- RIFS** Random Iterated Function System (RIFS). Authors: Pavel V. Moskalev, Alexey G. Bukhovets and Tatyana Ya. Biruchinskay.
- RMallow** Fit Multi-Modal Mallows' Models to ranking data. Author: Erik Gregory.
- RMendeley** Interface to Mendeley API methods. Authors: Carl Boettiger, Duncan Temple Lang.
- RSeed** borenstein analysis. Author: Claus Jonathan Fritzscheier.
- RTDAmeritrade** Author: Theodore Van Rooy.
- RcmdrPlugin.EBM** Rcmdr Evidence Based Medicine Plug-In package. Author: Daniel-Corneliu Leucuta.
- RcmdrPlugin.KMggplot2** Rcmdr Plug-In for Kaplan-Meier Plot and Other Plots by Using the ggplot2 Package. Authors: Triad sou. and Kengo NAGASHIMA.
- RcmdrPlugin.SCDA** Rcmdr plugin for designing and analyzing single-case experiments. Authors: Isis Bulte and Patrick Onghena.
- RcmdrPlugin.UCA** UCA Rcmdr Plug-in. Author: and Manuel Munoz-Marquez.
- RcmdrPlugin.doBy** Rcmdr doBy Plug-In. Author: Jonathan Lee.
- RcmdrPlugin.pointG** Rcmdr Graphical POINT of view for questionnaire data Plug-In. Author: Stephane Champely.
- RcppSMC** Rcpp bindings for Sequential Monte Carlo. Authors: Dirk Eddelbuettel and Adam M. Johansen.
- Rdistance** Distance sampling analyses. Author: Trent McDonald.
- ReCiPa** Redundancy Control in Pathways databases. Author: Juan C. Vivar.
- RenextGUI** GUI for Renext. Authors: Yves Deville and IRSN.

- RfmriVC** Varying stimulus coefficient fMRI models in R. Authors: Ludwig Bothmann, Stefanie Kalus.
- Rmalschains** Continuous Optimization using Memetic Algorithms with Local Search Chains (MA-LS-Chains) in R. Authors: Christoph Bergmeir, Daniel Molina, José M. Benítez. In view: *Optimization*.
- Rmisc** Rmisc: Ryan Miscellaneous. Author: Ryan M. Hope.
- Rmixmod** An interface of MIXMOD. Authors: Remi Lebet and Serge Iovleff and Florent Langrognet, with contributions from C. Biernacki and G. Celeux and G. Govaert.
- Rquake** Seismic Hypocenter Determination. Author: Jonathan M. Lees.
- RxCeolInf** R x C Ecological Inference With Optional Incorporation of Survey Information. Authors: D. James Greiner, Paul Baines, and Kevin M. Quinn. In view: *Bayesian*.
- SAScii** Import ASCII files directly into R using only a SAS input script. Author: Anthony Joseph Damico.
- SCMA** Single-Case Meta-Analysis. Authors: Isis Bulte and Patrick Onghena.
- SCRT** Single-Case Randomization Tests. Authors: Isis Bulte and Patrick Onghena.
- SCVA** Single-Case Visual Analysis. Authors: Isis Bulte and Patrick Onghena.
- SCperf** Supply Chain Perform. Author: Marlene Silva Marchena.
- SEER2R** reading and writing SEER*STAT data files. Author: Jun Luo.
- SGL** Fit a GLM (or cox model) with a combination of lasso and group lasso regularization. Authors: Noah Simon, Jerome Friedman, Trevor Hastie, and Rob Tibshirani.
- SGPdata** Exemplar data sets for SGP analyses. Authors: Damian W. Betebenner, Adam Van Iwaarden and Ben Domingue.
- SKAT** SNP-set (Sequence) Kernel Association Test. Authors: Seunggeun Lee, Larisa Miropolsky and Micheal Wu.
- SNSEquate** Standard and Nonstandard Statistical Models and Methods for Test Equating. Author: Jorge Gonzalez Burgos.
- SPA3G** SPA3G: R package for the method of Li and Cui (2012). Authors: Shaoyu Li and Yuehua Cui.
- SPIn** Optimal Shortest Probability Intervals. Author: Ying Liu.
- SPSL** Site Percolation on Square Lattice (SPSL). Author: Pavel V. Moskalev.
- SRMA** SRMA Analysis of Array-based Sequencing Data. Authors: Wenyi Wang, Nianxiang Zhang, Yan Xu.
- SemiParSampleSel** Semiparametric Sample Selection Modelling with Continuous Response. Authors: Giampiero Marra and Rosalba Radice.
- SightabilityModel** Wildlife Sightability Modeling. Author: John Fieberg.
- Simile** Interact with Simile models. Author: Simulistics Ltd.
- SoilR** Models of Soil Organic Matter Decomposition. Authors: Carlos A. Sierra, Markus Mueller.
- SparseGrid** Sparse grid integration in R. Author: Jelmer Ypma.
- SpatialTools** Author: Joshua French.
- SteinerNet** Steiner approach for Biological Pathway Graph Analysis. Authors: Afshin Sadeghi, Holger Froehlich.
- StressStrength** Computation and estimation of reliability of stress-strength models. Authors: Alessandro Barbiero, Riccardo Inchingolo.
- StructR** Structural geology tools. Author: Jeffrey R. Webber.
- SunterSampling** Sunter's sampling design. Authors: Alessandro Barbiero, Giancarlo Manzi.
- TCC** TCC: tag count comparison package. Authors: Koji Kadota, Tomoaki Nishiyama, Kentaro Shimizu.
- TPAM** Author: Yuping Zhang.
- TTAinterfaceTrendAnalysis** Temporal Trend Analysis Graphical Interface. Authors: David Devreker [aut], Alain Lefebvre [aut, cre].
- TUWmodel** Lumped hydrological model developed at the Vienna University of Technology for education purposes. Authors: Juraj Parajka, Alberto Viglione.
- Taxonstand** Taxonomic standardization of plant species names. Author: Luis Cayuela.
- TestSurvRec** Statistical tests to compare two survival curves with recurrent events. Author: Dr. Carlos Martinez.

