The R Journal

Volume 2/2, December 2010

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

Contents

The R 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 copyrighted by the respective authors.

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

Editor-in-Chief:
Peter Dalgaard
Center for Statistics
Copenhagen Business School
Solbjerg Plads 3
2000 Frederiksberg
Denmark

Editorial Board:
Vince Carey, Martyn Plummer, and Heather Turner.

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 Peter Dalgaard

Welcome to the 2nd issue of the 2nd volume of The R Journal.

I am pleased to say that we can offer ten peer-reviewed papers this time. Many thanks go to the authors and the reviewers who ensure that our articles live up to high academic standards. The transition from R News to The R Journal is now nearly completed. We are now listed by EBSCO and the registration procedure with Thomson Reuters is well on the way. We thereby move into the framework of scientific journals and away from the grey-literature newsletter format; however, it should be stressed that R News was a fairly high-impact piece of grey literature: A cited reference search turned up around 1300 references to the just over 200 papers that were published in R News!

I am particularly happy to see the paper by Soetart et al. on differential equation solvers. In many fields of research, the natural formulation of models is via local relations at the infinitesimal level, rather than via closed form mathematical expressions, and quite often solutions rely on simplifying assumptions. My own PhD work, some 25 years ago, concerned diffusion of substances within the human eye, with the ultimate goal of measuring the state of the blood-retinal barrier. Solutions for this problem could be obtained for short timespans, if one assumed that the eye was completely spherical. Extending the solutions to accommodate more realistic models (a necessity for fitting actual experimental data) resulted in quite unwieldy formulas, and even then, did not give you the kind of modelling freedom that you really wanted to elucidate the scientific issue.

In contrast, numerical procedures could fairly easily be set up and modified to better fit reality. The main problem was that they tended to be computationally demanding. Especially for transient solutions in two or three spatial dimensions, computers simply were not fast enough in a time where numerical performance was measured in fractions of a MFLOPS (million floating point operations per second). Today, the relevant measure is GFLOPS and we should be getting much closer to practicable solutions.

However, raw computing power is not sufficient; there are non-obvious aspects of numerical analysis that should not be taken lightly, notably issues of stability and accuracy. There is a reason that numerical analysis is a scientific field in its own right.

From a statistician's perspective, being able to fit models to actual data is of prime importance. For models with only a few parameters, you can get quite far with nonlinear regression and a good numerical solver. For ill-posed problems with functional parameters (the so-called "inverse problems"), and for stochastic differential equations, there still appears to be work to be done. Soetart et al. do not go into these issues, but I hope that their paper will be an inspiration for further work.

With this issue, in accordance with the rotation rules of the Editorial Board, I step down as Editor-in-Chief, to be succeded by Heather Turner. Heather has already played a key role in the transition from R News to The R Journal, as well as being probably the most efficient Associate Editor on the Board. The Editorial Board will be losing last year's Editor-in-Chief, Vince Carey, who has now been on board for the full four years. We shall miss Vince, who has always been good for a precise and principled argument and in the process taught at least me several new words. We also welcome Hadley Wickham as a new Associate Editor and member of the Editorial Board.

Season's greetings and best wishes for a happy 2011!

Solving Differential Equations in R

by Karline Soetaert, Thomas Petzoldt and R. Woodrow Setzer[1]

Abstract Although R is still predominantly applied for statistical analysis and graphical representation, it is rapidly becoming more suitable for mathematical computing. One of the fields where considerable progress has been made recently is the solution of differential equations. Here we give a brief overview of differential equations that can now be solved by R.

Introduction

Differential equations describe exchanges of matter, energy, information or any other quantities, often as they vary in time and/or space. Their thorough analytical treatment forms the basis of fundamental theories in mathematics and physics, and they are increasingly applied in chemistry, life sciences and economics.

Differential equations are solved by integration, but unfortunately, for many practical applications in science and engineering, systems of differential equations cannot be integrated to give an analytical solution, but rather need to be solved numerically.

Many advanced numerical algorithms that solve differential equations are available as (open-source) computer codes, written in programming languages like FORTRAN or C and that are available from repositories like GAMS (`http://gams.nist.gov/`) or NETLIB (`www.netlib.org`).

Depending on the problem, mathematical formalisations may consist of ordinary differential equations (ODE), partial differential equations (PDE), differential algebraic equations (DAE), or delay differential equations (DDE). In addition, a distinction is made between initial value problems (IVP) and boundary value problems (BVP).

With the introduction of R-package `odesolve` (Setzer, 2001), it became possible to use R (R Development Core Team, 2009) for solving very simple initial value problems of systems of ordinary differential equations, using the `lsoda` algorithm of Hindmarsh (1983) and Petzold (1983). However, many real-life applications, including physical transport modeling, equilibrium chemistry or the modeling of electrical circuits, could not be solved with this package.

Since `odesolve`, much effort has been made to improve R's capabilities to handle differential equations, mostly by incorporating published and well tested numerical codes, such that now a much more complete repertoire of differential equations can be numerically solved.

More specifically, the following types of differential equations can now be handled with add-on packages in R:

- Initial value problems (IVP) of ordinary differential equations (ODE), using package **deSolve** (Soetaert et al., 2010b).
- Initial value differential algebraic equations (DAE), package **deSolve** .
- Initial value partial differential equations (PDE), packages **deSolve** and **ReacTran** (Soetaert and Meysman, 2010).
- Boundary value problems (BVP) of ordinary differential equations, using package **bvpSolve** (Soetaert et al., 2010a), or **ReacTran** and **rootSolve** (Soetaert, 2009).
- Initial value delay differential equations (DDE), using packages **deSolve** or **PBSddesolve** (Couture-Beil et al., 2010).
- Stochastic differential equations (SDE), using packages **sde** (Iacus, 2008) and **pomp** (King et al., 2008).

In this short overview, we demonstrate how to solve the first four types of differential equations in R. It is beyond the scope to give an exhaustive overview about the vast number of methods to solve these differential equations and their theory, so the reader is encouraged to consult one of the numerous textbooks (e.g., Ascher and Petzold, 1998; Press et al., 2007; Hairer et al., 2009; Hairer and Wanner, 2010; LeVeque, 2007, and many others).

In addition, a large number of analytical and numerical methods exists for the analysis of bifurcations and stability properties of deterministic systems, the efficient simulation of stochastic differential equations or the estimation of parameters. We do not deal with these methods here.

Types of differential equations

Ordinary differential equations

Ordinary differential equations describe the change of a *state variable* y as a function f of one *independent variable* t (e.g., time or space), of y itself, and, optionally, a set of other variables p, often called *parameters*:

$$y' = \frac{dy}{dt} = f(t, y, p)$$

[1]The views expressed in this paper are those of the authors and do not necessarily reflect the views or policies of the U.S. Environmental Protection Agency

In many cases, solving differential equations requires the introduction of extra conditions. In the following, we concentrate on the numerical treatment of two classes of problems, namely initial value problems and boundary value problems.

Initial value problems

If the extra conditions are specified at the initial value of the independent variable, the differential equations are called **initial value problems** (IVP).

There exist two main classes of algorithms to numerically solve such problems, so-called *Runge-Kutta* formulas and *linear multistep* formulas (Hairer et al., 2009; Hairer and Wanner, 2010). The latter contains two important families, the Adams family and the backward differentiation formulae (BDF).

Another important distinction is between *explicit* and *implicit* methods, where the latter methods can solve a particular class of equations (so-called "stiff" equations) where explicit methods have problems with stability and efficiency. Stiffness occurs for instance if a problem has components with different rates of variation according to the independent variable. Very often there will be a tradeoff between using explicit methods that require little work per integration step and implicit methods which are able to take larger integration steps, but need (much) more work for one step.

In R, initial value problems can be solved with functions from package **deSolve** (Soetaert et al., 2010b), which implements many solvers from ODEPACK (Hindmarsh, 1983), the code `vode` (Brown et al., 1989), the differential algebraic equation solver `daspk` (Brenan et al., 1996), all belonging to the linear multistep methods, and comprising Adams methods as well as backward differentiation formulae. The former methods are explicit, the latter implicit. In addition, this package contains a de-novo implementation of a rather general Runge-Kutta solver based on Dormand and Prince (1980); Prince and Dormand (1981); Bogacki and Shampine (1989); Cash and Karp (1990) and using ideas from Butcher (1987) and Press et al. (2007). Finally, the implicit Runge-Kutta method `radau` (Hairer et al., 2009) has been added recently.

Boundary value problems

If the extra conditions are specified at different values of the independent variable, the differential equations are called **boundary value problems** (BVP). A standard textbook on this subject is Ascher et al. (1995).

Package **bvpSolve** (Soetaert et al., 2010a) implements three methods to solve boundary value problems. The simplest solution method is the *single shooting method*, which combines initial value problem integration with a nonlinear root finding algorithm (Press et al., 2007). Two more stable solution methods implement a mono implicit Runge-Kutta (MIRK) code, based on the FORTRAN code `twpbvpC` (Cash and Mazzia, 2005), and the collocation method, based on the FORTRAN code `colnew` (Bader and Ascher, 1987). Some boundary value problems can also be solved with functions from packages **ReacTran** and **rootSolve** (see below).

Partial differential equations

In contrast to ODEs where there is only one independent variable, partial differential equations (PDE) contain partial derivatives with respect to more than one independent variable, for instance t (time) and x (a spatial dimension). To distinguish this type of equations from ODEs, the derivatives are represented with the ∂ symbol, e.g.

$$\frac{\partial y}{\partial t} = f(t, x, y, \frac{\partial y}{\partial x}, p)$$

Partial differential equations can be solved by subdividing one or more of the continuous independent variables in a number of grid cells, and replacing the derivatives by discrete, algebraic approximate equations, so-called finite differences (cf. LeVeque, 2007; Hundsdorfer and Verwer, 2003).

For time-varying cases, it is customary to discretise the spatial coordinate(s) only, while time is left in continuous form. This is called the method-of-lines, and in this way, one PDE is translated into a large number of coupled ordinary differential equations, that can be solved with the usual initial value problem solvers (cf. Hamdi et al., 2007). This applies to parabolic PDEs such as the heat equation, and to hyperbolic PDEs such as the wave equation.

For time-invariant problems, usually all independent variables are discretised, and the derivatives approximated by algebraic equations, which are solved by root-finding techniques. This technique applies to elliptic PDEs.

R-package **ReacTran** provides functions to generate finite differences on a structured grid. After that, the resulting time-varying cases can be solved with specially-designed functions from package **deSolve**, while time-invariant cases can be solved with root-solving methods from package **rootSolve** .

Differential algebraic equations

Differential-algebraic equations (DAE) contain a mixture of differential (f) and algebraic equations (g), the latter e.g. for maintaining mass-balance conditions:

$$y' = f(t, y, p)$$
$$0 = g(t, y, p)$$

Important for the solution of a DAE is its index. The index of a DAE is the number of differentiations

needed until a system consisting only of ODEs is obtained.

Function `daspk` (Brenan et al., 1996) from package **deSolve** solves (relatively simple) DAEs of index at most 1, while function `radau` (Hairer et al., 2009) solves DAEs of index up to 3.

Implementation details

The implemented solver functions are explained by means of the `ode`-function, used for the solution of initial value problems. The interfaces to the other solvers have an analogous definition:

```
ode(y, times, func, parms, method = c("lsoda",
    "lsode", "lsodes", "lsodar",
    "vode", "daspk", "euler", "rk4",
    "ode23", "ode45", "radau", "bdf",
    "bdf_d", "adams", "impAdams",
    "impAdams_d"), ...)
```

To use this, the system of differential equations can be defined as an R-function (`func`) that computes derivatives in the ODE system (the model definition) according to the independent variable (e.g. time `t`). `func` can also be a function in a dynamically loaded shared library (Soetaert et al., 2010c) and, in addition, some solvers support also the supply of an analytically derived function of partial derivatives (Jacobian matrix).

If `func` is an R-function, it must be defined as:

```
func <- function(t, y, parms, ...)
```

where `t` is the actual value of the independent variable (e.g. the current time point in the integration), `y` is the current estimate of the variables in the ODE system, `parms` is the parameter vector and `...` can be used to pass additional arguments to the function.

The return value of `func` should be a list, whose first element is a vector containing the derivatives of `y` with respect to `t`, and whose next elements are optional global values that can be recorded at each point in `times`. The derivatives must be specified in the same order as the state variables `y`.

Depending on the algorithm specified in argument `method`, numerical simulation proceeds either exactly at the time steps specified in `times`, or using time steps that are independent from `times` and where the output is generated by interpolation. With the exception of method `euler` and several fixed-step Runge-Kutta methods all algorithms have automatic time stepping, which can be controlled by setting accuracy requirements (see below) or by using optional arguments like `hini` (initial time step), `hmin` (minimal time step) and `hmax` (maximum time step). Specific details, e.g. about the applied interpolation methods can be found in the manual pages and the original literature cited there.

Numerical accuracy

Numerical solution of a system of differential equations is an approximation and therefore prone to numerical errors, originating from several sources:

1. time step and accuracy order of the solver,
2. floating point arithmetics,
3. properties of the differential system and stability of the solution algorithm.

For methods with automatic stepsize selection, accuracy of the computation can be adjusted using the non-negative arguments `atol` (absolute tolerance) and `rtol` (relative tolerance), which control the local errors of the integration.

Like R itself, all solvers use double-precision floating-point arithmetics according to IEEE Standard 754 (2008), which means that it can represent numbers between approx. $\pm 2.25\ 10^{-308}$ to approx. $\pm 1.8\ 10^{308}$ and with 16 significant digits. It is therefore not advisable to set **rtol** below 10^{-16}, except setting it to zero with the intention to use absolute tolerance exclusively.

The solvers provided by the packages presented below have proven to be quite robust in most practical cases, however users should always be aware about the problems and limitations of numerical methods and carefully check results for plausibility. The section "Troubleshooting" in the package vignette (Soetaert et al., 2010d) should be consulted as a first source for solving typical problems.

Examples

An initial value ODE

Consider the famous van der Pol equation (van der Pol and van der Mark, 1927), that describes a non-conservative oscillator with non-linear damping and which was originally developed for electrical circuits employing vacuum tubes. The oscillation is described by means of a 2^{nd} order ODE:

$$z'' - \mu(1 - z^2)z' + z = 0$$

Such a system can be routinely rewritten as a system of two 1^{st} order ODEs, if we substitute z'' with y_1' and z' with y_2:

$$y_1' = y_2$$
$$y_2' = \mu \cdot (1 - {y_1}^2) \cdot y_2 - y_1$$

There is one parameter, μ, and two differential variables, y_1 and y_2 with initial values (at $t = 0$):

$$y_{1_{(t=0)}} = 2$$
$$y_{2_{(t=0)}} = 0$$

The van der Pol equation is often used as a test problem for ODE solvers, as, for large μ, its dynamics consists of parts where the solution changes very slowly, alternating with regions of very sharp changes. This "stiffness" makes the equation quite challenging to solve.

In R, this model is implemented as a function (`vdpol`) whose inputs are the current time (`t`), the values of the state variables (`y`), and the parameters (`mu`); the function returns a `list` with as first element the derivatives, concatenated.

```
vdpol <- function (t, y, mu) {
  list(c(
    y[2],
    mu * (1 - y[1]^2) * y[2] - y[1]
  ))
}
```

After defining the initial condition of the state variables (`yini`), the model is solved, and output written at selected time points (`times`), using **deSolve**'s integration function `ode`. The default routine `lsoda`, which is invoked by `ode` automatically switches between stiff and non-stiff methods, depending on the problem (Petzold, 1983).

We run the model for a typically stiff (`mu = 1000`) and nonstiff (`mu = 1`) situation:

```
library(deSolve)
yini <- c(y1 = 2, y2 = 0)
stiff <- ode(y = yini, func = vdpol,
    times = 0:3000, parms = 1000)

nonstiff <- ode(y = yini, func = vdpol,
    times = seq(0, 30, by = 0.01),
    parms = 1)
```

The model returns a matrix, of class `deSolve`, with in its first column the time values, followed by the values of the state variables:

```
head(stiff, n = 3)
```

```
     time       y1            y2
[1,]    0 2.000000  0.0000000000
[2,]    1 1.999333 -0.0006670373
[3,]    2 1.998666 -0.0006674088
```

Figures are generated using the S3 plot method for objects of class `deSolve`:

```
plot(stiff, type = "l", which = "y1",
  lwd = 2, ylab = "y",
  main = "IVP ODE, stiff")

plot(nonstiff, type = "l", which = "y1",
  lwd = 2, ylab = "y",
  main = "IVP ODE, nonstiff")
```

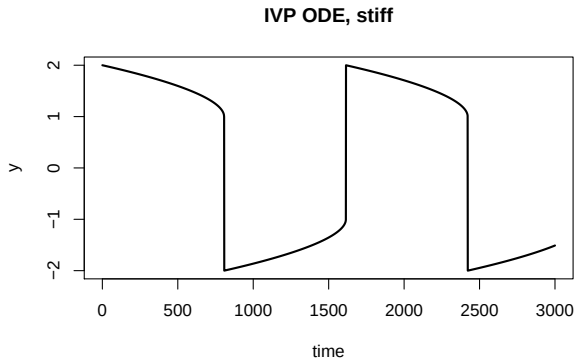


Figure 1: Solution of the van der Pol equation, an initial value ordinary differential equation, stiff case, $\mu = 1000$.