- ThresholdROC** Optimum threshold estimation based on cost function in a two and three state setting. Author: Konstantina Skaltsa.
- TimeWarp** Date calculations and manipulation. Authors: Tony Plate, Jeffrey Horner, Lars Hansen.
- VarEff** Variation of effective population size. Authors: Natacha Nikolic and Claude Chevalet.
- VarSwapPrice** Pricing a variance swap on an equity index. Author: Paolo Zagaglia. In view: *Finance*.
- VariABEL** Testing of genotypic variance heterogeneity to detect potentially interacting SNP. Author: Maksim Struchalin.
- Voss** Generic Voss algorithm (random sequential additions). Author: Pavel V. Moskalev.
- WeightedPortTest** Weighted Portmanteau Tests for Time Series Goodness-of-fit. Authors: Thomas J. Fisher and Colin M. Gallagher.
- WideLM** Fitting many skinny linear models to a single data set. Authors: Mark Seligman, with contributions from Chris Fraley. In view: *High-PerformanceComputing*.
- acer** The ACER Method for Extreme Value Estimation. Author: Even Haug.
- acs** Download and manipulate data from the US Census American Community Survey. Author: Ezra Haber Glenn.
- adagio** Discrete and Global Optimization Routines. Author: Hans W Borchers.
- adaptMCMC** Implementation of a generic adaptive Monte Carlo Markov Chain sampler. Authors: Andreas Scheidegger.
- ageprior** Prior distributions for molecular dating. Author: Michael Matschiner.
- anametrix** Connects to Anametrix. Author: Roman Jugai.
- anoint** Analysis of interactions. Authors: Ravi Varadhan and Stephanie Kovalchik.
- appell** Compute Appell's F1 hypergeometric function. Authors: Daniel Sabanes Bove with contributions by F. D. Colavecchia, R. C. Forrey, G. Gasaneo, N. L. J. Michel, L. F. Shampine, M. V. Stoitsov and H. A. Watts.
- apple** Approximate Path for Penalized Likelihood Estimators. Authors: Yi Yu, Yang Feng.
- assertive** Readable check functions to ensure code integrity. Authors: Richard Cotton [aut, cre].
- awsMethods** Class and Methods definitions for packages **aws**, **adimpro**, **fmri**, **dwi**. Author: Joerg Polzehl.
- bams** Breakpoint annotation model smoothing. Author: Toby Dylan Hocking.
- bandit** Functions for simple A/B split test and multi-armed bandit analysis. Author: Thomas Lotze.
- base64** Base 64 encoder/decoder. Authors: Romain Francois, based on code by Bob Trower available at <http://base64.sourceforge.net/>.
- bayesMCCLust** Mixtures-of-Experts Markov Chain Clustering and Dirichlet Multinomial Clustering. Author: Christoph Pamminger.
- bayesPop** Probabilistic Population Projection. Authors: Hana Sevcikova, Adrian Raftery.
- bcrm** Bayesian continuous reassessment method (CRM) designs for Phase I dose-finding trials. Author: Michael Sweeting. In view: *Clinical-Trials*.
- bda** Algorithms for Birth Data Analysis. Author: Bin Wang.
- betafam** Detecting rare variants for quantitative traits using nuclear families. Author: Wei Guo.
- betapart** Partitioning beta diversity into turnover and nestedness components. Authors: Andres Baselga and David Orme.
- bigdata** Big Data Analytics. Authors: Han Liu, Tuo Zhao.
- bigml** R bindings for the BigML API. Authors: Justin Donaldson [aut, cre].
- binomlogit** Efficient MCMC for Binomial Logit Models. Author: Agnes Fussl.
- bionetdata** Biological and chemical data networks. Authors: Matteo Re.
- bisectr** Tools to find bad commits with git bisect. Author: Winston Chang.
- bit64** A S3 class for vectors of 64bit integers. Author: Jens Oehlschlägel.
- biwavelet** Conduct univariate and bivariate wavelet analyses. Authors: Tarik C. Gouhier, Aslak Grinsted.
- bmp** Read Windows Bitmap (BMP) images. Author: Gregory Jefferis.
- boss** Boosted One-Step Statistics: Fast and accurate approximations for GLM, GEE and Mixed models for use in GWAS. Author: Arend Voorman.

- boussinesq** Analytic Solutions for (ground-water) Boussinesq Equation. Author: Emanuele Cordano. In view: *Environmetrics*.
- bpkde** Back Projected Kernel Density Estimation. Authors: Kjell Konis, Victor Panaretos.
- catdata** Categorical Data. Authors: Gerhard Tutz, Gunther Schaubberger.
- cec2005benchmark** Benchmark for the CEC 2005 Special Session on Real-Parameter Optimization. Authors: Yasser González-Fernández and Marta Soto.
- cepp** Context Driven Exploratory Projection Pursuit. Author: Mohit Dayal.
- cin** Causal Inference for Neuroscience. Authors: Xi (Rossi) LUO with contributions from Dylan Small, Chiang-shan Li, and Paul Rosenbaum.
- clhs** Conditioned Latin Hypercube Sampling. Author: Pierre Roudier.
- cloudRmpi** Cloud-based MPI Parallel Processing for R (cloudRmpi). Author: Barnet Wagman. In view: *HighPerformanceComputing*.
- cloudRmpiJars** Third-party jars for cloudRmpi. Author: Barnet Wagman. In view: *HighPerformanceComputing*.
- cloudUtil** Cloud Util Plots. Authors: Christian Panse, Ermir Qeli.
- coloc** Colocalisation tests of two genetic traits. Author: Chris Wallace.
- comparison** Multivariate likelihood ratio calculation and evaluation. Author: David Lucy.
- complex.surv.dat.sim** Simulation of recurrent event survival data. Authors: David Moríña, Centre Tecnològic de Nutrició i Salut and Albert Navarro.
- compound.Cox** Regression estimation based on the compound covariate method under the Cox proportional hazard model. Author: Takeshi Emura & Yi-Hau Chen.
- condmixt** Conditional Density Estimation with Neural Network Conditional Mixtures. Author: Julie Carreau.
- cpm** Sequential Parametric and Nonparametric Change Detection. Authors: Gordon J. Ross, incorporating code from the GNU Scientific Library.
- crblocks** Categorical Randomized Block Data Analysis. Authors: David Allingham, D.J. Best.
- crp.CSFP** CreditRisk+ portfolio model. Authors: Dr. Matthias Fischer, Kevin Jakob & Stefan Kolb. In view: *Finance*.
- crstep** Stepwise covariate selection for the Fine & Gray competing risks regression model. Author: Ravi Varadhan & Deborah Kuk.
- csound** Accessing Csound functionality through R. Author: Ethan Brown.
- cumplyr** Extends ddply to allow calculation of cumulative quantities. Author: John Myles White.
- curvclust** Curve clustering. Authors: Madison Giacofci, Sophie Lambert-Lacroix, Guillemette Marot, Franck Picard.
- dataframe** Fast data frames. Author: Tim Hesterberg.
- deTestSet** Testset for differential equations. Authors: Karline Soetaert, Jeff Cash, Francesca Mazzia. In view: *DifferentialEquations*.
- deltaPlotR** Identification of dichotomous differential item functioning (DIF) using Angoff's Delta Plot method. Authors: David Magis, Bruno Facon.
- depend.truncation** Inference for parametric and semiparametric models based on dependent truncation data. Authors: Takeshi Emura.
- deseasonalize** Optimal deseasonalization for geographical time series using AR fitting. Author: A. I. McLeod. In view: *TimeSeries*.
- dgmb** dgmb Simulating data for PLS structural models. Authors: Alba Martinez-Ruiz and Claudia Martinez-Araneda.
- diffEq** Functions from the book Solving Differential Equations in R. Author: Karline Soetaert.
- disclap** Discrete Laplace Family. Authors: Mikkel Meyer Andersen and Poul Svante Eriksen.
- disclapmix** Discrete Laplace mixture inference using the EM algorithm. Authors: Mikkel Meyer Andersen and Poul Svante Eriksen.
- discreteMTP** Multiple testing procedures for discrete test statistics. Authors: Ruth Heller [aut], Hadas Gur [aut], Shay Yaacoby [aut, cre].
- disp2D** 2D Hausdorff and Simplex Dispersion Orderings. Author: Guillermo Ayala.
- distory** Distance Between Phylogenetic Histories. Authors: John Chakerian and Susan Holmes. In view: *Phylogenetics*.
- dkDNA** Diffusion kernels on a set of genotypes. Authors: Gota Morota and Masanori Koyama.

- dma** Dynamic model averaging. Authors: Tyler H. McCormick, Adrian Raftery, David Madigan.
- doParallel** Foreach parallel adaptor for the parallel package. Author: Revolution Analytics.
- dostats** Compute statistics helper functions. Author: Andrew Redd.
- eigendog** EigenDog. Author: The EigenDog Team.
- endorse** R Package for Analyzing Endorsement Experiments. Authors: Yuki Shiraito, Kosuke Imai.
- events** Store and manipulate event data. Author: Will Lowe.
- evora** Epigenetic Variable Outliers for Risk prediction Analysis. Author: Andrew E Teschen-dorff.
- exptest** Tests for Exponentiality. Authors: Ruslan Pusev and Maxim Yakovlev.
- extraBinomial** Extra-binomial approach for pooled sequencing data. Author: Xin Yang.
- faisalconjoint** Faisal Conjoint Model: A New Approach to Conjoint Analysis. Authors: Faisal Afzal Siddiqui, Ghulam Hussain, and Mudassiruddin.
- fanc** Penalized likelihood factor analysis via non-convex penalty. Authors: Kei Hirose, Michio Yamamoto.
- fanovaGraph** Building Kriging models from FANOVA graphs. Authors: Jana Fruth, Thomas Muehlenstaedt, Olivier Roustant.
- faoutlier** Influential case detection methods for factor analysis and SEM. Author: Phil Chalmers.
- fastGHQuad** Fast Rcpp implementation of Gauss-Hermite quadrature. Author: Alexander W Blocker.
- fastcox** Lasso and elastic-net penalized Cox's regression in high dimensions models using the cocktail algorithm. Authors: Yi Yang, Hui Zou.
- fgof** Fast Goodness-of-fit Test. Authors: Ivan Kojadinovic and Jun Yan.
- filterviewR** Get started easily with FilterView from R. Author: Nelson Ray.
- fishmove** Prediction of Fish Movement Parameters. Author: Johannes Radinger.
- foodweb** visualisation and analysis of food web networks. Author: Giselle Perdomo.
- fracprolif** Fraction Proliferation via a quiescent growth model. Authors: Shawn Garbett, Darren Tyson.
- frmqa** Financial Risk Management and Quantitative Analysis. Author: Thanh T. Tran. In view: *Finance*.
- frt** Full Randomization Test. Authors: Giangiacomo Bravo, Lucia Tamburino.
- fume** FUME package. Author: Santander Meteorology Group.
- fwsim** Fisher-Wright Population Simulation. Authors: Mikkel Meyer Andersen and Poul Svante Eriksen.
- gammSlice** Generalized additive mixed model analysis via slice sampling. Authors: Tung Pham and Matt Wand.
- gcdnet** A generalized coordinate descent (GCD) algorithm for computing the solution path of the hybrid Huberized support vector machine (HHSVM) and its generalization. Authors: Yi Yang, Hui Zou.
- genomicper** Circular Genomic Permutation using GWAS p-values of association. Authors: Claudia P. Cabrera-Cardenas, Pau Navarro, Chris S. Haley.
- geospt** Spatial geostatistics; some geostatistical and radial basis functions, prediction and cross validation; design of optimal spatial sampling networks based on geostatistical modelling. Authors: Carlos Melo, Alí Santacruz, Oscar Melo and others. In view: *Spatial*.
- geotools** Geo tools. Author: Antoine Lucas.
- gglasso** Group Lasso Penalized Learning Using A Unified BMD Algorithm. Authors: Yi Yang, Hui Zou.
- gldist** An Asymmetry-Steepness Parameterization of the Generalized Lambda Distribution. Authors: Yohan Chalabi and Diethelm Wuertz.
- glmmGS** Gauss-Seidel Generalized Linear Mixed Model solver. Authors: Michele Morara, Louise Ryan, Subharup Guha, Christopher Paciorek.
- googlePublicData** An R library to build Google's Public Data Explorer DSPL Metadata files. Author: George Vega Yon.
- govdat** Interface to API methods for government data. Author: Scott Chamberlain.
- graphComp** Visual graph comparison. Authors: Khadija El Amrani, Ulrich Mansmann.
- gridDebug** Debugging Grid Graphics. Authors: Paul Murrell and Velvet Ly.