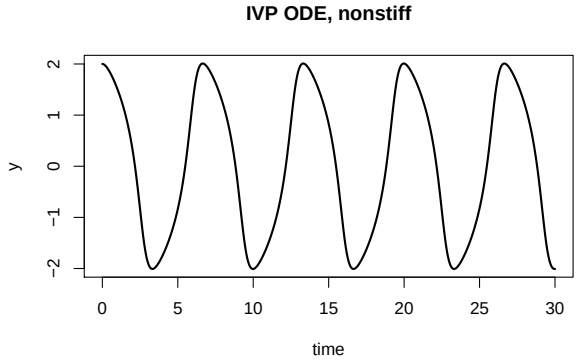


Figure 2: Solution of the van der Pol equation, an initial value ordinary differential equation, non-stiff case, $\mu = 1$.

solver	non-stiff	stiff
ode23	0.37	271.19
lsoda	0.26	0.23
adams	0.13	616.13
bdf	0.15	0.22
radau	0.53	0.72

Table 1: Comparison of solvers for a stiff and a non-stiff parametrisation of the van der Pol equation (time in seconds, mean values of ten simulations on an AMD AM2 X2 3000 CPU).

A comparison of timings for two explicit solvers, the Runge-Kutta method (`ode23`) and the `adams` method, with the implicit multistep solver (`bdf`, backward differentiation formula) shows a clear advantage for the latter in the stiff case (Figure 1). The default solver (`lsoda`) is not necessarily the fastest, but shows robust behavior due to automatic stiffness detection. It uses the explicit multistep Adams method for the non-stiff case and the BDF method for the stiff case. The accuracy is comparable for all

solvers with `atol` = `rtol` = 10^{-6}, the default.

A boundary value ODE

The webpage of Jeff Cash (Cash, 2009) contains many test cases, including their analytical solution (see below), that BVP solvers should be able to solve. We use equation no. 14 from this webpage as an example:

$$\xi y'' - y = -(\xi\pi^2 + 1)\cos(\pi x)$$

on the interval $[-1,1]$, and subject to the boundary conditions:

$$y_{(x=-1)} = 0$$
$$y_{(x=+1)} = 0$$

The second-order equation first is rewritten as two first-order equations:

$$y_1' = y_2$$
$$y_2' = 1/\xi \cdot (y_1 - (\xi\pi^2 + 1)\cos(\pi x))$$

It is implemented in R as:

```
Prob14 <- function(x, y, xi)  {
  list(c(
    y[2],
    1/xi * (y[1] - (xi*pi*pi+1) * cos(pi*x))
  ))
}
```

With decreasing values of ξ, this problem becomes increasingly difficult to solve. We use three values of ξ, and solve the problem with the shooting, the MIRK and the collocation method (Ascher et al., 1995).

Note how the initial conditions `yini` and the conditions at the end of the integration interval `yend` are specified, where `NA` denotes that the value is not known. The independent variable is called `x` here (rather than `times` in `ode`).

```
library(bvpSolve)
x <- seq(-1, 1, by = 0.01)
shoot <- bvpshoot(yini = c(0, NA),
    yend = c(0, NA), x = x, parms = 0.01,
    func = Prob14)

twp <- bvptwp(yini = c(0, NA), yend = c(0,
    NA), x = x, parms = 0.0025,
    func = Prob14)

coll <- bvpcol(yini = c(0, NA),
    yend = c(0, NA), x = x, parms = 1e-04,
    func = Prob14)
```

The numerical approximation generated by `bvptwp` is very close to the analytical solution, e.g. for $\xi = 0.0025$:

```
xi <- 0.0025
analytic <- cos(pi * x) + exp((x -
    1)/sqrt(xi)) + exp(-(x + 1)/sqrt(xi))
max(abs(analytic - twp[, 2]))
```

```
[1] 7.788209e-10
```

A similar low discrepancy ($4 \cdot 10^{-11}$) is noted for the $\xi = 0.0001$ as solved by `bvpcol`; the shooting method is considerably less precise ($1.4 \cdot 10^{-5}$), although the same tolerance (`atol` = 10^{-8}) was used for all runs.

The plot shows how the shape of the solution is affected by the parameter ξ, becoming more and more steep near the boundaries, and therefore more and more difficult to solve, as ξ gets smaller.

```
plot(shoot[, 1], shoot[, 2], type = "l", lwd = 2,
  ylim = c(-1, 1), col = "blue",
  xlab = "x", ylab = "y", main = "BVP ODE")
lines(twp[, 1], twp[, 2], col = "red", lwd = 2)
lines(coll[, 1], coll[, 2], col = "green", lwd = 2)
legend("topright", legend = c("0.01", "0.0025",
  "0.0001"), col = c("blue", "red", "green"),
  title = expression(xi), lwd = 2)
```

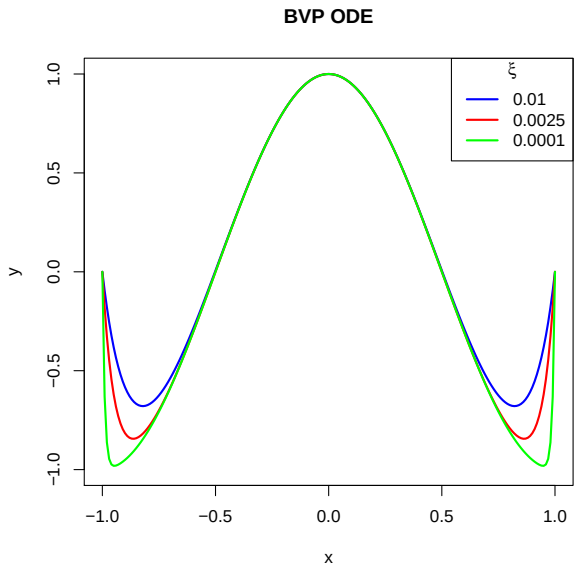


Figure 3: Solution of the BVP ODE problem, for different values of parameter ξ.

Differential algebraic equations

The so called "Rober problem" describes an autocatalytic reaction (Robertson, 1966) between three chemical species, y_1, y_2 and y_3. The problem can be formulated either as an ODE (Mazzia and Magherini, 2008), or as a DAE:

$$y_1' = -0.04y_1 + 10^4 y_2 y_3$$
$$y_2' = 0.04y_1 - 10^4 y_2 y_3 - 310^7 y_2^2$$
$$1 = y_1 + y_2 + y_3$$

where the first two equations are differential equations that specify the dynamics of chemical species y_1 and y_2, while the third algebraic equation ensures that the summed concentration of the three species remains 1.

The DAE has to be specified by the *residual function* instead of the rates of change (as in ODEs).

$$r_1 = -y_1' - 0.04y_1 + 10^4 y_2 y_3$$
$$r_2 = -y_2' + 0.04y_1 - 10^4 y_2 y_3 - 3\,10^7 y_2^2$$
$$r_3 = -1 + y_1 + y_2 + y_3$$

Implemented in R this becomes:

```
daefun<-function(t, y, dy, parms) {
    res1 <- - dy[1] - 0.04 * y[1] +
              1e4 * y[2] * y[3]
    res2 <- - dy[2] + 0.04 * y[1] -
              1e4 * y[2] * y[3] - 3e7 * y[2]^2
    res3 <- y[1] + y[2] + y[3] - 1
    list(c(res1, res2, res3),
         error = as.vector(y[1] + y[2] + y[3]) - 1)
}

yini  <- c(y1 = 1, y2 = 0, y3 = 0)
dyini <- c(-0.04, 0.04, 0)
times <- 10 ^ seq(-6,6,0.1)
```

The input arguments of function `daefun` are the current time (`t`), the values of the state variables and their derivatives (`y`, `dy`) and the parameters (`parms`). It returns the residuals, concatenated and an output variable, the error in the algebraic equation. The latter is added to check upon the accuracy of the results.

For DAEs solved with `daspk`, both the state variables and their derivatives need to be initialised (`y` and `dy`). Here we make sure that the initial conditions for `y` obey the algebraic constraint, while also the initial condition of the derivatives is consistent with the dynamics.

```
library(deSolve)
print(system.time(out <-daspk(y = yini,
  dy = dyini, times = times, res = daefun,
  parms = NULL)))

  user  system elapsed
  0.07    0.00    0.11
```

An S3 `plot` method can be used to plot all variables at once:

```
plot(out, ylab = "conc.", xlab = "time",
  type = "l", lwd = 2, log = "x")
mtext("IVP DAE", side = 3, outer = TRUE,
  line = -1)
```

There is a very fast initial change in concentrations, mainly due to the quick reaction between y_1 and y_2 and amongst y_2. After that, the slow reaction of y_1 with y_2 causes the system to change much more smoothly. This is typical for stiff problems.

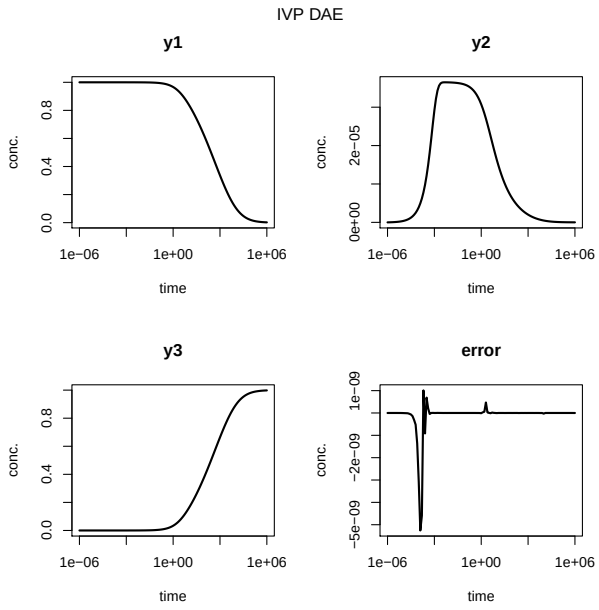


Figure 4: Solution of the DAE problem for the substances y_1, y_2, y_3; mass balance error: deviation of total sum from one.

Partial differential equations

In **partial differential equations** (PDE), the function has several independent variables (e.g. time and depth) and contains their partial derivatives.

Many partial differential equations can be solved by numerical approximation (finite differencing) after rewriting them as a set of ODEs (see Schiesser, 1991; LeVeque, 2007; Hundsdorfer and Verwer, 2003).

Functions `tran.1D`, `tran.2D`, and `tran.3D` from R package **ReacTran** (Soetaert and Meysman, 2010) implement finite difference approximations of the diffusive-advective transport equation which, for the 1-D case, is:

$$-\frac{1}{A_x} \cdot \left[\frac{\partial}{\partial x} A_x \left(-D \cdot \frac{\partial C}{\partial x} \right) - \frac{\partial}{\partial x} (A_x \cdot u \cdot C) \right]$$

Here D is the "diffusion coefficient", u is the "advection rate", and A_x is some property (e.g. surface area) that depends on the independent variable, x.

It should be noted that the accuracy of the finite difference approximations can not be specified in the **ReacTran** functions. It is up to the user to make sure that the solutions are sufficiently accurate, e.g. by including more grid points.

One dimensional PDE

Diffusion-reaction models are a fundamental class of models which describe how concentration of matter, energy, information, etc. evolves in space and time under the influence of diffusive transport and transformation (Soetaert and Herman, 2009).

As an example, consider the 1-D diffusion-reaction model in $[0,10]$:

$$\frac{\partial C}{\partial t} = \frac{\partial}{\partial x}\left(D \cdot \frac{\partial C}{\partial x}\right) - Q$$

with C the concentration, `t` the time, `x` the distance from the origin, `Q`, the consumption rate, and with boundary conditions (values at the model edges):

$$\frac{\partial C}{\partial x}_{x=0} = 0$$
$$C_{x=10} = C_{ext}$$

To solve this model in R, first the 1-D model `Grid` is defined; it divides 10 cm (`L`) into 1000 boxes (`N`).

```
library(ReacTran)
Grid <- setup.grid.1D(N = 1000, L = 10)
```

The model equation includes a transport term, approximated by **ReacTran** function `tran.1D` and a consumption term (`Q`). The downstream boundary condition, prescribed as a concentration (`C.down`) needs to be specified, the zero-gradient at the upstream boundary is the default:

```
pde1D <-function(t, C, parms)  {
  tran <- tran.1D(C = C, D = D,
                  C.down = Cext, dx = Grid)$dC
  list(tran - Q)  # return value: rate of change
}
```

The model parameters are:

```
D    <- 1    # diffusion constant
Q    <- 1    # uptake rate
Cext <- 20
```

In a first application, the model is solved to *steady-state*, which retrieves the condition where the concentrations are invariant:

$$0 = \frac{\partial}{\partial x}\left(D \cdot \frac{\partial C}{\partial x}\right) - Q$$

In R, steady-state conditions can be estimated using functions from package **rootSolve** which implement amongst others a Newton-Raphson algorithm (Press et al., 2007). For 1-dimensional models, `steady.1D` is most efficient. The initial "guess" of the steady-state solution (`y`) is unimportant; here we take simply `N` random numbers. Argument `nspec = 1` informs the solver that only one component is described.

Although a system of 1000 equations needs to be solved, this takes only a fraction of a second:

```
library(rootSolve)
print(system.time(
std   <- steady.1D(y = runif(Grid$N),
  func = pde1D, parms = NULL, nspec = 1)
))
```

```
   user  system elapsed
   0.02    0.00    0.02
```

The values of the state-variables (`y`) are plotted against the distance, in the middle of the grid cells (`Grid$x.mid`).

```
plot (Grid$x.mid, std$y, type = "l",
  lwd = 2, main = "steady-state PDE",
  xlab = "x", ylab = "C", col = "red")
```

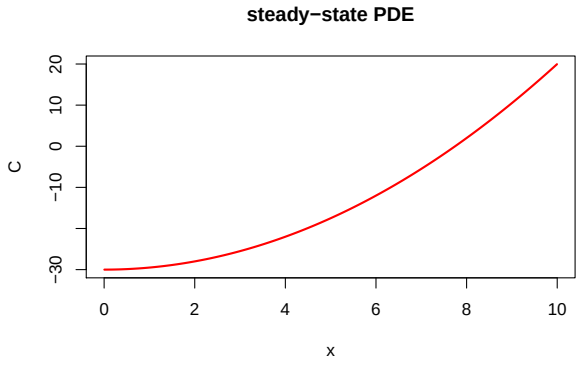


Figure 5: Steady-state solution of the 1-D diffusion-reaction model.

The analytical solution compares well with the numerical approximation:

```
analytical <- Q/2/D*(Grid$x.mid^2 - 10^2) + Cext
max(abs(analytical - std$y))
```

```
[1] 1.250003e-05
```

Next the model is run dynamically for 100 time units using **deSolve** function `ode.1D`, and starting with a uniform concentration:

```
require(deSolve)
times <- seq(0, 100, by = 1)
system.time(
  out <- ode.1D(y = rep(1, Grid$N),
    times = times, func = pde1D,
    parms = NULL, nspec = 1)
)
```

```
   user  system elapsed
   0.61    0.02    0.63
```

Here, `out` is a matrix, whose 1^{st} column contains the output times, and the next columns the values of the state variables in the different boxes; we print the first columns of the last three rows of this matrix:

```
tail(out[, 1:4], n = 3)
```

```
      time         1         2         3
[99,]   98 -27.55783 -27.55773 -27.55754
[100,]  99 -27.61735 -27.61725 -27.61706
[101,] 100 -27.67542 -27.67532 -27.67513
```

We plot the result using a blue-yellow-red color scheme, and using `deSolve`'s S3 method `image`. Figure 6 shows that, as time proceeds, gradients develop from the uniform distribution, until the system almost reaches steady-state at the end of the simulation.

```
image(out, xlab = "time, days",
      ylab = "Distance, cm",
      main = "PDE", add.contour = TRUE)
```

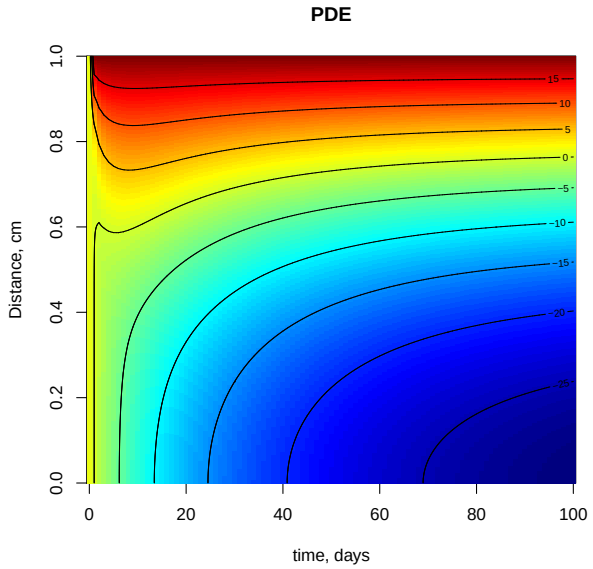


Figure 6: Dynamic solution of the 1-D diffusion-reaction model.