- gridGraphviz** Drawing Graphs with Grid. Author: Paul Murrell.
- growcurves** Bayesian semiparametric growth curve models that additionally include multiple membership random effects. Author: Terrance Savitsky.
- gsbDesign** Group Sequential Bayes Design. Authors: Florian Gerber, Thomas Gsponer.
- gsmaRt** Gene Set Microarray Testing. Authors: Stephan Artmann, Mathias Fuchs.
- gvcm.cat** Regularized Categorical Effects/Categorical Effect Modifiers in GLMs. Author: Margaret-Ruth Oelker.
- harvestr** A Parallel Simulation Framework. Author: Andrew Redd. In view: *HighPerformanceComputing*.
- hiPOD** hierarchical Pooled Optimal Design. Author: Wei E. Liang.
- hitandrunk** "Hit and Run" method for sampling uniformly from convex shapes. Author: Gert van Valkenhoef.
- hof** More higher order functions for R. Author: Hadley Wickham.
- httr** Tools for working with URLs and HTTP. Author: Hadley Wickham.
- hydroPSO** Model-Independent Particle Swarm Optimisation for Environmental Models. Authors: Mauricio Zambrano-Bigiarini [aut, cre] and Rodrigo Rojas [ctb]. In views: *Environmetrics*, *Optimization*.
- iDEMO** integrated degradation models. Authors: Ya-Shan Cheng, Chien-Yu Peng.
- iGasso** Genetic association tests. Author: Dr. Kai Wang.
- iSubpathwayMiner** The package can implement the graph-based reconstruction, analyses, and visualization of the KEGG pathways. Author: Chunquan Li.
- idbg** R debugger. Author: Ronen Kimchi.
- igraph0** Network analysis and visualization. Author: Gabor Csardi.
- igraphdata** A collection of network data sets for the igraph package. Author: Gabor Csardi.
- imputeYn** Imputing the last largest censored datum under weighted least squares. Authors: Hasinur Rahaman Khan and Ewart Shaw.
- intReg** Interval Regression. Author: Ott Toomet. In view: *Econometrics*.
- intpoint** linear programming solver by the interior point method and graphically (two dimensions). Author: Alejandro Quintela del Rio.
- irace** Iterated Racing Procedures. Authors: Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, Mauro Birattari, Eric Yuan and Prasanna Balaprakash.
- irtrees** Estimation of Tree-Based Item Response Models. Authors: Ivailo Partchev and Paul De Boeck. In view: *Psychometrics*.
- ivbma** Bayesian Instrumental Variable Estimation and Model Determination via Conditional Bayes Factors. Authors: Alex Lenkoski, Anna Karl, Andreas Neudecker.
- iwtp** Numerical evaluation willingness to pay based on interval data. Authors: Yuri Belyaev, Wenchao Zhou.
- jmec** Fit joint model for event and censoring with cluster-level frailties. Author: Willem M. van der Wal.
- joineR** Joint modelling of repeated measurements and time-to-event data. Authors: Pete Philipson, Ines Sousa, Peter Diggle, Paula Williamson, Ruwanthi Kolamunnage-Dona, Robin Henderson.
- kappaSize** Sample Size Estimation Functions for Studies of Interobserver Agreement. Author: Michael A Rotondi.
- kaps** K Adaptive Partitioning for Survival data. Authors: Soo-Heang Eo and HyungJun Cho. In view: *Survival*.
- kequate** The kernel method of test equating. Authors: Bjorn Andersson, Kenny Branberg and Marie Wiberg.
- knitr** A general-purpose package for dynamic report generation in R. Author: Yihui Xie. In view: *ReproducibleResearch*.
- knnGarden** Multi-distance based k-Nearest Neighbors. Author: Boxian Wei & Fan Yang & Xinniao Wang & Yanni Ge.
- koRpus** An R Package for Text Analysis. Authors: m.eik michalke, with contributions from Earl Brown, Alberto Mirisola, and Laura Hauser. In view: *NaturalLanguageProcessing*.
- kriging** Ordinary Kriging. Author: Omar E. Olmedo.
- ktspair** k-Top Scoring Pairs for Microarray Classification. Author: Julien Damond.
- labeledLoop** Labeled Loop. Author: Kohske Takahashi.

- lava** Linear Latent Variable Models. Author: Klaus K. Holst.
- lava.tobit** LVM with censored and binary outcomes. Author: Klaus K. Holst.
- leapp** latent effect adjustment after primary projection. Authors: Yunting Sun, Nancy R. Zhang, Art B. Owen.
- leiv** Bivariate linear errors-in-variables estimation. Author: David Leonard.
- lestat** A package for LEarning STATistics. Author: Petter Mostad.
- likelihood** Methods for maximum likelihood estimation. Author: Lora Murphy.
- lisp** List-processing à la SRFI-1. Author: Peter Dannenberg.
- lle** Locally linear embedding. Authors: Holger Diedrich, Dr. Markus Abel (Department of Physics, University Potsdam).
- lmbc** Linear Model Bias Correction for RNA-Seq Data. Author: David Dalpiaz.
- lmf** Functions for estimation and inference of selection in age-structured populations. Author: Thomas Kvalnes.
- logitnorm** Functions for the logitnormal distribution. Author: Thomas Wutzler.
- longclust** Model-Based Clustering and Classification for Longitudinal Data. Authors: P. D. McNicholas, K. Raju Jampani and Sanjeena Subedi.
- lqmm** Linear Quantile Mixed Models. Author: Marco Geraci.
- lsr** Companion to "Learning Statistics with R". Author: Daniel Navarro.
- lxb** Fast LXB file reader. Author: Björn Winckler.
- mada** Meta-Analysis of Diagnostic Accuracy (mada). Author: Philipp Doebler.
- makeProject** Creates an empty package framework for the LCFD format. Author: Noah Silverman.
- makeR** Package for managing projects with multiple versions derived from a single source repository. Author: Jason Bryer.
- malaria.em** EM Estimation of Malaria Haplotype Probabilities from Multiply Infected Human Blood Samples. Author: Xiaohong Li.
- markdown** Markdown rendering for R. Authors: JJ Allaire, Jeffrey Horner, Vicent Marti, and Nat-acha Porte.
- marqLevAlg** An algorithm for least-squares curve fitting. Authors: D. Commenges, M. Prague and Amadou Diakite.
- maxlike** Model species distributions by estimating the probability of occurrence using presence-only data. Authors: Richard Chandler and Andy Royle.
- mcbiopi** Matrix Computation Based Identification Of Prime Implicants. Author: Holger Schwender.
- mcmcse** Monte Carlo standard errors for MCMC. Author: James M. Flegal.
- mets** Analysis of Multivariate Event Times. Authors: Klaus K. Holst and Thomas Scheike.
- mgraph** Graphing map attributes and non-map variables in R. Author: George Owusu.
- miscFuncs** Miscellaneous Useful Functions. Author: Benjamin M. Taylor.
- mlgt** Multi-Locus Geno-Typing. Author: Dave T. Gerrard.
- mmand** Mathematical Morphology in Any Number of Dimensions. Author: Jon Clayden.
- mmds** Mixture Model Distance Sampling (mmds). Author: David Lawrence Miller.
- mmm2** Multivariate marginal models with shared regression parameters. Authors: Ozgur Asar, Ozlem Ilk.
- mmod** Modern measures of population differentiation. Author: David Winter.
- mobForest** Model based Random Forest analysis. Authors: Nikhil Garge and Barry Eggleston.
- mosaicManip** Project MOSAIC (mosaic-web.org) manipulate examples. Authors: Andrew Rich, Daniel Kaplan, Randall Pruim.
- mpMap** Multi-parent RIL genetic analysis. Author: Emma Huang.
- mpc** MPC — Multiple Precision Complex Library. Author: Murray Stokely.
- mpoly** Symbolic computation and more with multivariate polynomials. Author: David Kahle.
- mrds** Mark-Recapture Distance Sampling (mrds). Authors: Jeff Laake, David Borchers, Len Thomas, David Miller and Jon Bishop.
- mrp** Multilevel Regression and Poststratification. Authors: Andrew Gelman, Michael Malecki, Daniel Lee, Yu-Sung Su.

- mrpdata** Data for Multilevel Regression and Post-stratification. Authors: Andrew Gelman, Michael Malecki, Daniel Lee, Yu-Sung Su, Jiqiang Guo.
- mseapca** Metabolite set enrichment analysis for factor loading in principal component analysis. Author: Hiroyuki Yamamoto.
- msgps** Degrees of freedom of elastic net, adaptive lasso and generalized elastic net. Author: Kei Hirose.
- multivator** A multivariate emulator. Author: Robin K. S. Hankin.
- muma** Metabolomic Univariate and Multivariate Analysis. Authors: Edoardo Gaude, Francesca Chignola, Dimitrios Spiliotopoulos, Silvia Mari, Andrea Spitaleri and Michela Ghitti.
- myepisodes** MyEpisodes RSS/API functions. Author: Matt Malin.
- nadiv** Functions to construct non-additive genetic relatedness matrices. Author: Matthew Wolak.
- networkDynamic** Dynamic Extensions for Network Objects. Authors: Ayn Leslie-Cook, Zack Almquist, Pavel N. Krivitsky, Skye Bender-deMoll, David R. Hunter, Martina Morris, Carter T. Butts.
- neuroblastoma** Neuroblastoma. Author: Toby Dylan Hocking.
- nlmsn** Fitting nonlinear models with scale mixture of skew-normal distributions. Authors: Aldo Garay, Marcos Prates and Victor Lachos.
- oblique.tree** Oblique Trees for Classification Data. Author: Alfred Truong. In view: *MachineLearning*.
- occ** Estimates PET neuroreceptor occupancies. Author: Joaquim Radua. In view: *MedicalImaging*.
- okmesonet** Retrieve Oklahoma Mesonet climatological data. Authors: Brady Allred, Torre Hovick, Samuel Fuhlendorf.
- opencpu.demo** OpenCPU Demo apps. Authors: Jeroen Ooms and others as specified in the manual page.
- opm** Tools for analysing OmniLog(R) Phenotype Microarray data. Authors: Markus Goeker, with contributions by Lea A.I. Vaas, Johannes Sikorski, Nora Buddruhs and Anne Fiebig.
- ops** Optimal Power Space Transformation. Author: Micha Sammeth.
- p2distance** Welfare's Synthetic Indicator. Authors: A.J. Perez-Luque; R. Moreno; R. Perez-Perez and F.J. Bonet.
- pacose** iPACOSE, PACOSE and other methods for covariance selection. Authors: Vincent Guillemot, Andreas Bender.
- pairedCI** Confidence intervals for the ratio of locations and for the ratio of scales of two paired samples. Authors: Cornelia Froemke, Ludwig Hothorn and Michael Schneider.
- pairheatmap** A tool for comparing heatmaps. Author: Xiaoyong Sun.
- paleotree** Paleontological and Phylogenetic Analyses of Evolution. Author: David Bapst. In view: *Phylogenetics*.
- parfm** Parametric Frailty Models. Authors: Federico Rotolo and Marco Munda. In view: *Survival*.
- pcaL1** Three L1-Norm PCA Methods. Authors: Sapan Jot and Paul Brooks.
- pcaPA** Parallel Analysis for ordinal and numeric data using polychoric and Pearson correlations with S3 classes. Authors: Carlos A. Arias and Víctor H. Cervantes.
- pcenum** Permutations and Combinations Enumeration. Author: Benjamin Auder.
- pdC** Permutation Distribution Clustering. Author: Andreas M. Brandmaier.
- pgmm** Parsimonious Gaussian mixture models. Authors: Paul McNicholas, Raju Jampani, Aaron McDaid, Brendan Murphy, Larry Banks.
- pgnorm** The p-generalized normal distribution. Author: Steve Kalke.
- phcfM** phcfM R package. Author: Ghislain Vieilledent.
- phenology** Tools to manage a parametric function that describes phenology. Author: Marc Girondot.
- phia** Post-Hoc Interaction Analysis. Author: Helios De Rosario-Martinez.
- phom** Persistent Homology in R. Author: Andrew Tausz.
- pkgmaker** Package development utilities. Author: Renaud Gaujoux.
- plmDE** Additive partially linear models for differential gene expression analysis. Author: Jonas Mueller.
- pmc** Phylogenetic Monte Carlo. Author: Carl Boettiger. In view: *Phylogenetics*.
- pmclust** Parallel Model-Based Clustering. Authors: Wei-Chen Chen, George Ostrouchov. In views: *Cluster*, *HighPerformanceComputing*.