It should be noted that the steady-state model is effectively a boundary value problem, while the transient model is a prototype of a "parabolic" partial differential equation (LeVeque, 2007).

Whereas R can also solve the other two main classes of PDEs, i.e. of the "hyperbolic" and "elliptic" type, it is well beyond the scope of this paper to elaborate on that.

Discussion

Although R is still predominantly applied for statistical analysis and graphical representation, it is more and more suitable for mathematical computing, e.g. in the field of matrix algebra (Bates and Maechler, 2008). Thanks to the differential equation solvers, R is also emerging as a powerful environment for dynamic simulations (Petzoldt, 2003; Soetaert and Herman, 2009; Stevens, 2009).

The new package **deSolve** has retained all the funtionalities of its predecessor **odesolve** (Setzer, 2001), such as the potential to define models both in R code, or in compiled languages. However, compared to **odesolve**, it includes a more complete set of integrators, and a more extensive set of options to tune the integration routines, it provides more complete output, and has extended the applicability domain to include also DDEs, DAEs and PDEs.

Thanks to the DAE solvers `daspk` (Brenan et al., 1996) and `radau` (Hairer and Wanner, 2010) it is now also possible to model electronic circuits or equilibrium chemical systems. These problems are often of index ≤ 1. In many mechanical systems, physical constraints lead to DAEs of index up to 3, and these more complex problems can be solved with `radau`.

The inclusion of BVP and PDE solvers have opened up the application area to the field of reactive transport modelling (Soetaert and Meysman, 2010), such that R can now be used to describe quantities that change not only in time, but also along one or more spatial axes. We use it to model how ecosystems change along rivers, or in sediments, but it could equally serve to model the growth of a tumor in human brains, or the dispersion of toxicants in human tissues.

The open source matrix language R has great potential for dynamic modelling, and the tools currently available are suitable for solving a wide variety of practical and scientific problems. The performance is sufficient even for larger systems, especially when models can be formulated using matrix algebra or are implemented in compiled languages like C or Fortran (Soetaert et al., 2010b). Indeed, there is emerging interest in performing statistical analysis on differential equations, e.g. in package **nlmeODE** (Tornøe et al., 2004) for fitting non-linear mixed-effects models using differential equations, package **FME** (Soetaert and Petzoldt, 2010) for sensitivity analysis, parameter estimation and Markov chain Monte-Carlo analysis or package **ccems** for combinatorially complex equilibrium model selection (Radivoyevitch, 2008).

However, there is ample room for extensions and improvements. For instance, the PDE solvers are quite memory intensive, and could benefit from the implementation of sparse matrix solvers that are more efficient in this respect[2]. In addition, the methods implemented in **ReacTran** handle equations defined on very simple shapes only. Extending the PDE approach to finite elements (Strang and Fix, 1973) would open up the application domain of R to any irregular geometry. Other spatial discretisation schemes could be added, e.g. for use in fluid dynamics.

Our models are often applied to derive unknown parameters by fitting them against data; this relies on the availability of apt parameter fitting algorithms.

Discussion of these items is highly welcomed, in the new special interest group about dynamic mod-

[2] for instance, the "preconditioned Krylov" part of the `daspk` method is not yet supported

[3] `https://stat.ethz.ch/mailman/listinfo/r-sig-dynamic-models`

els[3] in R.

Bibliography

U. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Philadelphia, PA, 1995.

U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, 1998.

G. Bader and U. Ascher. A new basis implementation for a mixed order boundary value ODE solver. *SIAM J. Scient. Stat. Comput.*, 8:483–500, 1987.

D. Bates and M. Maechler. **Matrix***: A Matrix Package for R*, 2008. R package version 0.999375-9.

P. Bogacki and L. Shampine. A 3(2) pair of Runge-Kutta formulas. *Appl. Math. Lett.*, 2:1–9, 1989.

K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM Classics in Applied Mathematics, 1996.

P. N. Brown, G. D. Byrne, and A. C. Hindmarsh. **VODE**, a variable-coefficient ode solver. *SIAM J. Sci. Stat. Comput.*, 10:1038–1051, 1989.

J. C. Butcher. *The Numerical Analysis of Ordinary Differential Equations, Runge-Kutta and General Linear Methods*. Wiley, Chichester, New York, 1987.

J. R. Cash. *35 Test Problems for Two Way Point Boundary Value Problems*, 2009. URL http://www.ma.ic.ac.uk/~jcash/BVP_software/PROBLEMS.PDF.

J. R. Cash and A. H. Karp. A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides. *ACM Transactions on Mathematical Software*, 16:201–222, 1990.

J. R. Cash and F. Mazzia. A new mesh selection algorithm, based on conditioning, for two-point boundary value codes. *J. Comput. Appl. Math.*, 184: 362–381, 2005.

A. Couture-Beil, J. T. Schnute, and R. Haigh. **PBSddesolve***: Solver for Delay Differential Equations*, 2010. R package version 1.08.11.

J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 6:19–26, 1980.

E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. Second Revised Edition*. Springer-Verlag, Heidelberg, 2010.

E. Hairer, S. P. Noørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems. Second Revised Edition*. Springer-Verlag, Heidelberg, 2009.

S. Hamdi, W. E. Schiesser, and G. W. Griffiths. Method of lines. *Scholarpedia*, 2(7):2859, 2007.

A. C. Hindmarsh. **ODEPACK**, a systematized collection of ODE solvers. In R. Stepleman, editor, *Scientific Computing, Vol. 1 of IMACS Transactions on Scientific Computation*, pages 55–64. IMACS / North-Holland, Amsterdam, 1983.

W. Hundsdorfer and J. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations. Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2003.

S. M. Iacus. **sde***: Simulation and Inference for Stochastic Differential Equations*, 2008. R package version 2.0.3.

IEEE Standard 754. Ieee standard for floating-point arithmetic, Aug 2008.

A. A. King, E. L. Ionides, and C. M. Breto. **pomp***: Statistical Inference for Partially Observed Markov Processes*, 2008. R package version 0.21-3.

R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations, Steady State and Time Dependent Problems*. SIAM, 2007.

F. Mazzia and C. Magherini. *Test Set for Initial Value Problem Solvers, release 2.4*. Department of Mathematics, University of Bari, Italy, 2008. URL http://pitagora.dm.uniba.it/~testset. Report 4/2008.

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

T. Petzoldt. R as a simulation platform in ecological modelling. *R News*, 3(3):8–16, 2003.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

P. J. Prince and J. R. Dormand. High order embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 7:67–75, 1981.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL http://www.R-project.org. ISBN 3-900051-07-0.

T. Radivoyevitch. Equilibrium model selection: dTTP induced R1 dimerization. *BMC Systems Biology*, 2:15, 2008.

H. H. Robertson. The solution of a set of reaction rate equations. In J. Walsh, editor, *Numerical Analysis: An Introduction*, pages 178–182. Academic Press, London, 1966.

W. E. Schiesser. *The Numerical Method of Lines: Integration of Partial Differential Equations*. Academic Press, San Diego, 1991.

R. W. Setzer. *The* **odesolve** *Package: Solvers for Ordinary Differential Equations*, 2001. R package version 0.1-1.

K. Soetaert. **rootSolve***: Nonlinear Root Finding, Equilibrium and Steady-State Analysis of Ordinary Differential Equations*, 2009. R package version 1.6.

K. Soetaert and P. M. J. Herman. *A Practical Guide to Ecological Modelling. Using R as a Simulation Platform*. Springer, 2009. ISBN 978-1-4020-8623-6.

K. Soetaert and F. Meysman. **ReacTran***: Reactive Transport Modelling in 1D, 2D and 3D*, 2010. R package version 1.2.

K. Soetaert and T. Petzoldt. Inverse modelling, sensitivity and Monte Carlo analysis in R using package **FME**. *Journal of Statistical Software*, 33(3):1–28, 2010. URL http://www.jstatsoft.org/v33/i03/.

K. Soetaert, J. R. Cash, and F. Mazzia. **bvpSolve***: Solvers for Boundary Value Problems of Ordinary Differential Equations*, 2010a. R package version 1.2.

K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving differential equations in R: Package **deSolve**. *Journal of Statistical Software*, 33(9):1–25, 2010b. ISSN 1548-7660. URL http://www.jstatsoft.org/v33/i09.

K. Soetaert, T. Petzoldt, and R. W. Setzer. *R Package* **deSolve***: Writing Code in Compiled Languages*, 2010c. **deSolve** vignette - R package version 1.8.

K. Soetaert, T. Petzoldt, and R. W. Setzer. *R Package* **deSolve***: Solving Initial Value Differential Equations*, 2010d. **deSolve** vignette - R package version 1.8.

M. H. H. Stevens. *A Primer of Ecology with R*. Use R Series. Springer, 2009. ISBN: 978-0-387-89881-0.

G. Strang and G. Fix. *An Analysis of The Finite Element Method*. Prentice Hall, 1973.

C. W. Tornøe, H. Agersø, E. N. Jonsson, H. Madsen, and H. A. Nielsen. Non-linear mixed-effects pharmacokinetic/pharmacodynamic modelling in nlme using differential equations. *Computer Methods and Programs in Biomedicine*, 76:31–40, 2004.

B. van der Pol and J. van der Mark. Frequency demultiplication. *Nature*, 120:363–364, 1927.

Karline Soetaert
Netherlands Institute of Ecology
K.Soetaert@nioo.knaw.nl

Thomas Petzoldt
Technische Universität Dresden
Thomas.Petzoldt@tu-dresden.de

R. Woodrow Setzer
US Environmental Protection Agency
Setzer.Woodrow@epamail.epa.gov

Table 2: Summary of the main functions that solve differential equations.

Function	Package	Description
ode	deSolve	IVP of ODEs, full, banded or arbitrary sparse Jacobian
ode.1D	deSolve	IVP of ODEs resulting from 1-D reaction-transport problems
ode.2D	deSolve	IVP of ODEs resulting from 2-D reaction-transport problems
ode.3D	deSolve	IVP of ODEs resulting from 3-D reaction-transport problems
daspk	deSolve	IVP of DAEs of index ≤ 1, full or banded Jacobian
radau	deSolve	IVP of DAEs of index ≤ 3, full or banded Jacobian
dde	PBSddesolve	IVP of delay differential equations, based on Runge-Kutta formulae
dede	deSolve	IVP of delay differential equations, based on Adams and BDF formulae
bvpshoot	bvpSolve	BVP of ODEs; the shooting method
bvptwp	bvpSolve	BVP of ODEs; mono-implicit Runge-Kutta formula
bvpcol	bvpSolve	BVP of ODEs; collocation formula
steady	rootSolve	steady-state of ODEs; full, banded or arbitrary sparse Jacobian
steady.1D	rootSolve	steady-state of ODEs resulting from 1-D reaction-transport problems
steady.2D	rootSolve	steady-state of ODEs resulting from 2-D reaction-transport problems
steady.3D	rootSolve	steady-state of ODEs resulting from 3-D reaction-transport problems
tran.1D	ReacTran	numerical approximation of 1-D advective-diffusive transport problems
tran.2D	ReacTran	numerical approximation of 2-D advective-diffusive transport problems
tran.3D	ReacTran	numerical approximation of 3-D advective-diffusive transport problems

Table 3: Summary of the auxilliary functions that solve differential equations.

Function	Package	Description
lsoda	deSolve	IVP ODEs, full or banded Jacobian, automatic choice for stiff or non-stiff method
lsodar	deSolve	same as lsoda, but includes a root-solving procedure.
lsode, vode	deSolve	IVP ODEs, full or banded Jacobian, user specifies if stiff or non-stiff
lsodes	deSolve	IVP ODEs, arbitrary sparse Jacobian, stiff method
rk4, rk, euler	deSolve	IVP ODEs, using Runge-Kutta and Euler methods
zvode	deSolve	IVP ODEs, same as vode, but for complex variables
runsteady	rootSolve	steady-state ODEs by dynamically running, full or banded Jacobian
stode	rootSolve	steady-state ODEs by Newton-Raphson method, full or banded Jacobian
stodes	rootSolve	steady-state ODEs by Newton-Raphson method, arbitrary sparse Jacobian

Source References

by Duncan Murdoch

Abstract Since version 2.10.0, R includes expanded support for source references in R code and '.Rd' files. This paper describes the origin and purposes of source references, and current and future support for them.

One of the strengths of R is that it allows "computation on the language", i.e. the parser returns an R object which can be manipulated, not just evaluated. This has applications in quality control checks, debugging, and elsewhere. For example, the **codetools** package (Tierney, 2009) examines the structure of parsed source code to look for common programming errors. Functions marked by `debug()` can be executed one statement at a time, and the `trace()` function can insert debugging statements into any function.

Computing on the language is often enhanced by being able to refer to the original source code, rather than just to a deparsed (reconstructed) version of it based on the parsed object. To support this, we added source references to R 2.5.0 in 2007. These are attributes attached to the result of `parse()` or (as of 2.10.0) `parse_Rd()` to indicate where a particular part of an object originated. In this article I will describe their structure and how they are used in R. The article is aimed at developers who want to create debuggers or other tools that use the source references, at users who are curious about R internals, and also at users who want to use the existing debugging facilities. The latter group may wish to skip over the gory details and go directly to the section "Using Source References".

The R parsers

We start with a quick introduction to the R parser. The `parse()` function returns an R object of type `"expression"`. This is a list of statements; the statements can be of various types. For example, consider the R source shown in Figure 1.

```
1: x <- 1:10          # Initialize x
2: for (i in x) {
3:   print(i)         # Print each entry
4: }
5: x
```

Figure 1: The contents of 'sample.R'.

If we parse this file, we obtain an expression of length 3:

```
> parsed <- parse("sample.R")
> length(parsed)
[1] 3

> typeof(parsed)

[1] "expression"
```

The first element is the assignment, the second element is the `for` loop, and the third is the single `x` at the end:

```
> parsed[[1]]

x <- 1:10

> parsed[[2]]

for (i in x) {
    print(i)
}

> parsed[[3]]

x
```

The first two elements are both of type `"language"`, and are made up of smaller components. The difference between an `"expression"` and a `"language"` object is mainly internal: the former is based on the generic vector type (i.e. type `"list"`), whereas the latter is based on the `"pairlist"` type. Pairlists are rarely encountered explicitly in any other context. From a user point of view, they act just like generic vectors.

The third element `x` is of type `"symbol"`. There are other possible types, such as `"NULL"`, `"double"`, etc.: essentially any simple R object could be an element.

The comments in the source code and the white space making up the indentation of the third line are not part of the parsed object.

The `parse_Rd()` function parses '.Rd' documentation files. It also returns a recursive structure containing objects of different types (Murdoch and Urbanek, 2009; Murdoch, 2010).

Source reference structure

As described above, the result of `parse()` is essentially a list (the `"expression"` object) of objects that may be lists (the `"language"` objects) themselves, and so on recursively. Each element of this structure from the top down corresponds to some part of the source file used to create it: in our example, `parse[[1]]` corresponds to the first line of 'sample.R', `parse[[2]]` is the second through fourth lines, and `parse[[3]]` is the fifth line.

The comments and indentation, though helpful to the human reader, are not part of the parsed object. However, by default the parsed object does contain a `"srcref"` attribute:

```
> attr(parsed, "srcref")
```

```
[[1]]
x <- 1:10

[[2]]
for (i in x) {
  print(i)          # Print each entry
}

[[3]]
x
```

Although it appears that the "srcref" attribute contains the source, in fact it only references it, and the print.srcref() method retrieves it for printing. If we remove the class from each element, we see the true structure:

```
> lapply(attr(parsed, "srcref"), unclass)

[[1]]
[1] 1 1 1 9 1 9
attr(,"srcfile")
sample.R

[[2]]
[1] 2 1 4 1 1 1
attr(,"srcfile")
sample.R

[[3]]
[1] 5 1 5 1 1 1
attr(,"srcfile")
sample.R
```

Each element is a vector of 6 integers: (first line, first byte, last line, last byte, first character, last character). The values refer to the position of the source for each element in the original source file; the details of the source file are contained in a "srcfile" attribute on each reference.