- polywog** Bootstrapped Basis Regression with Oracle Model Selection. Authors: Brenton Kenkel and Curtis S. Signorino.
- popdemo** Provides Tools For Demographic Modelling Using Projection Matrices. Authors: Iain Stott, Dave Hodgson, Stuart Townley.
- postCP** A package to estimate posterior probabilities in change-point models using constrained HMM. Authors: Gregory Nuel and The Minh Luong.
- qLearn** Estimation and inference for Q-learning. Authors: Jingyi Xin, Bibhas Chakraborty, and Eric B. Laber.
- qmap** Quantile-mapping for post-processing climate model output. Author: Lukas Gudmundsson.
- qtbase** Interface between R and Qt. Authors: Michael Lawrence, Deepayan Sarkar.
- qtlhot** Inference for QTL Hotspots. Authors: Elias Chaibub Neto and Brian S Yandell.
- qtlnet** Causal Inference of QTL Networks. Authors: Elias Chaibub Neto and Brian S. Yandell.
- qtpaint** Qt-based painting infrastructure. Authors: Michael Lawrence, Deepayan Sarkar.
- qtutils** Miscellaneous Qt-based utilities. Author: Deepayan Sarkar.
- quantspec** Quantile-based Spectral Analysis of Time Series. Author: Tobias Kley. In view: *Time-Series*.
- rBeta2009** The Beta Random Number and Dirichlet Random Vector Generating Functions. Authors: Ching-Wei Cheng, Ying-Chao Hung, Narayanaswamy Balakrishnan.
- rFerns** Random ferns classifier. Author: Miron B. Kursa.
- rSFA** Slow Feature Analysis in R. Authors: Wolfgang Konen, Martin Zaefferer, Patrick Koch; Bug hunting and testing by Ayodele Fasika, Ashwin Kumar, Prawyn Jebakumar.
- rapport** A report templating system. Authors: Aleksandar Blagotić and Gergely Daróczy.
- rbamtools** Reading, manipulation and writing BAM (Binary alignment) files. Author: Wolfgang Kaisers.
- rcppbugs** R binding for cppbugs. Author: Whit Armstrong.
- rdryad** Dryad API interface. Authors: Scott Chamberlain [aut, cre], Carl Boettiger [aut], Karthik Ram [aut].
- readMLData** Reading machine learning benchmark data sets from different sources in their original format. Author: Petr Savicky.
- readbitmap** Read bitmap images (BMP, JPEG, PNG) without external dependencies. Author: Gregory Jefferis.
- reams** Resampling-Based Adaptive Model Selection. Authors: Philip Reiss and Lei Huang.
- rehh** Searching for footprints of selection using Haplotype Homozygosity based tests. Authors: Mathieu Gautier and Renaud Vitalis.
- relSim** Relative Simulator. Author: James M. Curran.
- replicationDemos** Agnotology. Author: Rasmus E. Benestad.
- represent** Determine the representativity of two multidimensional data sets. Author: Harmen Draisma.
- rgbif** Interface to the Global Biodiversity Information Facility API methods. Authors: Scott Chamberlain [aut, cre], Carl Boettiger [aut], Karthik Ram [aut].
- ri** ri: R package for performing randomization-based inference for experiments. Authors: Peter M. Aronow and Cyrus Samii.
- risaac** A fast cryptographic random number generator using ISAAC by Robert Jenkins. Author: David L Gibbs.
- riskRegression** Risk regression for survival analysis. Authors: Thomas Alexander Gerds, Thomas Harder Scheike. In view: *Survival*.
- rknn** Random KNN Classification and Regression. Author: Shengqiao Li.
- rngSetSeed** Initialization of the R base random number generator Mersenne-Twister with a vector of an arbitrary length. Author: Petr Savicky.
- robustHD** Robust methods for high-dimensional data. Author: Andreas Alfons.
- robustfa** An Object Oriented Framework for Robust Factor Analysis. Author: Ying-Ying Zhang (Robert).
- rolasized** Solarized colours in R. Author: Christian Zang.
- ror** Robust Ordinal Regression MCDA library. Author: Tommi Tervonen.
- rparscore** Classification trees for ordinal responses. Authors: Giuliano Galimberti, Gabriele Soffritti, Matteo Di Maso.

- rplos** Interface to PLoS Journals API methods. Authors: Scott Chamberlain, Carl Boettiger.
- rportfolios** Random portfolio generation. Author: Frederick Novomestky.
- rreval** Remote R Evaluator (rreval). Author: Barnet Wagman. In view: *HighPerformanceComputing*.
- riskDistributions** Collection of functions for fitting distributions (related to the 'rrisk' project). Authors: Natalia Belgorodski, Matthias Greiner, Kristin Tolksdorf, Katharina Schueller.
- rsgcc** Gini methodology-based correlation and clustering analysis of both microarray and RNA-Seq gene expression data. Authors: Chuang Ma, Xiangfeng Wang.
- rstiefel** Random orthonormal matrix generation on the Stiefel manifold. Author: Peter Hoff.
- sROC** Nonparametric Smooth ROC Curves for Continuous Data. Author: Xiao-Feng Wang.
- scam** Shape constrained additive models. Author: Natalya Pya.
- scio** Sparse Column-wise Inverse Operator. Authors: Xi (Rossi) Luo and Weidong Liu.
- sdprisk** Measures of Risk for the Compound Poisson Risk Process with Diffusion. Authors: Benjamin Baumgartner, Riccardo Gatto.
- semGOF** Goodness-of-fit indices for structural equations models. Author: Elena Bertossi.
- semTools** Useful tools for structural equation modeling. Authors: Sunthud Pornprasertmanit, Patrick Miller, Alex Schoemann, Yves Rosseel.
- sentiment** Tools for Sentiment Analysis. Author: Timothy P. Jurka.
- sessionTools** Tools for saving, restoring and teleporting R sessions. Author: Matthew D. Furia.
- setWidth** Set the value of options("width") on terminal emulators. Author: Jakson Aquino.
- sft** Functions for Systems Factorial Technology Analysis of Data. Authors: Joe Hout, Leslie Blaha.
- shotGroups** Analyze shot group data. Author: Daniel Wollschlaeger.
- simSummary** Simulation summary. Author: Gregor Gorjanc.
- simsem** SIMulated Structural Equation Modeling. Authors: Sunthud Pornprasertmanit, Patrick Miller, Alexander Schoemann.
- sitools** Format a number to a string with SI prefix. Authors: Jonas Stein, Ben Tupper.
- skewtools** Tools for analyze Skew-Elliptical distributions and related models. Author: Javier E. Contreras-Reyes.
- smcure** Fit Semiparametric Mixture Cure Models. Authors: Chao Cai, Yubo Zou, Yingwei Peng, Jiajia Zhang.
- soilDB** Soil Database Interface. Authors: D.E. Beaudette and J.M. Skovlin.
- soobench** Single Objective Optimization Benchmark Functions. Author: Olaf Mersmann Bernd Bischl.
- spMC** Continuous Lag Spatial Markov Chains. Author: Luca Sartore.
- spaceExt** Extension of SPACE. Author: Shiyuan He.
- spacom** Spatially weighted context data for multi-level modelling. Authors: Till Junge, Sandra Penic, Guy Elcheroth.
- sparseLTSEigen** RcppEigen back end for sparse least trimmed squares regression. Author: Andreas Alfons.
- sparsenet** Fit sparse linear regression models via nonconvex optimization. Authors: Rahul Mazumder, Trevor Hastie, Jerome Friedman.
- spartan** Spartan (Simulation Parameter Analysis R Toolkit Application). Authors: Kieran Alden, Mark Read, Paul Andrews, Jon Timmis, Henrique Veiga-Fernandes, Mark Coles.
- spatialprobit** Spatial Probit Models. Authors: Stefan Wilhelm and Miguel Godinho de Matos.
- spcadjust** Functions for calibrating control charts. Authors: Axel Gandy and Jan Terje Kvaloy.
- spclust** Selective phenotyping for experimental crosses. Author: Emma Huang.
- spcov** Sparse Estimation of a Covariance Matrix. Authors: Jacob Bien and Rob Tibshirani.
- speedglm** Fitting Linear and Generalized Linear Models to large data sets. Author: Marco ENEA. In view: *HighPerformanceComputing*.
- splm** Econometric Models for Spatial Panel Data. Authors: Giovanni Millo, Gianfranco Piras.
- stabledist** Stable Distribution Functions. Authors: Diethelm Wuertz, Martin Maechler and Rmetrics core team members. In view: *Distributions*.
- stellaR** stellar evolution tracks and isochrones. Authors: Matteo Dell'Omodarme [aut, cre], Giada Valle [aut]. In view: *ChemPhys*.