The reason both bytes and characters are recorded in the source reference is historical. When they were introduced, they were mainly used for retrieving source code for display; for this, bytes are needed. Since R 2.9.0, they have also been used to aid in error messages. Since some characters take up more than one byte, users need to be informed about character positions, not byte positions, and the last two entries were added.

The "srcfile" attribute is also not as simple as it looks. For example,

```
> srcref <- attr(parsed, "srcref")[[1]]
> srcfile <- attr(srcref, "srcfile")
> typeof(srcfile)

[1] "environment"

> ls(srcfile)

[1] "Enc"       "encoding"  "filename"
[4] "timestamp" "wd"
```

The "srcfile" attribute is actually an environment containing an encoding, a filename, a timestamp, and a working directory. These give information about the file from which the parser was reading. The reason it is an environment is that environments are *reference* objects: even though all three source references contain this attribute, in actuality there is only one copy stored. This was done to save memory, since there are often hundreds of source references from each file.

Source references in objects returned by parse_Rd() use the same structure as those returned by parse(). The main difference is that in Rd objects source references are attached to *every* component, whereas parse() only constructs source references for complete statements, not for their component parts, and they are attached to the container of the statements. Thus for example a braced list of statements processed by parse() will receive a "srcref" attribute containing source references for each statement within, while the statements themselves will not hold their own source references, and subexpressions within each statement will not generate source references at all. In contrast the "srcref" attribute for a section in an '.Rd' file will be a source reference for the whole section, and each component part in the section will have its own source reference.

Relation to the "source" attribute

By default the R parser also creates an attribute named "source" when it parses a function definition. When available, this attribute is used by default in lieu of deparsing to display the function definition. It is unrelated to the "srcref" attribute, which is intended to *point to* the source, rather than to *duplicate* the source. An integrated development environment (IDE) would need to know the correspondence between R code in R and the true source, and "srcref" attributes are intended to provide this.

When are "srcref" attributes added?

As mentioned above, the parser adds a "srcref" attribute by default. For this, it is assumes that options("keep.source") is left at its default setting of TRUE, and that parse() is given a filename as argument file, or a character vector as argument text. In the latter case, there is no source file to reference, so parse() copies the lines of source into a "srcfilecopy" object, which is simply a "srcfile" object that contains a copy of the text.

Developers may wish to add source references in other situations. To do that, an object inheriting from class "srcfile" should be passed as the srcfile argument to parse().

The other situation in which source references are likely to be created in R code is when calling

`source()`. The `source()` function calls `parse()`, creating the source references, and then evaluates the resulting code. At this point newly created functions will have source references attached to the body of the function.

The section "Breakpoints" below discusses how to make sure that source references are created in package code.

Using source references

Error locations

For the most part, users need not be concerned with source references, but they interact with them frequently. For example, error messages make use of them to report on the location of syntax errors:

```
> source("error.R")

Error in source("error.R") : error.R:4:1: unexpected
'else'
3:   print( "less" )
4: else
   ^
```

A more recent addition is the use of source references in code being executed. When R evaluates a function, it evaluates each statement in turn, keeping track of any associated source references. As of R 2.10.0, these are reported by the debugging support functions `traceback()`, `browser()`, `recover()`, and `dump.frames()`, and are returned as an attribute on each element returned by `sys.calls()`. For example, consider the function shown in Figure 2.

```
1: # Compute the absolute value
2: badabs <- function(x) {
3:   if (x < 0)
4:     x <- -x
5:   x
6: }
```

Figure 2: The contents of 'badabs.R'.

This function is syntactically correct, and works to calculate the absolute value of scalar values, but is not a valid way to calculate the absolute values of the elements of a vector, and when called it will generate an incorrect result and a warning:

```
> source("badabs.R")
> badabs( c(5, -10) )

[1]   5 -10

Warning message:
In if (x < 0) x <- -x :
  the condition has length > 1 and only the first
  element will be used
```

In this simple example it is easy to see where the problem occurred, but in a more complex function it might not be so simple. To find it, we can convert the warning to an error using

```
> options(warn=2)
```

and then re-run the code to generate an error. After generating the error, we can display a stack trace:

```
> traceback()

5: doWithOneRestart(return(expr), restart)
4: withOneRestart(expr, restarts[[1L]])
3: withRestarts({
       .Internal(.signalCondition(
           simpleWarning(msg, call), msg, call))
       .Internal(.dfltWarn(msg, call))
   }, muffleWarning = function() NULL) at badabs.R#2
2: .signalSimpleWarning("the condition has length
       > 1 and only the first element will be used",
       quote(if (x < 0) x <- -x)) at badabs.R#3
1: badabs(c(5, -10))
```

To read a traceback, start at the bottom. We see our call from the console as line "`1:`", and the warning being signalled in line "`2:`". At the end of line "`2:`" it says that the warning originated "`at badabs.R#3`", i.e. line 3 of the 'badabs.R' file.

Breakpoints

Users may also make use of source references when setting breakpoints. The `trace()` function lets us set breakpoints in particular R functions, but we need to specify which function and where to do the setting. The `setBreakpoint()` function is a more friendly front end that uses source references to construct a call to `trace()`. For example, if we wanted to set a breakpoint on 'badabs.R' line 3, we could use

```
> setBreakpoint("badabs.R#3")

D:\svn\papers\srcrefs\badabs.R#3:
 badabs step 2 in <environment: R_GlobalEnv>
```

This tells us that we have set a breakpoint in step 2 of the function `badabs` found in the global environment. When we run it, we will see

```
> badabs( c(5, -10) )

badabs.R#3
Called from: badabs(c(5, -10))

Browse[1]>
```

telling us that we have broken into the browser at the requested line, and it is waiting for input. We could then examine `x`, single step through the code, or do any other action of which the browser is capable.

By default, most packages are built without source reference information, because it adds quite substantially to the size of the code. However, setting

the environment variable `R_KEEP_PKG_SOURCE=yes` before installing a source package will tell R to keep the source references, and then breakpoints may be set in package source code. The `envir` argument to `setBreakpoints()` will need to be set in order to tell it to search outside the global environment when setting breakpoints.

The `#line` directive

In some cases, R source code is written by a program, not by a human being. For example, `Sweave()` extracts lines of code from Sweave documents before sending the lines to R for parsing and evaluation. To support such preprocessors, the R 2.10.0 parser recognizes a new directive of the form

```
#line nn "filename"
```

where `nn` is an integer. As with the same-named directive in the C language, this tells the parser to assume that the next line of source is line `nn` from the given filename for the purpose of constructing source references. The `Sweave()` function doesn't currently make use of this, but in the future, it (and other preprocessors) could output `#line` directives so that source references and syntax errors refer to the original source location rather than to an intermediate file.

The `#line` directive was a late addition to R 2.10.0. Support for this in `Sweave()` appeared in R 2.12.0.

The future

The source reference structure could be improved. First, it adds quite a lot of bulk to R objects in memory. Each source reference is an integer vector of length 6 with a class and `"srcfile"` attribute. It is hard to measure exactly how much space this takes because much is shared with other source references, but it is on the order of 100 bytes per reference. Clearly a more efficient design is possible, at the expense of moving support code to C from R. As part of this move, the use of environments for the `"srcfile"` attribute could be dropped: they were used as the only available R-level reference objects. For developers, this means that direct access to particular parts of a source reference should be localized as much as possible: They should write functions to extract particular information, and use those functions where needed, rather than extracting information directly. Then, if the implementation changes, only those extractor functions will need to be updated.

Finally, source level debugging could be implemented to make use of source references, to single step through the actual source files, rather than displaying a line at a time as the `browser()` does.

Bibliography

D. Murdoch. Parsing Rd files. 2010. URL `http://developer.r-project.org/parseRd.pdf`.

D. Murdoch and S. Urbanek. The new R help system. *The R Journal*, 1/2:60–65, 2009.

L. Tierney. *codetools: Code Analysis Tools for R*, 2009. R package version 0.2-2.

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

hglm: A Package for Fitting Hierarchical Generalized Linear Models

by Lars Rönnegård, Xia Shen and Moudud Alam

Abstract We present the **hglm** package for fitting hierarchical generalized linear models. It can be used for linear mixed models and generalized linear mixed models with random effects for a variety of links and a variety of distributions for both the outcomes and the random effects. Fixed effects can also be fitted in the dispersion part of the model.

Introduction

The **hglm** package (Alam et al., 2010) implements the estimation algorithm for hierarchical generalized linear models (HGLM; Lee and Nelder, 1996). The package fits generalized linear models (GLM; McCullagh and Nelder, 1989) with random effects, where the random effect may come from a distribution conjugate to one of the exponential-family distributions (normal, gamma, beta or inverse-gamma). The user may explicitly specify the design matrices both for the fixed and random effects. In consequence, correlated random effects, as well as random regression models can be fitted. The dispersion parameter can also be modeled with fixed effects.

The main function is `hglm()` and the input is specified in a similar manner as for `glm()`. For instance,

```
R> hglm(fixed = y ~ week, random = ~ 1|ID,
        family = binomial(link = logit))
```

fits a logit model for `y` with `week` as fixed effect and `ID` representing the clusters for a normally distributed random intercept. Given an `hglm` object, the standard generic functions are `print()`, `summary()` and `plot()`.

Generalized linear mixed models (GLMM) have previously been implemented in several R functions, such as the `lmer()` function in the **lme4** package (Bates and Maechler, 2010) and the `glmmPQL()` function in the **MASS** package (Venables and Ripley, 2002). In GLMM, the random effects are assumed to be Gaussian whereas the `hglm()` function allows other distributions to be specified for the random effect. The `hglm()` function also extends the fitting algorithm of the **dglm** package (Dunn and Smyth, 2009) by including random effects in the linear predictor for the mean, i.e. it extends the algorithm so that it can cope with mixed models. Moreover, the model specification in `hglm()` can be given as a formula or alternatively in terms of `y`, `X`, `Z` and `X.disp`. Here `y` is the vector of observed responses, `X` and `Z` are the design matrices for the fixed and random effects, respectively, in the linear predictor for the means and `X.disp` is the design matrix for the fixed effects in the dispersion parameter. This enables a more flexible modeling of the random effects than specifying the model by an R formula. Consequently, this option is not as user friendly but gives the user the possibility to fit random regression models and random effects with known correlation structure.

The **hglm** package produces estimates of fixed effects, random effects and variance components as well as their standard errors. In the output it also produces diagnostics such as deviance components and leverages.

Three illustrating models

The **hglm** package makes it possible to

1. include fixed effects in a model for the residual variance,
2. fit models where the random effect distribution is not necessarily Gaussian,
3. estimate variance components when we have correlated random effects.

Below we describe three models that can be fitted using `hglm()`, which illustrate these three points. Later, in the Examples section, five examples are presented that include the R syntax and output for the `hglm()` function.

Linear mixed model with fixed effects in the residual variance

We start by considering a normal-normal model with heteroscedastic residual variance. In biology, for instance, this is important if we wish to model a random genetic effect (e.g., Rönnegård and Carlborg, 2007) for a trait y, where the residual variance differs between the sexes.

For the response y and observation number i we have:

$$y_i \mid \beta, u, \beta_d \sim N\left(X_i\beta + Z_i u, \exp\left(X_{d,i}\beta_d\right)\right)$$
$$u \sim MVN\left(0, \mathbf{I}\sigma_u^2\right)$$

where β are the fixed effects in the mean part of the model, the random effect u represents random variation among clusters of observations and β_d is the fixed effect in the residual variance part of the model. The variance of the random effect u is given by σ_u^2.

The subscript i for the matrices $\mathbf{X}$, $\mathbf{Z}$, and $\mathbf{X}_d$ indicates the i'th row. Here, a log link function is used for the residual variance and the model for the residual variance is therefore given by $\exp(X_{d,i}\beta_d)$. In the more general GLM notation, the residual variance here is described by the dispersion term ϕ, so we have $\log(\phi_i) = X_{d,i}\beta_d$.

This model cannot be fitted with the **dglm** package, for instance, because we have random effects in the mean part of the model. It is also beyond the scope of the `lmer()` function since we allow a model for the residual variance.

The implementation in `hglm()` for this model is demonstrated in Example 2 in the Examples section below.

A Poisson model with gamma distributed random effects

For dependent count data it is common to model a Poisson distributed response with a gamma distributed random effect (Lee et al., 2006). If we assume no overdispersion conditional on u and thereby have a fixed dispersion term, this model may be specified as:

$$E(y_i \mid \beta, u) = \exp(X_i\beta + Z_i v)$$

where a level j in the random effect v is given by $v_j = \log(u_j)$ and u_j are iid with gamma distribution having mean and variance: $E(u_j) = 1$, $var(u_j) = \lambda$.

This model can also be fitted with the **hglm** package, since it extends existing GLMM functions (e.g. `lmer()`) to allow a non-normal distribution for the random effect. Later on, in Example 3, we show the `hglm()` code used for fitting a gamma-Poisson model with fixed effects included in the dispersion parameter.

A linear mixed model with a correlated random effect

In animal breeding it is important to estimate variance components prior to ranking of animal performances (Lynch and Walsh, 1998). In such models the genetic effect of each animal is modeled as a level in a random effect and the correlation structure $\mathbf{A}$ is a matrix with known elements calculated from the pedigree information. The model is given by

$$y_i \mid \beta, u \sim N\left(X_i\beta + Z_i u, \sigma_e^2\right)$$
$$u \sim MVN\left(0, \mathbf{A}\sigma_u^2\right)$$

This may be reformulated as (see Lee et al., 2006; Rönnegård and Carlborg, 2007)

$$y_i \mid \beta, u \sim N\left(X_i\beta + Z_i^* u^*, \sigma_e^2\right)$$
$$u^* \sim MVN(0, \mathbf{I}\sigma_u^2)$$

where $\mathbf{Z}^* = \mathbf{ZL}$ and $\mathbf{L}$ is the Cholesky factorization of $\mathbf{A}$.

Thus the model can be fitted using the `hglm()` function with a user-specified input matrix `Z` (see R code in Example 4 below).

Overview of the fitting algorithm

The fitting algorithm is described in detail in Lee et al. (2006) and is summarized as follows. Let n be the number of observations and k be the number of levels in the random effect. The algorithm is then:

1. Initialize starting values.
2. Construct an augmented model with response $y_{aug} = \begin{pmatrix} y \\ E(u) \end{pmatrix}$.
3. Use a GLM to estimate β and v given the vector ϕ and the dispersion parameter for the random effect λ. Save the deviance components and leverages from the fitted model.
4. Use a gamma GLM to estimate β_d from the first n deviance components d and leverages h obtained from the previous model. The response variable and weights for this model are $d/(1-h)$ and $(1-h)/2$, respectively. Update the dispersion parameter by putting ϕ equal to the predicted response values for this model.
5. Use a similar GLM as in Step 4 to estimate λ from the last k deviance components and leverages obtained from the GLM in Step 3.
6. Iterate between steps 3-5 until convergence.

For a more detailed description of the algorithm in a particular context, see below.

H-likelihood theory

Let y be the response and u an unobserved random effect. The **hglm** package fits a hierarchical model $y \mid u \sim f_m(\mu, \phi)$ and $u \sim f_d(\psi, \lambda)$ where f_m and f_d are specified distributions for the mean and dispersion parts of the model.

We follow the notation of Lee and Nelder (1996), which is based on the GLM terminology by McCullagh and Nelder (1989). We also follow the likelihood approach where the model is described in terms of likelihoods. The conditional (log-)likelihood for y given u has the form of a GLM

$$\ell(\theta', \phi; y \mid u) = \frac{y\theta' - b(\theta')}{a(\phi)} + c(y, \phi) \qquad (1)$$

where θ' is the canonical parameter, ϕ is the dispersion term, μ' is the conditional mean of y given u

where $\eta' = g(\mu')$, i.e. $g()$ is a link function for the GLM. The linear predictor is given by $\eta' = \eta + v$ where $\eta = X\beta$ and $v = v(u)$ for some strict monotonic function of u. The link function $v(u)$ should be specified so that the random effects occur linearly in the linear predictor to ensure meaningful inference from the h-likelihood (Lee et al., 2007). The *h-likelihood* or hierarchical likelihood is defined by

$$h = \ell(\theta', \phi; y \mid u) + \ell(\alpha; v) \tag{2}$$

where $\ell(\alpha; v)$ is the log density for v with parameter α. The estimates of β and v are given by $\frac{\partial h}{\partial \beta} = 0$ and $\frac{\partial h}{\partial v} = 0$. The dispersion components are estimated by maximizing the *adjusted profile h-likelihood*

$$h_p = \left(h - \frac{1}{2}\log|-\frac{1}{2\pi}H| \right)_{\beta=\hat{\beta}, v=\hat{v}} \tag{3}$$

where H is the Hessian matrix of the h-likelihood. The dispersion term ϕ can be connected to a linear predictor $X_d\beta_d$ given a link function $g_d(\cdot)$ with $g_d(\phi) = X_d\beta_d$. The adjusted profile likelihoods of ℓ and h may be used for inference of β, v and the dispersion parameters ϕ and λ (pp. 186 in Lee et al., 2006). More detail and discussion of h-likelihood theory is presented in the **hglm** vignette.