- stpp** Space-Time Point Pattern simulation, visualisation and analysis. Authors: Edith Gabriel, Peter J Diggle, stan function by Barry Rowlingson.
- superdiag** R Code for Testing Markov Chain Non-convergence. Authors: Tsung-han Tsai, Jeff Gill and Jonathan Rapkin.
- survSNP** Power and Sample Size Calculations for SNP Association Studies with Censored Time to Event Outcomes. Authors: Kouros Owzar, Zhiguo Li, Nancy Cox, Sin-Ho Jung and Chan-hee Yi.
- svHttp** SciViews GUI API — R HTTP server. Author: Philippe Grosjean.
- svKomodo** SciViews GUI API — Functions to interface with Komodo Edit/IDE. Author: Philippe Grosjean.
- sybil** SyBiL — Efficient constrained based modelling in R. Authors: Gabriel Gelius-Dietrich [cre], C. Jonathan Fritzscheier [ctb], Rajen Piernikarczyk [ctb], Benjamin Braasch [ctb].
- sybilDynFBA** Dynamic FBA : dynamic flux balance analysis. Author: Abdelmoneim Amer Desouki.
- synbreed** Framework for the analysis of genomic prediction data using R. Authors: Valentin Wimmer, Theresa Albrecht, Hans-Juergen Auinger, Chris-Carolin Schoen with contributions by Larry Schaeffer, Malena Erbe, Ulrike Ober and Christian Reimer.
- synbreedData** Data for the **synbreed** package. Authors: Valentin Wimmer, Theresa Albrecht, Hans-Juergen Auinger, Chris-Carolin Schoen with contributions by Malena Erbe, Ulrike Ober and Christian Reimer.
- tawny.types** Common types for **tawny**. Author: Brian Lee Yung Rowe.
- tempdisagg** Methods for Temporal Disaggregation and Interpolation of Time Series. Authors: Christoph Sax, Peter Steiner. In view: *Time-Series*.
- ternvis** Visualisation, verification and calibration of ternary probabilistic forecasts. Author: Tim Jupp.
- tfplot** Time Frame User Utilities. Author: Paul Gilbert.
- timetools** provides objects and tools to manipulate irregular unhomogeneous time data and sub-time data. Author: Vladislav Navel.
- tlmcc** Linear Student-t Mixed-Effects Models with Censored Data. Authors: Larissa Matos, Marcos Prates and Victor Lachos. In view: *Survival*.
- tm.plugin.factiva** A plug-in for the **tm** text mining framework to import articles from Factiva. Author: Milan Bouchet-Valat.
- tpe** Tree preserving embedding. Authors: Albert D. Shieh, Tatsunori B. Hashimoto, and Edoardo M. Airolidi.
- translate** Bindings for the Google Translate API v2. Author: Peter Danenberg.
- truncdist** Truncated Random Variables. Authors: Frederick Novomestky, Saralees Nadarajah.
- turboEM** A Suite of Convergence Acceleration Schemes for EM and MM algorithms. Authors: Jennifer F. Bobb, Hui Zhao, and Ravi Varadhan.
- vacem** Vaccination Activities Coverage Estimation Model. Authors: Justin Lessler, Jessica Metcalf.
- varbvs** Variational inference for Bayesian variable selection. Author: Peter Carbonetto.
- vec2dtransf** 2D Cartesian Coordinate Transformation. Author: German Carrillo. In view: *Spatial*.
- vimcom** Intermediate the communication between Vim and R. Author: Jakson Aquino.
- visreg** Visualization of regression functions. Authors: Patrick Breheny, Woodrow Burchett.
- visualFields** Statistical methods for visual fields. Authors: Ivan Marin-Franch, Paul H Artes.
- waffect** A package to simulate constrained phenotypes under a disease model H1. Authors: Gregory Nuel and Vittorio Perduca.
- weightedKmeans** Weighted KMeans Clustering. Authors: Graham Williams, Joshua Z Huang, Xiaojun Chen, Qiang Wang, Longfei Xiao.
- wfe** Weighted Linear Fixed Effects Estimators for Causal Inference. Authors: In Song Kim, Kosuke Imai.

Other changes

- The following packages which were previously double-hosted on CRAN and Bioconductor were moved to the Archive, and are now solely available from Bioconductor: **aroma.light**, **DART**.
- The following packages which were previously double-hosted on CRAN and Omegahat were moved to the Archive, and are now solely available from Omegahat: **CGIwithR**, **Rstem**, **XMLSchema**.

- The following packages were moved to the Archive: **Aelasticnet**, **BARD**, **BioIDMapper**, **CoCo**, **CoCoCg**, **CoCoGraph**, **DiagnosisMed**, **FEST**, **FactoClass**, **FracSim**, **GGally**, **GLDEX**, **IniStatR**, **LLAhclust**, **MMG**, **OSACC**, **Package:**, **RFreak**, **RImageJ**, **ROracleUI**, **RPP-analyzer**, **SSOAP**, **Sim.DiffProcGUI**, **SoPhy**, **SocrataR**, **SubpathwayMiner**, **TSTutorial**, **TradeStrategyAnalyzer**, **agsemisc**, **binomSamSize**, **caMassClass**, **catmap**, **cheatmap**, **dmRTools**, **doSMP**, **dummies**, **egarch**, **farmR**, **formula.tools**, **fso**, **geoPlot**, **geofd**, **ggcolpairs**, **gllm**, **glpk**, **gmaps**, **gmvalid**, **gsc**, **hsmm**, **kernelPop**, **kinship**, **knnflex**, **lago**, **last.call**, **latentnet**, **lcd**, **log10**, **makesweave**, **mapReduce**, **marelacTeaching**, **mirf**, **mvgraph**, **mvtnorm-pcs**, **ncomplete**, **nnDiag**, **nonrandom**, **nover-**

lap, **nvis**, **ofp**, **operator.tools**, **optpart**, **pheno**, **pkDAClass**, **press**, **pressData**, **revoIPC**, **rimage**, **rmetasim**, **rpvm**, **segclust**, **siatclust**, **snpXpert**, **spef**, **statnet**, **stringkernels**, **svcR**, **time**, **treelet**, **ttime**, **twopartqtl**, **urn**, **yest**.

- The following packages were resurrected from the Archive: **DescribeDisplay**, **EMC**, **EMCC**, **G1DBN**, **SMC**, **diseasemapping**, **fArma**, **irt-Prob**, **partsm**, **powerpkg**, **riv**.

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

Achim Zeileis
Universität Innsbruck, Austria
Achim.Zeileis@R-project.org

R Foundation News

by Kurt Hornik

Donations and new members

Donations

Ilker Esener, Switzerland
Greenwich Statistics, France
Kansai University, Faculty Commerce, Japan
Olaf Krenz, Germany
Phil Shepherd, New Zealand
Joshua Wiley, USA

New benefactors

Greenwich Statistics, France
Quantide, Italy

New supporting institutions

Clinical Trial Unit, University Hospital Basel,
Switzerland

New supporting members

Benjamin French, USA
Arthur W. Green, USA
Scott Kostyshak, USA
Philippe Baril Lecavalier, Canada
Thomas Roth, Germany
Mathe W. Woodyard, USA

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