Detailed description of the hglm fitting algorithm for a linear mixed model with heteroscedastic residual variance

In this section we describe the fitting algorithm in detail for a linear mixed model where fixed effects are included in the model for the residual variance. The extension to distributions other than Gaussian is described at the end of the section.

Lee and Nelder (1996) showed that linear mixed models can be fitted using a hierarchy of GLM by using an augmented linear model. The linear mixed model

$$y = \mathbf{X}b + \mathbf{Z}u + e$$
$$v = \mathbf{Z}\mathbf{Z}^T\sigma_u^2 + \mathbf{R}\sigma_e^2$$

where $\mathbf{R}$ is a diagonal matrix with elements given by the estimated dispersion model (i.e. ϕ defined below). In the first iteration of the HGLM algorithm, $\mathbf{R}$ is an identity matrix. The model may be written as an augmented weighted linear model:

$$y_a = \mathbf{T}_a\delta + e_a \tag{4}$$

where

$$y_a = \begin{pmatrix} y \\ 0_q \end{pmatrix} \qquad \mathbf{T}_a = \begin{pmatrix} \mathbf{X} & \mathbf{Z} \\ \mathbf{0} & \mathbf{I}_q \end{pmatrix}$$
$$\delta = \begin{pmatrix} b \\ u \end{pmatrix} \qquad e_a = \begin{pmatrix} e \\ -u \end{pmatrix}$$

Here, q is the number of columns in $\mathbf{Z}$, 0_q is a vector of zeros of length q, and $\mathbf{I}_q$ is the identity matrix of size $q \times q$. The variance-covariance matrix of the augmented residual vector is given by

$$V(e_a) = \begin{pmatrix} \mathbf{R}\sigma_e^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_q\sigma_u^2 \end{pmatrix}$$

Given σ_e^2 and σ_u^2, this weighted linear model gives the same estimates of the fixed and random effects (b and u respectively) as Henderson's mixed model equations (Henderson, 1976).

The estimates from weighted least squares are given by:

$$\mathbf{T}_a^t\mathbf{W}^{-1}\mathbf{T}_a\hat{\delta} = \mathbf{T}_a^t\mathbf{W}^{-1}y_a$$

where $\mathbf{W} \equiv V(e_a)$.

The two variance components are estimated iteratively by applying a gamma GLM to the residuals e_i^2 and u_i^2 with intercept terms included in the linear predictors. The leverages h_i for these models are calculated from the diagonal elements of the hat matrix:

$$\mathbf{H}_a = \mathbf{T}_a(\mathbf{T}_a^t\mathbf{W}^{-1}\mathbf{T}_a)^{-1}\mathbf{T}_a^t\mathbf{W}^{-1} \tag{5}$$

A gamma GLM is used to fit the dispersion part of the model with response

$$y_{d,i} = e_i^2/(1-h_i) \tag{6}$$

where $E(y_d) = \mu_d$ and $\mu_d \equiv \phi$ (i.e. σ_e^2 for a Gaussian reponse). The GLM model for the dispersion parameter is then specified by the link function $g_d(.)$ and the linear predictor $X_d\beta_d$, with prior weights $(1-h_i)/2$, for

$$g_d(\mu_d) = X_d\beta_d \tag{7}$$

Similarly, a gamma GLM is fitted to the dispersion term α (i.e. σ_u^2 for a GLMM) for the random effect v, with

$$y_{\alpha,j} = u_j^2/(1-h_{n+j}), j = 1,2,...,q \tag{8}$$

and

$$g_\alpha(\mu_\alpha) = \lambda \tag{9}$$

where the prior weights are $(1-h_{n+j})/2$ and the estimated dispersion term for the random effect is given by $\hat{\alpha} = g_\alpha^{-1}(\hat{\lambda})$.

The algorithm iterates by updating both $\mathbf{R} = diag(\hat{\phi})$ and $\sigma_u^2 = \hat{\alpha}$, and subsequently going back to Eq. (4).

For a non-Gaussian response variable y, the estimates are obtained simply by fitting a GLM instead of Eq. (4) and by replacing e_i^2 and u_j^2 with the deviance components from the augmented model (see Lee et al., 2006).

Implementation details

Distributions and link functions

There are two important classes of models that can be fitted in **hglm**: GLMM and conjugate HGLM. GLMMs have Gaussian random effects. Conjugate HGLMs have been commonly used partly due to the fact that explicit formulas for the marginal likelihood exist. HGLMs may be used to fit models in survival analysis (frailty models), where for instance the complementary-log-log link function can be used on binary responses (see e.g., Carling et al., 2004). The gamma distribution plays an important role in modeling responses with a constant coefficient of variation (see Chapter 8 in McCullagh and Nelder, 1989). For such responses with a gamma distributed random effect we have a gamma-gamma model. A summary of the most important models is given in Tables 1 and 2. Note that the random-effect distribution can be an arbitrary conjugate exponential-family distribution. For the specific case where the random-effect distribution is a conjugate to the distribution of y, this is called a *conjugate HGLM*. Further implementation details can be found in the **hglm** vignette.

Possible future developments

In the current version of `hglm()` it is possible to include a single random effect in the mean part of the model. An important development would be to include several random effects in the mean part of the model and also to include random effects in the dispersion parts of the model. The latter class of models is called Double HGLM and has been shown to be a useful tool for modeling heavy tailed distributions (Lee and Nelder, 2006).

The algorithm of `hglm()` gives true marginal likelihood estimates for the fixed effects in conjugate HGLM (Lee and Nelder, 1996, pp. 629), whereas for other models the estimates are approximated. Lee and co-workers (see Lee et al., 2006, and references therein) have developed higher-order approximations, which are not implemented in the current version of the **hglm** package. For such extensions, we refer to the commercially available GenStat software (Payne et al., 2007), the recently available R package **HGLMMM** (Molas, 2010) and also to coming updates of **hglm**.

Examples

Example 1: A linear mixed model

Data description The output from the `hglm()` function for a linear mixed model is compared to the results from the `lme()` function in the **nlme** (Pinheiro et al., 2009) package using simulated data. In the simulated data there are five clusters with 20 observations in each cluster. For the mean part of the model, the simulated intercept value is $\mu = 0$, the variance for the random effect is $\sigma_u^2 = 0.2$, and the residual variance is $\sigma_e^2 = 1.0$.

Both functions produce the same estimate of the fixed intercept effect of 0.1473 (s.e. 0.16) and also the same variance component estimates. The `summary.hglm()` function gives the estimate of the variance component for the random intercept (0.082) as well as the residual variance (0.84). It also gives the logarithm of the variance component estimates together with standard errors below the lines `Model estimates for the dispersion term` and `Dispersion model for the random effects`. The `lme()` function gives the square root of the variance component estimates.

The model diagnostics produced by the `plot.hglm` function are shown in Figures 1 and 2. The data are completely balanced and therefore produce equal leverages (hatvalues) for all observations and also for all random effects (Figure 1). Moreover, the assumption of the deviance components being gamma distributed is acceptable (Figure 2).

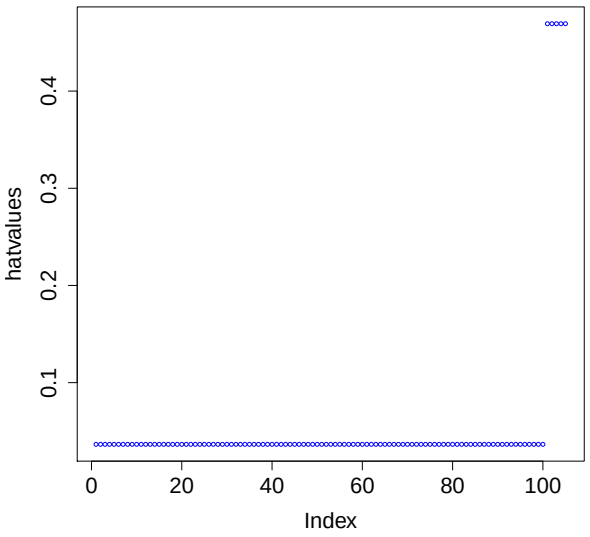


Figure 1: Hatvalues (i.e. diagonal elements of the augmented hat-matrix) for each observation 1 to 100, and for each level in the random effect (index 101-105).

Table 1: Commonly used distributions and link functions possible to fit with `hglm()`

Model name	$y \mid u$ distribution	Link $g(\mu)$	u distribution	Link $v(u)$
Linear mixed model	Gaussian	identity	Gaussian	identity
Binomial conjugate	Binomial	logit	Beta	logit
Binomial GLMM	Binomial	logit	Gaussian	identity
Binomial frailty	Binomial	comp-log-log	Gamma	log
Poisson GLMM	Poisson	log	Gaussian	identity
Poisson conjugate	Poisson	log	Gamma	log
Gamma GLMM	Gamma	log	Gaussian	identity
Gamma conjugate	Gamma	inverse	Inverse-Gamma	inverse
Gamma-Gamma	Gamma	log	Gamma	log

Table 2: `hglm` code for commonly used models

Model name	Setting for `family` argument	Setting for `rand.family` argument
Linear mixed model[a]	`gaussian(link = identity)`	`gaussian(link = identity)`
Beta-Binomial	`binomial(link = logit)`	`Beta(link = logit)`
Binomial GLMM	`binomial(link = logit)`	`gaussian(link = identity)`
Binomial frailty	`binomial(link = cloglog)`	`Gamma(link = log)`
Poisson GLMM	`poisson(link = log)`	`gaussian(link = identity)`
Poisson frailty	`poisson(link = log)`	`Gamma(link = log)`
Gamma GLMM	`Gamma(link = log)`	`gaussian(link = identity)`
Gamma conjugate	`Gamma(link = inverse)`	`inverse.gamma(link = inverse)`
Gamma-Gamma	`Gamma(link = log)`	`Gamma(link = log)`

[a] For example, the `hglm()` code for a linear mixed model is

```
hglm(family = gaussian(link = identity), rand.family = gaussian(link = identity), ...)
```

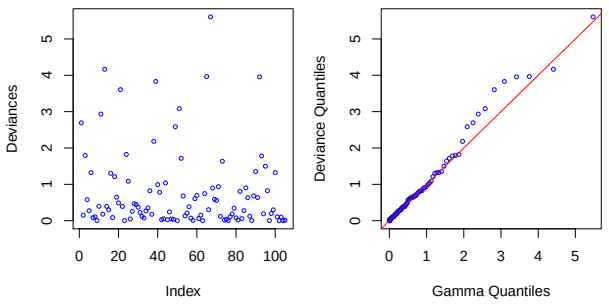


Figure 2: Deviance diagnostics for each observation and each level in the random effect.

The R code and output for this example is as follows:

```
R> set.seed(123)
R> n.clus <- 5 #No. of clusters
R> n.per.clus <- 20  #No. of obs. per cluster
R> sigma2_u <- 0.2 #Variance of random effect
R> sigma2_e <- 1 #Residual variance
R> n <- n.clus*n.per.clus
R> X <- matrix(1, n, 1)
R> Z <- diag(n.clus)%x%rep(1, n.per.clus)
R> a <- rnorm(n.clus, 0, sqrt(sigma2_u))
R> e <- rnorm(n, 0, sqrt(sigma2_e))
R> mu <- 0
R> y <- mu + Z%*%a + e
R> lmm <- hglm(y = y, X = X, Z = Z)
R> summary(lmm)
```

```
R> plot(lmm)

Call:
hglm.default(X = X, y = y, Z = Z)

DISPERSION MODEL
WARNING: h-likelihood estimates through EQL can be biased.
Model estimates for the dispersion term:[1] 0.8400608

Model estimates for the dispersion term:
Link = log
Effects:
  Estimate Std. Error
   -0.1743     0.1441

Dispersion = 1 is used in Gamma model on deviances
to calculate the standard error(s).
Dispersion parameter for the random effects
[1] 0.08211

Dispersion model for the random effects:
Link = log
Effects:
  Estimate Std. Error
   -2.4997     0.8682

Dispersion = 1 is used in Gamma model on deviances
to calculate the standard error(s).

MEAN MODEL
Summary of the fixed effects estimates
    Estimate Std. Error t value Pr(>|t|)
X.1   0.1473     0.1580   0.933    0.353
Note: P-values are based on 96 degrees of freedom
Summary of the random effects estimate
     Estimate Std. Error
[1,]  -0.3237     0.1971
[2,]  -0.0383     0.1971
```

```
[3,]   0.3108     0.1971
[4,]  -0.0572     0.1971
[5,]   0.1084     0.1971

EQL estimation converged in 5 iterations.

R> #Same analysis with the lme function
R> library(nlme)
R> clus <- rep(1:n.clus,
+              rep(n.per.clus, n.clus))
R> summary(lme(y ~ 0 + X,
+              random = ~ 1 | clus))

Linear mixed-effects model fit by REML
Data: NULL
      AIC      BIC    logLik
  278.635 286.4203 -136.3175

Random effects:
Formula: ~1 | clus
        (Intercept) Residual
StdDev:   0.2859608   0.9166

Fixed effects: y ~ 0 + X
      Value Std.Error DF   t-value p-value
X 0.1473009 0.1573412 95 0.9361873  0.3516

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-2.5834807 -0.6570612  0.0270673  0.6677986  2.1724148

Number of Observations: 100
Number of Groups: 5
```

Example 2: Analysis of simulated data for a linear mixed model with heteroscedastic residual variance

Data description Here, a heteroscedastic residual variance is added to the simulated data from the previous example. Given the explanatory variable x_d, the simulated residual variance is 1.0 for $x_d = 0$ and 2.72 for $x_d = 1$. The output shows that the variance of the random effect is 0.109, and that $\hat{\beta}_d = (-0.32, 1.47)$, i.e. the two residual variances are estimated as 0.72 and 3.16. (Code continued from Example 1)

```
R> beta.disp <- 1
R> X_d <- matrix(1, n, 2)
R> X_d[,2] <- rbinom(n, 1, .5)
R> colnames(X_d) <- c("Intercept", "x_d")
R> e <- rnorm(n, 0,
+       sqrt(sigma2_e*exp(beta.disp*X_d[,2])))
R> y <- mu + Z%*%a + e
R> summary(hglm(y = y, X = X, Z = Z,
+               X.disp = X_d))

Call:
hglm.default(X = X, y = y, Z = Z, X.disp = X_d)

DISPERSION MODEL
WARNING: h-likelihood estimates through EQL can be biased.
Model estimates for the dispersion term:
Link = log
Effects:
         Estimate Std. Error
Intercept  -0.3225     0.2040
x_d         1.4744     0.2881

Dispersion = 1 is used in Gamma model on deviances
to calculate the standard error(s).
Dispersion parameter for the random effects
[1] 0.1093

Dispersion model for the random effects:
Link = log
Effects:
  Estimate Std. Error
   -2.2135     0.8747

Dispersion = 1 is used in Gamma model on deviances
to calculate the standard error(s).
MEAN MODEL
Summary of the fixed effects estimates
    Estimate Std. Error t value Pr(>|t|)
X.1  -0.0535     0.1836  -0.291    0.771
Note: P-values are based on 96 degrees of freedom
Summary of the random effects estimate
     Estimate Std. Error
[1,]   0.0498     0.2341
[2,]  -0.2223     0.2276
[3,]   0.4404     0.2276
[4,]  -0.1786     0.2276
[5,]  -0.0893     0.2296

EQL estimation converged in 5 iterations.
```

Example 3: Fitting a Poisson model with gamma random effects, and fixed effects in the dispersion term

Data description We simulate a Poisson model with random effects and estimate the parameter in the dispersion term for an explanatory variable x_d. The estimated dispersion parameter for the random effects is 0.6556. (Code continued from Example 2)

```
R> u <- rgamma(n.clus,1)
R> eta <- exp(mu + Z%*%u)
R> y <- rpois(length(eta), eta)
R> gamma.pois <- hglm(y = y, X = X, Z = Z,
+                     X.disp = X_d,
+                     family = poisson(
+                     link = log),
+                     rand.family =
+                     Gamma(link = log))
R> summary(gamma.pois)

Call:
hglm.default(X = X, y = y, Z = Z,
    family = poisson(link = log),
    rand.family = Gamma(link = log), X.disp = X_d)

DISPERSION MODEL
WARNING: h-likelihood estimates through EQL can be biased.
Model estimates for the dispersion term:
Link = log
Effects:
         Estimate Std. Error
Intercept  -0.0186     0.2042
x_d         0.4087     0.2902

Dispersion = 1 is used in Gamma model on deviances
to calculate the standard error(s).
Dispersion parameter for the random effects
```

```
[1] 1.926

Dispersion model for the random effects:
Link = log
Effects:
  Estimate Std. Error
    0.6556     0.7081

Dispersion = 1 is used in Gamma model on deviances
to calculate the standard error(s).
MEAN MODEL
Summary of the fixed effects estimates
    Estimate Std. Error t value Pr(>|t|)
X.1   2.3363     0.6213    3.76 0.000293
---

Note: P-values are based on 95 degrees of freedom
Summary of the random effects estimate
     Estimate Std. Error
[1,]   1.1443     0.6209
[2,]  -1.6482     0.6425
[3,]  -2.5183     0.6713
[4,]  -1.0243     0.6319
[5,]   0.2052     0.6232

EQL estimation converged in 3 iterations.
```

Example 4: Incorporating correlated random effects in a linear mixed model - a genetics example

Data description The data consists of 2025 individuals from two generations where 1000 individuals have observed trait values y that are approximately normal (Figure 3). The data we analyze was simulated for the QTLMAS 2009 Workshop (Coster et al., 2010)[1]. A longitudinal growth trait was simulated. For simplicity we analyze only the values given on the third occasion at age 265 days.

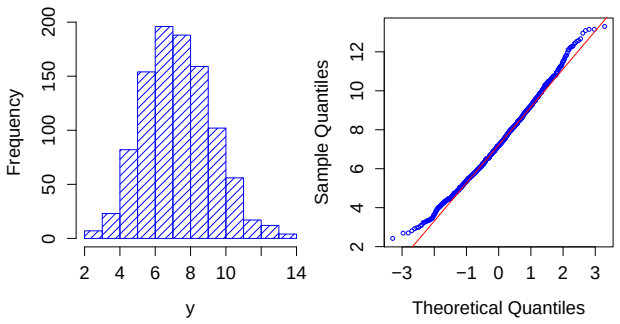


Figure 3: Histogram and qqplot for the analyzed trait.

We fitted a model with a fixed intercept and a random animal effect, a, where the correlation structure of a is given by the additive relationhip matrix **A** (which is obtained from the available pedigree information). An incidence matrix $\mathbf{Z}_0$ was constructed and relates observation number with id-number in the pedigree. For observation y_i coming from individual j in the ordered pedigree file $\mathbf{Z}_0[i,j] = 1$, and all other elements are 0. Let **L** be the Cholesky factorization of **A**, and $\mathbf{Z} = \mathbf{Z}_0\mathbf{L}$. The design matrix for the fixed effects, **X**, is a column of ones. The estimated variance components are $\hat{\sigma}^2_e = 2.21$ and $\hat{\sigma}^2_u = 1.50$.

The R code for this example is given below.

```
R> data(QTLMAS)
R> y <- QTLMAS[,1]
R> Z <- QTLMAS[,2:2026]
R> X <- matrix(1, 1000, 1)
R> animal.model <- hglm(y = y, X = X, Z = Z)
R> print(animal.model)

Call:
hglm.default(X = X, y = y, Z = Z)

Fixed effects:
     X.1
7.279766
Random effects:
   [1] -1.191733707  1.648604776  1.319427376 -0.928258503
   [5] -0.471083317 -1.058333534  1.011451565  1.879641994
   [9]  0.611705900 -0.259125073 -1.426788944 -0.005165978
...

Dispersion parameter for the mean model:[1] 2.211169
Dispersion parameter for the random effects:[1] 1.502516

EQL estimation converged in 2 iterations
```

Example 5: Binomial-beta model applied to seed germination data

Data description The seed germination data presented by Crowder (1978) has previously been analyzed using a binomial GLMM (Breslow and Clayton, 1993) and a binomial-beta HGLM (Lee and Nelder, 1996). The data consists of 831 observations from 21 germination plates. The effect of seed variety and type of root extract was studied in a 2×2 factorial lay-out. We fit the binomial-beta HGLM used by Lee and Nelder (1996) and setting `fix.disp = 1` in `hglm()` produces comparable estimates to the ones obtained by Lee and Nelder (with differences $< 2 \times 10^{-3}$). The beta distribution parameter α in Lee and Nelder (1996) was defined as $1/(2a)$ where a is the dispersion term obtained from `hglm()`. The output from the R code given below gives $\hat{a} = 0.0248$ and the corresponding estimate given in Lee and Nelder (1996) is $\hat{a} = 1/(2\hat{\alpha}) = 0.023$. We conclude that the **hglm** package produces similar results as the ones presented in Lee and Nelder (1996) and the dispersion parameters estimated using the EQL method in GenStat differ by less than 1%. Additional examples, together with comparisons to estimates produced by GenStat, are given in the **hglm** vignette included in the package on CRAN.

```
R> data(seeds)
R> germ <- hglm(
+    fixed = r/n ~ extract*I(seed=="073"),
```

[1] http://www.qtlmas2009.wur.nl/UK/Dataset

```
+     weights = n, data = seeds,
+     random = ~1|plate, family = binomial(),
+     rand.family = Beta(), fix.disp = 1)
R> summary(germ)

Call:
hglm.formula(family = binomial(), rand.family = Beta(),
    fixed = r/n ~ extract * I(seed == "O73"),
    random = ~1 | plate, data = seeds,
    weights = n, fix.disp = 1)

DISPERSION MODEL
WARNING: h-likelihood estimates through EQL can be biased.
Model estimates for the dispersion term:[1] 1

Model estimates for the dispersion term:
Link = log
Effects:
[1] 1

Dispersion = 1 is used in Gamma model on deviances to
calculate the standard error(s).
Dispersion parameter for the random effects
[1] 0.02483

Dispersion model for the random effects:
Link = log

Effects:
  Estimate Std. Error
   -3.6956     0.5304

Dispersion = 1 is used in Gamma model on deviances to
calculate the standard error(s).
MEAN MODEL
Summary of the fixed effects estimates
                              Estimate Std. Error t value
(Intercept)                    -0.5421    0.1928  -2.811
extractCucumber                 1.3386    0.2733   4.898
I(seed == "O73")TRUE            0.0751    0.3114   0.241
extractCucumber:I(seed=="O73")  -0.8257    0.4341  -1.902
                              Pr(>|t|)
(Intercept)                   0.018429
extractCucumber               0.000625
I(seed == "O73")TRUE          0.814264
extractCucumber:I(seed=="O73") 0.086343
---

Note: P-values are based on 10 degrees of freedom
Summary of the random effects estimate
      Estimate Std. Error
 [1,]  -0.2333     0.2510
 [2,]   0.0085     0.2328
 ...
[21,]  -0.0499     0.2953

EQL estimation converged in 7 iterations.
```

Summary

The hierarchical generalized linear model approach offers new possibilities to fit generalized linear models with random effects. The **hglm** package extends existing GLMM fitting algorithms to include fixed effects in a model for the residual variance, fits models where the random effect distribution is not necessarily Gaussian and estimates variance components for correlated random effects. For such models there are important applications in, for instance: genetics (Noh et al., 2006), survival analysis (Ha and Lee, 2005), credit risk modeling (Alam and Carling, 2008), count data (Lee et al., 2006) and dichotomous responses (Noh and Lee, 2007). We therefore expect that this new package will be of use for applied statisticians in several different fields.

Bibliography

M. Alam and K. Carling. Computationally feasible estimation of the covariance structure in generalized linear mixed models GLMM. *Journal of Statistical Computation and Simulation*, 78:1227–1237, 2008.

M. Alam, L. Ronnegard, and X. Shen. *hglm: Hierarchical Generalized Linear Models*, 2010. URL http://CRAN.R-project.org/package=hglm. R package version 1.1.1.

D. Bates and M. Maechler. *lme4: Linear mixed-effects models using S4 classes*, 2010. URL http://CRAN.R-project.org/package=lme4. R package version 0.999375-37.

N. E. Breslow and D. G. Clayton. Approximate inference in generalized linear mixed models. *Journal of the American Statistical Association*, 88:9–25, 1993.

K. Carling, L. Rönnegård, and K. Roszbach. An analysis of portfolio credit risk when counterparties are interdependent within industries. *Sveriges Riksbank Working Paper*, 168, 2004.

A. Coster, J. Bastiaansen, M. Calus, C. Maliepaard, and M. Bink. QTLMAS 2010: Simulated dataset. *BMC Proceedings*, 4(Suppl 1):S3, 2010.

M. J. Crowder. Beta-binomial ANOVA for proportions. *Applied Statistics*, 27:34–37, 1978.

P. K. Dunn and G. K. Smyth. *dglm: Double generalized linear models*, 2009. URL http://CRAN.R-project.org/package=dglm. R package version 1.6.1.

I. D. Ha and Y. Lee. Comparison of hierarchical likelihood versus orthodox best linear unbiased predictor approaches for frailty models. *Biometrika*, 92: 717–723, 2005.

C. R. Henderson. A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics*, 32(1):69–83, 1976.

Y. Lee and J. A. Nelder. Double hierarchical generalized linear models with discussion. *Applied Statistics*, 55:139–185, 2006.

Y. Lee and J. A. Nelder. Hierarchical generalized linear models with discussion. *J. R. Statist. Soc. B*, 58: 619–678, 1996.

Y. Lee, J. A. Nelder, and Y. Pawitan. *Generalized linear models with random effects*. Chapman & Hall/CRC, 2006.

Y. Lee, J. A. Nelder, and M. Noh. H-likelihood: problems and solutions. *Statistics and Computing*, 17: 49–55, 2007.

M. Lynch and B. Walsh. *Genetics and analysis of Quantitative Traits*. Sinauer Associates, Inc., 1998. ISBN 087893481.

P. McCullagh and J. A. Nelder. *Generalized linear models*. Chapman & Hall/CRC, 1989.

M. Molas. *HGLMMM: Hierarchical Generalized Linear Models*, 2010. URL http://CRAN.R-project.org/package=HGLMMM. R package version 0.1.1.

M. Noh and Y. Lee. REML estimation for binary data in GLMMs. *Journal of Multivariate Analysis*, 98:896–915, 2007.

M. Noh, B. Yip, Y. Lee, and Y. Pawitan. Multicomponent variance estimation for binary traits in family-based studies. *Genetic Epidemiology*, 30:37–47, 2006.

R. W. Payne, D. A. Murray, S. A. Harding, D. B. Baird, and D. M. Soutar. GenStat for Windows (10th edition) introduction, 2007. URL http://www.vsni.co.uk/software/genstat.

J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and the R Core team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2009. URL http://CRAN.R-project.org/package=nlme. R package version 3.1-96.

L. Rönnegård and Ö. Carlborg. Separation of base allele and sampling term effects gives new insights in variance component QTL analysis. *BMC Genetics*, 8(1), 2007.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0.

Lars Rönnegård
Statistics Unit
Dalarna University, Sweden
and
Department of Animal Breeding and Genetics
Swedish University of Agricultural Sciences, Sweden
lrn@du.se

Xia Shen
Department of Cell and Molecular Biology
Uppsala University, Sweden
and
Statistics Unit
Dalarna University, Sweden
xia.shen@lcb.uu.se

Moudud Alam
Statistics Unit
Dalarna University, Sweden
maa@du.se

dclone: Data Cloning in R

by Péter Sólymos

Abstract The **dclone** R package contains low level functions for implementing maximum likelihood estimating procedures for complex models using data cloning and Bayesian Markov Chain Monte Carlo methods with support for JAGS, WinBUGS and OpenBUGS.

Introduction

Hierarchical models, including generalized linear models with mixed random and fixed effects, are increasingly popular. The rapid expansion of applications is largely due to the advancement of the Markov Chain Monte Carlo (MCMC) algorithms and related software (Gelman et al., 2003; Gilks et al., 1996; Lunn et al., 2009). Data cloning is a statistical computing method introduced by Lele et al. (2007). It exploits the computational simplicity of the MCMC algorithms used in the Bayesian statistical framework, but it provides the maximum likelihood point estimates and their standard errors for complex hierarchical models. The use of the data cloning algorithm is especially valuable for complex models, where the number of unknowns increases with sample size (i.e. with latent variables), because inference and prediction procedures are often hard to implement in such situations.

The **dclone** R package (Sólymos, 2010) provides infrastructure for data cloning. Users who are familiar with Bayesian methodology can instantly use the package for maximum likelihood inference and prediction. Developers of R packages can build on the low level functionality provided by the package to implement more specific higher level estimation procedures for users who are not familiar with Bayesian methodology. This paper demonstrates the implementation of the data cloning algorithm, and presents a case study on how to write high level functions for specific modeling problems.

Theory of data cloning

Imagine a hypothetical situation where an experiment is repeated by k different observers, and all k experiments happen to result in exactly the same set of observations, $y^{(k)} = (y, y, \ldots, y)$. The likelihood function based on the combination of the data from these k experiments is $L(\theta, y^{(k)}) = [L(\theta, y)]^k$. The location of the maximum of $L(\theta, y^{(k)})$ exactly equals the location of the maximum of the function $L(\theta, y)$, and the Fisher information matrix based on this likelihood is k times the Fisher information matrix based on $L(\theta, y)$.

One can use MCMC methods to calculate the posterior distribution of the model parameters (θ) conditional on the data. Under regularity conditions, if k is large, the posterior distribution corresponding to k clones of the observations is approximately normal with mean $\hat{\theta}$ and variance $1/k$ times the inverse of the Fisher information matrix. When k is large, the mean of this posterior distribution is the maximum likelihood estimate and k times the posterior variance is the corresponding asymptotic variance of the maximum likelihood estimate if the parameter space is continuous. When some of the parameters are on the boundaries of their feasible space (Stram and Lee, 1994), point estimates can be correct, but currently the Fisher information cannot be estimated correctly by using data cloning. This is an area for further research, but such situations challenge other computing techniques as well.

Data cloning is a computational algorithm to compute maximum likelihood estimates and the inverse of the Fisher information matrix, and is related to simulated annealing (Brooks and Morgan, 1995). By using data cloning, the statistical accuracy of the estimator remains a function of the sample size and not of the number of cloned copies. Data cloning does not improve the statistical accuracy of the estimator by artificially increasing the sample size. The data cloning procedure avoids the analytical or numerical evaluation of high dimensional integrals, numerical optimization of the likelihood function, and numerical computation of the curvature of the likelihood function. Interested readers should consult Lele et al. (2007, 2010) for more details and mathematical proofs for the data cloning algorithm.

The data cloning algorithm

Consider the following Poisson generalized linear mixed model (GLMM) with a random intercept for i.i.d. observations of Y_i counts from $i = 1, 2, \ldots, n$ localities:

$$\begin{aligned} \alpha_i &\sim \text{normal}\left(0, \sigma^2\right) \\ \lambda_i &= \exp\left(\alpha_i + \boldsymbol{X}_i^{\mathsf{T}} \boldsymbol{\beta}\right) \\ Y_i \mid \lambda_i &\sim \text{Poisson}\left(\lambda_i\right) \end{aligned}$$

The corresponding code for the simulation with $\boldsymbol{\beta} = (1.8, -0.9)$, $\sigma = 0.2$, $x_i \sim \text{U}(0,1)$ is:

```
> library(dclone)
> set.seed(1234)
> n <- 50
> beta <- c(1.8, -0.9)
> sigma <- 0.2
> x <- runif(n, min = 0, max = 1)
> X <- model.matrix(~ x)
```

```
> alpha <- rnorm(n, mean = 0, sd = sigma)
> lambda <- exp(alpha + drop(X %*% beta))
> Y <- rpois(n, lambda)
```

The first step in the data cloning algorithm is to construct the full Bayesian model of the problem with proper prior distributions for unknown parameters. We use flat normal priors for βs and for $\log(\sigma)$. First we use the **rjags** (Plummer, 2010b) and **coda** (Plummer et al., 2010) R packages and the JAGS (Plummer, 2010a) software for model fitting. But the **dclone** package also supports WinBUGS (Spiegelhalter et al., 2003) and OpenBUGS (Spiegelhalter et al., 2007) via the R packages **R2WinBUGS** (Sturtz et al., 2005) and **BRugs** (Thomas et al., 2006), respectively. The corresponding model in the BUGS language is:

```
> glmm.model <- function() {
+    for (i in 1:n) {
+       Y[i] ~ dpois(lambda[i])
+       lambda[i] <- exp(alpha[i] +
+          inprod(X[i,], beta[1,]))
+       alpha[i] ~ dnorm(0, tau)
+    }
+    for (j in 1:np) {
+       beta[1,j] ~ dnorm(0, 0.001)
+    }
+    log.sigma ~ dnorm(0, 0.001)
+    sigma <- exp(log.sigma)
+    tau <- 1 / pow(sigma, 2)
+ }
```

Note that instead of writing the model into a file, we store it as an R function (see JAGS and WinBUGS documentation for how to correctly specify the model in the BUGS language). Although the BUGS and R syntaxes seem similar, the BUGS model function cannot be evaluated within R. Storing the BUGS model as an R function is handy, because the user does not have to manage different files when modeling. Nevertheless, the model can be supplied in a separate file by giving its name as character.

We also have to define the data as elements of a named list along with the names of nodes that we want to monitor (we can also set up initial values, number of burn-in iterations, number of iterations for the posterior sample, thinning values, etc.; see **dclone** package documentation for details). Now we can do the Bayesian inference by calling the `jags.fit` function:

```
> dat <- list(Y = Y, X = X, n = n,
+    np = ncol(X))
> mod <- jags.fit(dat,
+    c("beta", "sigma"), glmm.model, n.iter = 1000)
```

The output `mod` is an `"mcmc.list"` object, which can be explored by methods such as `summary` or `plot` provided by the **coda** package.

The **dclone** package provides the `bugs.fit` wrapper function for WinBUGS/OpenBUGS. The BUGS model needs to be changed to run smoothly in WinBUGS/OpenBUGS:

```
> glmm.model.bugs <- function() {
+    for (i in 1:n) {
+       Y[i] ~ dpois(lambda[i])
+       lambda[i] <- exp(alpha[i] +
+          inprod(X[i,], beta[1,]))
+       alpha[i] ~ dnorm(0, tau) %_% I(-5, 5)
+    }
+    for (j in 1:np) {
+       beta[1,j] ~ dnorm(0, 0.01) %_% I(-5, 5)
+    }
+    log.sigma ~ dnorm(0, 0.01) %_% I(-5, 5)
+    sigma <- exp(log.sigma)
+    tau <- 1 / pow(sigma, 2)
+ }
```

In the `bugs.fit` function, the settings besides the `data`, `params`, `model`, and `inits` arguments follow the settings in the `bugs`/`openbugs` functions in the **R2WinBUGS** package. This leads to some differences between the arguments of the `jags.fit` and the `bugs.fit` functions. For example `bugs.fit` uses `n.thin` instead of `thin`, and `n.burnin` is equivalent to `n.adapt + n.update` as compared to `jags.fit`. The `bugs.fit` can return the results either in `"mcmc.list"` or `"bugs"` format. The reason for leaving different arguments for `jags.fit` and `bugs.fit` is that the aim of the **dclone** package is not to make the MCMC platforms interchangeable, but to provide data cloning facility for each. It is easy to adapt an existing BUGS code for data cloning, but it sometimes can be tricky to adapt a JAGS code to WinBUGS and vice versa, because of differences between the two dialects (i.e. truncation, censoring, autoregressive priors, etc., see Plummer (2010b)).

Here are the results from the three MCMC platforms:

```
> mod.wb <- bugs.fit(dat, c("beta", "sigma"),
+    glmm.model.bugs, DIC = FALSE, n.thin = 1)
> mod.ob <- bugs.fit(dat, c("beta", "sigma"),
+    glmm.model.bugs, program = "openbugs",
+    DIC = FALSE, n.thin = 1)
```

```
> sapply(list(JAGS = mod, WinBUGS = mod.wb,
+    OpenBUGS = mod.ob), coef)
```

```
          JAGS WinBUGS OpenBUGS
beta[1]  1.893   1.910   1.9037
beta[2] -1.050  -1.074  -1.0375
sigma    0.161   0.130   0.0732
```

The idea in the next step of the data cloning algorithm is that instead of using the likelihood for the observed data, we use the likelihood corresponding to k copies (clones) of the data. Actually cloning (repeating) the data k times is important if the model includes unobserved (latent) variables: in this way latent variables are cloned as well, thus contributing to the likelihood. We can use the `rep` function to repeat the data vectors, but it is less convenient for e.g. matrices or data frames. Thus, there is the `dclone` generic function with methods for various R object classes:

```
> dclone(1:5, 1)
```

```
[1] 1 2 3 4 5

> dclone(1:5, 2)

 [1] 1 2 3 4 5 1 2 3 4 5
attr(,"n.clones")
[1] 2
attr(,"n.clones")attr(,"method")
[1] "rep"

> dclone(matrix(1:4, 2, 2), 2)

     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    1    3
[4,]    2    4
attr(,"n.clones")
[1] 2
attr(,"n.clones")attr(,"method")
[1] "rep"

> dclone(data.frame(a=1:2, b=3:4), 2)

    a b
1_1 1 3
2_1 2 4
1_2 1 3
2_2 2 4
```

The number of clones can be extracted by the `nclones` function; it returns `NULL` for $k = 1$ and k otherwise.

The BUGS data specification might contain some elements that we do not want to clone (e.g. `"np"`, the number of columns of the design matrix in this case). Thus the `dclone` method has different behaviour for lists, than for non list classes (including data frames). We can define which elements should not be cloned, or which should be multiplied by k instead of being cloned k times.

```
> dat2 <- dclone(dat, n.clones = 2,
+     multiply = "n", unchanged = "np")
> nclones(dat2)

[1] 2
attr(,"method")
      Y       X       n      np
  "rep"   "rep" "multi"      NA
```

The `"method"` attribute of the cloned object stores this information. There are three different ways of cloning (besides `NA` standing for unchanged): `"rep"` is for (longitudinal) repetitions, `"multi"` is for multiplication, and `"dim"` is repeating the data along an extra dimension (see later).

Now we do the model fitting with $k = 2$. The `"mcmc.list"` object inherits the information about the cloning:

```
> mod2 <- jags.fit(dat2,
+    c("beta", "sigma"), glmm.model, n.iter = 1000)
```

Similarly, the `bugs.fit` function takes care of the cloning information passed through the data argument:

```
> mod.wb2 <- bugs.fit(dat2, c("beta", "sigma"),
+    glmm.model.bugs, DIC = FALSE, n.thin = 1)
> mod.ob2 <- bugs.fit(dat2, c("beta", "sigma"),
+    glmm.model.bugs, program = "openbugs",
+    DIC = FALSE, n.thin = 1)
```

And here are the results based on $k = 2$ for the three MCMC platforms:

```
> sapply(list(JAGS = mod2, WinBUGS = mod.wb2,
+    OpenBUGS = mod.ob2), coef)

          JAGS WinBUGS OpenBUGS
beta[1]  1.918   1.905    1.896
beta[2] -1.114  -1.080   -1.078
sigma    0.207   0.187    0.243
```

For some models, indexing can be more complex, and simple repetitions of the data (`"rep"` method) are not appropriate. In case of non independent data (time series or spatial autoregressive models), cloning should be done over an extra dimension to ensure that clones are independent. For this purpose, one can use the `dcdim` function:

```
> (obj <- dclone(dcdim(data.matrix(1:5)), 2))

     clone.1 clone.2
[1,]       1       1
[2,]       2       2
[3,]       3       3
[4,]       4       4
[5,]       5       5
attr(,"n.clones")
[1] 2
attr(,"n.clones")attr(,"method")
[1] "dim"
attr(,"n.clones")attr(,"method")attr(,"drop")
[1] TRUE
```

If data cloning consists of repetitions of the data, our BUGS model usually does not need modifications. If we add an extra dimension to the data, the BUGS model and the data specification must reflect the extra dimension, too.

To demonstrate this, we consider a model and data set from Ponciano et al. (2009). They used the single-species population growth data from laboratory experiments of Gause (1934) with *Paramecium aurelia*. Gause initiated liquid cultures on day 0 at a concentration of two individuals per 0.5 cm^3 of culture media. Then, on days 2–19, he took daily 0.5 cm^3 samples of the microbe cultures and counted the number of cells in each sample. Ponciano et al. (2009) fitted discrete time stochastic models of population dynamics to describe Gause's data taking into account both process noise and observation error. The Beverton-Holt model incorporates a latent variable component (N_t, $t = 0,1,\ldots,q$) to describe an unobserved time series of actual population abundance. The latent variable component contains density dependence (β) and stochastic process noise (σ^2). The

model incorporates a Poisson observation component to account for variability caused by sampling:

$$\begin{aligned} \mu_t &= \log(\lambda) + \log(N_{t-1}) - \log(1 + \beta N_{t-1}) \\ \log(N_t) &\sim \text{normal}(\mu_t, \sigma^2) \\ Y_t \mid N_t &\sim \text{Poisson}(N_t) \end{aligned}$$

λ is the finite rate of increase in population abundance. The corresponding BUGS model is:

```
> beverton.holt <- function() {
+    for (j in 1:k) {
+       for(i in 2:(n+1)){
+          Y[(i-1),j] ~ dpois(exp(log.N[i,j]))
+          log.N[i,j] ~ dnorm(mu[i,j], 1 / sigma^2)
+          mu[i,j] <- log(lambda) + log.N[(i-1),j]
+              - log(1 + beta * exp(log.N[(i-1),j]))
+       }
+       log.N[1,j] ~ dnorm(mu0, 1 / sigma^2)
+    }
+    beta ~ dlnorm(-1, 1)
+    sigma ~ dlnorm(0, 1)
+    tmp ~ dlnorm(0, 1)
+    lambda <- tmp + 1
+    mu0 <- log(lambda) + log(2) - log(1 + beta * 2)
+ }
```

Note that besides the indexing for the time series, the model contains another dimension for the clones. We define the data set by using the `dcdim` method for cloning the observations. We include an element `k = 1` that will be multiplied to indicate how many clones (columns) are in the data, while `n` (number of observations) remains unchanged:

```
> paurelia <- c(17, 29, 39, 63, 185, 258, 267,
+    392, 510, 570, 650, 560, 575, 650, 550,
+    480, 520, 500)
> bhdat <- list(Y=dcdim(data.matrix(paurelia)),
+    n=length(paurelia), k=1)
> dcbhdat <- dclone(bhdat, n.clones = 5,
+    multiply = "k", unchanged = "n")

> bhmod <- jags.fit(dcbhdat,
+    c("lambda","beta","sigma"), beverton.holt,
+    n.iter=1000)

> coef(bhmod)

   beta  lambda   sigma
0.00218 2.18755 0.12777
```

Results compare well with estimates in Ponciano et al. (2009) ($\hat{\beta} = 0.00235$, $\hat{\lambda} = 2.274$, $\hat{\sigma} = 0.1274$).

Iterative model fitting

We can use the `dc.fit` function to iteratively fit the same model with various k values as described in Lele et al. (2010). The function takes similar arguments to `dclone` and `jags.fit` (or `bugs.fit`, if `flavour = "bugs"` is used). Because the information in the data overrides the priors by increasing the number of clones, we can improve MCMC convergence by making the priors more informative during the iterative fitting process. We achieve this by modifying the BUGS model for the Poisson GLMM example:

```
> glmm.model.up <- function() {
+    for (i in 1:n) {
+       Y[i] ~ dpois(lambda[i])
+       lambda[i] <- exp(alpha[i] +
+          inprod(X[i,], beta[1,]))
+       alpha[i] ~ dnorm(0, 1/sigma^2)
+    }
+    for (j in 1:np) {
+       beta[1,j] ~ dnorm(pr[j,1], pr[j,2])
+    }
+    log.sigma ~ dnorm(pr[(np+1),1], pr[(np+1),2])
+    sigma <- exp(log.sigma)
+    tau <- 1 / pow(sigma, 2)
+ }
```

We also define a function to update the priors. The function returns values for flat prior specification in the first iteration, and uses the updated posterior means (via the `coef` method) and data cloning standard errors (via the `dcsd` method) in the rest, because priors that have large probability mass near the maximum likelihood estimate require fewer clones to achieve the desired accuracy.

```
> upfun <- function(x) {
+    if (missing(x)) {
+       np <- ncol(X)
+       return(cbind(rep(0, np+1),
+           rep(0.001, np+1)))
+    } else {
+       ncl <- nclones(x)
+       if (is.null(ncl))
+          ncl <- 1
+       par <- coef(x)
+       se <- dcsd(x)
+       log.sigma <- mcmcapply(x[,"sigma"], log)
+       par[length(par)] <- mean(log.sigma)
+       se[length(se)] <- sd(log.sigma) * sqrt(ncl)
+       return(cbind(par, se))
+    }
+ }
```

Finally, we define prior specifications as part of the data (`"pr"`), and provide the updating function in the `dc.fit` call:

```
> updat <- list(Y = Y, X = X, n = n,
+    np = ncol(X), pr = upfun())
> k <- c(1, 5, 10, 20)
> dcmod <- dc.fit(updat, c("beta", "sigma"),
+    glmm.model.up, n.clones = k, n.iter = 1000,
+    multiply = "n", unchanged = "np",
+    update = "pr", updatefun = upfun)

> summary(dcmod)

Iterations = 1001:2000
Thinning interval = 1
Number of chains = 3
Sample size per chain = 1000
Number of clones = 20
```

```
1. Empirical mean and standard deviation for each
   variable, plus standard error of the mean:

          Mean     SD DC SD Naive SE
beta[1]  1.894 0.0368 0.164 0.000671
beta[2] -1.082 0.0734 0.328 0.001341
sigma    0.278 0.0256 0.114 0.000467
        Time-series SE R hat
beta[1]        0.00259  1.01
beta[2]        0.00546  1.01
sigma          0.00194  1.04

2. Quantiles for each variable:

          2.5%    25%   50%    75%  97.5%
beta[1]  1.823  1.869  1.89  1.920  1.964
beta[2] -1.230 -1.133 -1.08 -1.029 -0.943
sigma    0.226  0.260  0.28  0.296  0.323
```

The summary contains data cloning standard errors (`DC SD`) and $\hat{R}$ values for MCMC chain convergence (Gelman and Rubin, 1992).

Diagnostics

We can see how the increase in the number of clones affects our inferences on single nodes by using the `dctable` function. This function retrieves the information stored during the iterative fitting process (or can be used to compare more than one fitted model). Only the last MCMC object is returned by `dc.fit`, but descriptive statistics of the posterior distribution are stored in each step (Figure 1). The asymptotic convergence can be visually evaluated by plotting the posterior variances scaled by the variance for the model at $k = 1$ (or the smallest k). If scaled variances are decreasing at a $1/k$ rate and have reached a lower bound (say < 0.05), the data cloning algorithm has converged. If scaled variances are not decreasing at the proper rate, that might indicate identifiability issues (Lele et al., 2010). On the log scale, this graph should show an approximately linear decrease of log(scaled variance) vs. $\log(k)$ for each parameter (Figure 2).

```
> dct <- dctable(dcmod)
> plot(dct)

> plot(dct, type="log.var")
```

Lele et al. (2010) introduced diagnostic measures for checking the convergence of the data cloning algorithm which are based on the joint posterior distribution and not only on single parameters. These include calculating the largest eigenvalue of the posterior variance covariance matrix (`lambdamax.diag`), or calculating the mean squared error and another correlation-like fit statistic (r^2) based on a χ^2 approximation (`chisq.diag` with a `plot` method). The maximum eigenvalue reflects the degeneracy of the posterior distribution, while the two fit measures reflect the adequacy of the normal approximation. All three statistics should converge to zero as k increases. If this happens, different prior specifications are no longer influencing the results (Lele et al., 2007, 2010).

These measures and multivariate $\hat{R}$ values for MCMC chain convergence (Brooks and Gelman, 1997) are calculated during the iterations by `dc.fit` as well, and can be retrieved by the function `dcdiag`:

```
> dcdiag(dcmod)

  n.clones lambda.max ms.error r.squared r.hat
1        1    0.11538   0.1282   0.02103  1.66
2        5    0.02225   0.0229   0.00277  1.02
3       10    0.01145   0.0383   0.00612  1.01
4       20    0.00643   0.0241   0.00173  1.03
```

The data cloning algorithm requires that MCMC chains are properly mixed and the posterior distribution is nearly degenerate multivariate normal. These requirements have been satisfied in the case of the Poisson GLMM model. $\hat{R}$ values show better mixing properties of the MCMC chains with higher k values, and in this example it is expected, because we have used informative priors near the maximum likelihood estimates for the cases $k > 1$.

The functions `dctable` and `dcdiag` can be used to determine the number of clones required for a particular model and data set. Also, these diagnostic functions can alert the modeller when the model contains non-identifiable parameters. Lele et al. (2010) gives several examples; here we consider the normal-normal mixture:

$$\mu_i \sim \text{normal}(\gamma, \tau^2)$$
$$Y_i \mid \mu_i \sim \text{normal}(\mu_i, \sigma^2)$$

where the parameters $(\gamma, \sigma^2 + \tau^2)$ are known to be identifiable, but $(\gamma, \sigma^2, \tau^2)$ are not.

We simulate random observations under this model ($\gamma = 2.5, \sigma = 0.2, \tau = 0.5$) and fit the corresponding BUGS model:

```
> gamma <- 2.5
> sigma <- 0.2
> tau <- 0.5
> set.seed(2345)
> mu <- rnorm(n, gamma, tau)
> Y <- rnorm(n, mu, sigma)
> nn.model <- function() {
+    for (i in 1:n) {
+       Y[i] ~ dnorm(mu[i], prec1)
+       mu[i] ~ dnorm(gamma, prec2)
+    }
+    gamma ~ dnorm(0, 0.001)
+    log.sigma ~ dnorm(0, 0.001)
+    sigma <- exp(log.sigma)
+    prec1 <- 1 / pow(sigma, 2)
+    log.tau ~ dnorm(0, 0.001)
+    tau <- exp(log.tau)
+    prec2 <- 1 / pow(tau, 2)
+ }
> nndat <- list(Y = Y, n = n)
```

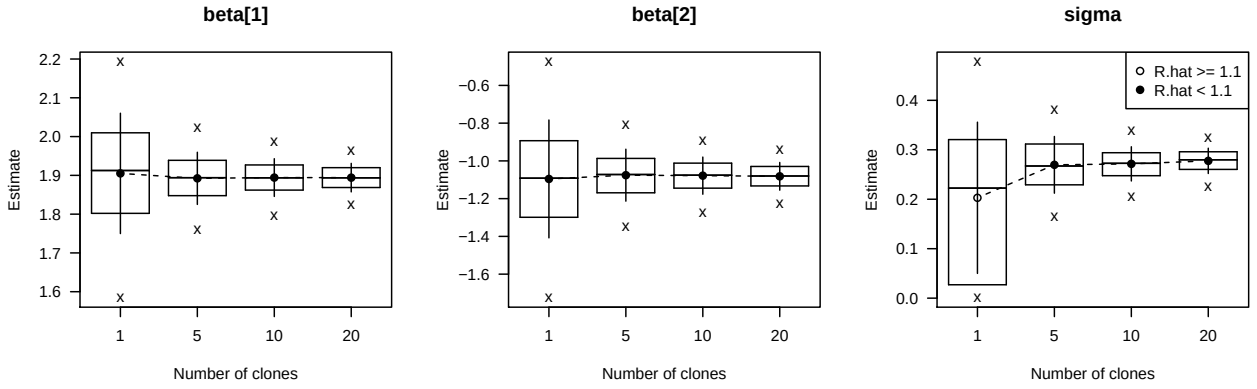


Figure 1: Summary statistics for the Poisson mixed model example. Means are converging towards the maximum likelihood estimates (points), standard errors (vertical lines) are getting shorter with increasing number of clones (95 and 50% quantile ranges and median also depicted).

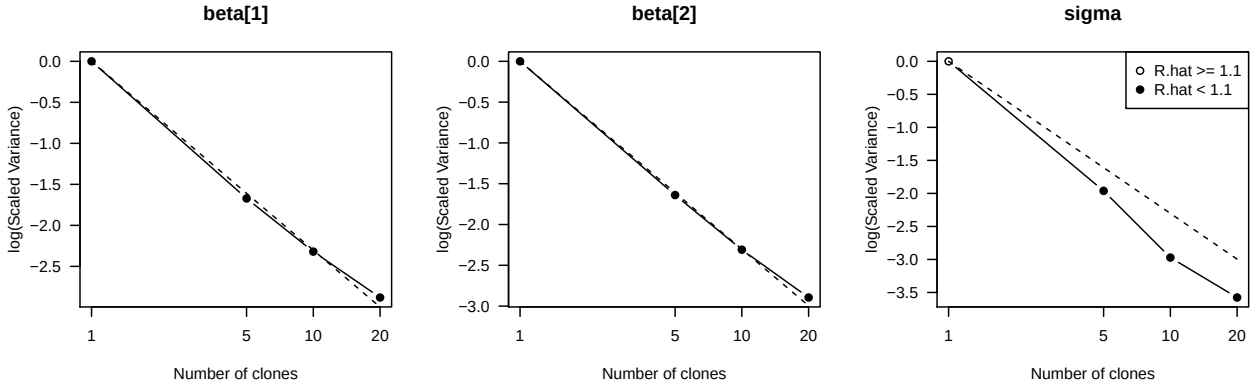


Figure 2: Convergence diagnostics for data cloning based on the Poisson mixed model example. Log of Scaled Variances should decrease linearly with $\log(k)$, the scaled variance value close to zero (< 0.05) indicates convergence of the data cloning algorithm.

```
> nnmod <- dc.fit(nndat, c("gamma","sigma","tau"),
+    nn.model, n.clones=c(1,10,20,30,40,50),
+    n.iter=1000, multiply="n")

> dcdiag(nnmod)

  n.clones lambda.max ms.error r.squared r.hat
1        1     0.0312    0.508   0.02985  1.18
2       10     0.0364    0.275   0.00355  2.06
3       20     1.2617    1.111   0.13714 50.15
4       30     0.1530    0.753   0.10267 12.91
5       40     1.7972    0.232   0.03770 92.87
6       50     1.8634    0.241   0.04003 15.72

> vars <- mcmcapply(nnmod[,c("sigma","tau")],
+    array)^2
> sigma^2 + tau^2

[1] 0.29

> summary(rowSums(vars))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.21    0.23    2.87    3.00    6.04    6.84
```

The high `r.hat` and the variable `lambda.max` and fit statistic values that are not converging to zero indicate possible problems with identifiability.

Inference and prediction

We can explore the results with methods defined for `"mcmc.list"` objects (many such methods are available in the **coda** package, e.g. `summary`, `plot`, etc.). The **dclone** package adds a few more methods: `coef` returns the mean of the posterior, `dcsd` the data cloning standard errors. Any function returning a scalar statistic can be passed via the `mcmcapply` function:

```
> coef(dcmod)

beta[1] beta[2]   sigma
  1.894  -1.082   0.278
```

```
> dcsd(dcmod)

beta[1] beta[2]   sigma
  0.164   0.328   0.114

> mcmcapply(dcmod, sd) * sqrt(nclones(dcmod))

beta[1] beta[2]   sigma
  0.164   0.328   0.114
```

The asymptotic multivariate normality can be used to get Wald-type confidence intervals for the estimates based on the inverse of the Fisher information matrix. The `vcov` method returns the inverse Fisher information matrix, the `confint` method calculates confidence intervals assuming multivariate normality for MCMC objects with $k > 1$:

```
> confint(dcmod)

          2.5 % 97.5 %
beta[1]  1.5718  2.217
beta[2] -1.7253 -0.438
sigma    0.0534  0.502

> vcov(dcmod)

         beta[1]  beta[2]    sigma
beta[1]  0.02705 -0.04604 -0.00291
beta[2] -0.04604  0.10783 -0.00156
sigma   -0.00291 -0.00156  0.01308
```

Confidence intervals can also be obtained via parametric bootstrap or based on profile likelihood (Ponciano et al., 2009), but these are not currently available in the **dclone** package and often require substantial user intervention.

These methods are handy when we make predictions. We can use the maximum likelihood estimates and the variance-covariance matrix defined as a multivariate normal node in the BUGS model. For the Poisson mixed model example, the BUGS model for prediction will look like:

```
> glmm.pred <- function() {
+    for (i in 1:n) {
+       Y[i] ~ dpois(lambda[i])
+       lambda[i] <- exp(mu[i])
+       mu[i] <- alpha[i] +
+          inprod(X[i,], beta[1,])
+       alpha[i] ~ dnorm(0, tau)
+    }
+    tmp[1:(np+1)] ~ dmnorm(param[], prec[,])
+    beta[1,1:np] <- tmp[1:np]
+    sigma <- tmp[(np+1)]
+    tau <- 1 / pow(sigma, 2)
+ }
```

Now we add the estimates and the precision matrix `prec` to the data (the `make.symmetric` function prevents some problems related to matrix symmetry and numerical precision), and define $\boldsymbol{X}$ for the predictions (now we simply use the observed values of the covariates). Then do the modeling as usual by sampling the node `"lambda"`:

```
> prec <- make.symmetric(solve(vcov(dcmod)))
> prdat <- list(X = X, n = nrow(X), np = ncol(X),
+    param = coef(dcmod), prec = prec)
> prmod <- jags.fit(prdat, "lambda", glmm.pred,
+    n.iter = 1000)
```

Writing high level functions

Suppose we want to provide a user friendly function to fit the Poisson mixed model with random intercept. We are now modeling the observed abundances (`count` based on point counts) of the Ovenbird (*Seiurus aurocapilla*) as a function of ecological site characteristics (upland/lowland, `uplow`) and percentage of total human disturbance around the sites (`thd` in the `ovenbird` data set). Data were collected from 182 sites in the Northern Boreal region of Alberta, Canada, between 2003 and 2008. Data were collected by the Alberta Biodiversity Monitoring Institute and are available at `http://www.abmi.ca`.

Our goal is to determine the effect of human disturbance on Ovenbird abundance, by controlling for site characteristics. But we know that other factors not taken into account, e.g. the amount of deciduous forest, might influence the abundance as well (Hobson and Bayne, 2002). So the random intercept will account for this unexplained environmental variability. The Poisson error component will account for random deviations from expected abundances (λ_i) and observed counts (Y_i) represent a realization of this quantity.

Here is the high level function for fitting the Poisson mixed model built on data cloning with a simple `print`, `summary` and `predict` method:

```
> glmmPois <- function(formula,
+ data = parent.frame(), n.clones, ...) {
+    lhs <- formula[[2]]
+    Y <- eval(lhs, data)
+    formula[[2]] <- NULL
+    rhs <- model.frame(formula, data)
+    X <- model.matrix(attr(rhs, "terms"), rhs)
+    dat <- list(n = length(Y), Y = Y,
+       X = X, np = ncol(X))
+    dcdat <- dclone(dat, n.clones,
+       multiply = "n", unchanged = "np")
+    mod <- jags.fit(dcdat, c("beta", "sigma"),
+       glmm.model, ...)
+    coefs <- coef(mod)
+    names(coefs) <- c(colnames(X),
+       "sigma")
+    rval <- list(coefficients = coefs,
+       call = match.call(),
+       mcmc = mod, y = Y, x = rhs,
+       model = X, formula = formula)
+    class(rval) <- "glmmPois"
+    rval
+ }
> print.glmmPois <- function(x, ...) {
+    cat("glmmPois model\n\n")
+    print(format(coef(x), digits = 4),
```

```
+       print.gap = 2, quote = FALSE)
+    cat("\n")
+    invisible(x)
+ }
> summary.glmmPois <- function(object, ...) {
+    x <- cbind("Estimate" = coef(object),
+       "Std. Error" = dcsd(object$mcmc),
+       confint(object$mcmc))
+    cat("Call:", deparse(object$call,
+       width.cutoff = getOption("width")),
+       "\n", sep="\n")
+    cat("glmmPois model\n\n")
+    printCoefmat(x, ...)
+    cat("\n")
+    invisible(x)
+ }
> predict.glmmPois <- function(object,
+ newdata = NULL, type = c("mu", "lambda", "Y"),
+ level = 0.95, ...){
+    prec <- solve(vcov(object$mcmc))
+    prec <- make.symmetric(prec)
+    param <- coef(object)
+    if (is.null(newdata)) {
+       X <- object$model
+    } else {
+       rhs <- model.frame(object$formula, newdata)
+       X <- model.matrix(attr(rhs, "terms"), rhs)
+    }
+    type <- match.arg(type)
+    prdat <- list(n = nrow(X), X = X,
+       np = ncol(X), param = param, prec = prec)
+    prval <- jags.fit(prdat, type, glmm.pred, ...)
+    a <- (1 - level)/2
+    a <- c(a, 1 - a)
+    rval <- list(fit = coef(prval),
+       ci.fit = quantile(prval, probs = a))
+    rval
+ }
```

Note that the functions `glmm.model` and `glmm.pred` containing the BUGS code are used within these R functions. This implementation works fine, but is not adequate when building a contributed R package, because functions such as `dnorm` and `inprod` are not valid R objects, etc. For R packages, the best way is to represent the BUGS model as a character vector with lines as elements, and put that inside the R function. The `custommodel` function of the **dclone** package can be used to create such character vectors and pass them to other **dclone** functions via the `model` argument.

Now we fit the model for the `ovenbird` data set to estimate the effect of human disturbance on Ovenbird abundance. We fit the model using the function `glmmPois`:

```
> data(ovenbird)
> obmod <- glmmPois(count ~ uplow + thd,
+    ovenbird, n.clones = 5, n.update = 1000,
+    n.iter = 1000)
```

Then print the object and inspect the summary,

```
> obmod

glmmPois model

 (Intercept)  uplowlowland           thd
     2.00312      -1.34242      -0.01647
       sigma
     1.19318

> summary(obmod)

Call:
glmmPois(formula = count ~ uplow + thd, data = ovenbird,
    n.clones = 5, n.update = 1000, n.iter = 1000)


glmmPois model

             Estimate Std. Error    2.5 % 97.5 %
(Intercept)   2.00312    0.13767  1.73328   2.27
uplowlowland -1.34242    0.21503 -1.76387  -0.92
thd          -0.01647    0.00569 -0.02763  -0.01
sigma         1.19318    0.09523  1.00653   1.38
```

Finally predict abundances as a function of disturbance (0–100%) by controlling for site characteristics (Figure 3):

```
> thd <- seq(0, 100, len = 101)
> ndata <- data.frame(uplow = rep("lowland",
+    length(thd)), thd = thd)
> levels(ndata$uplow) <- levels(ovenbird$uplow)
> obpred <- predict(obmod, ndata, "lambda")
```

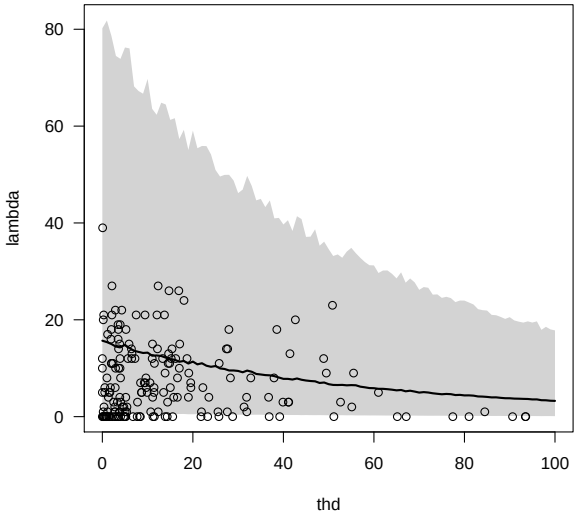


Figure 3: Expected Ovenbird abundance (`lambda`) as the function of percentage human disturbance (`thd`) based on the Poisson mixed model. Line represents the mean, gray shading indicates 95% prediction intervals. Points are observations.

Ovenbird abundance was significantly higher in upland sites, and human disturbance had a significant negative effect on expected Ovenbird abundance. Unexplained variation (σ^2 =1.425 $\pm$ 0.102 SE) was

substantial, thus the choice of the Poisson mixed model makes sense for this data set.

Summary

The data cloning algorithm is especially useful for complex models for which other likelihood based computational methods fail. The algorithm also can numerically reveal potential identifiability issues related to hierarchical models. The **dclone** package supports established MCMC software and provides low level functions to help implementing high level estimating procedures to get maximum likelihood inferences and predictions for more specialized problems based on the data cloning algorithm.

Acknowledgements

Subhash Lele, Khurram Nadeem and Gabor Grothendieck have provided invaluable suggestions and feedback on this work. Comments of Martyn Plummer and two anonymous reviewers greatly improved the quality of the paper. Funding was provided by the Alberta Biodiversity Monitoring Institute and the Natural Sciences and Engineering Research Council.

Péter Sólymos
Alberta Biodiversity Monitoring Institute
Department of Biological Sciences
University of Alberta
solymos@ualberta.ca

Bibliography

S. P. Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7: 434–455, 1997.

S. P. Brooks and B. J. T. Morgan. Optimization using simulated annealing. *Statistician*, 241–257:44, 1995.

G. F. Gause. *The struggle for existence*. Wilkins, Baltimore, Maryland, USA, 1934.

A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7:457–511, 1992.

A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian Data Analysis*. CRC Press, Boca Raton, 2 edition, 2003.

W. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall, London, 1996.

K. A. Hobson and E. M. Bayne. Breeding bird communities in boreal forest of western Canada: Consequences of "unmixing" the mixedwoods. *Condor*, 102:759–769, 2002.

S. R. Lele, B. Dennis, and F. Lutscher. Data cloning: easy maximum likelihood estimation for complex ecological models using Bayesian Markov chain Monte Carlo methods. *Ecology Letters*, 10:551–563, 2007.

S. R. Lele, K. Nadeem, and B. Schmuland. Estimability and likelihood inference for generalized linear mixed models using data cloning. *Journal of the American Statistical Association*, 2010. in press.

D. Lunn, D. Spiegelhalter, A. Thomas, and N. Best. The BUGS project: Evolution, critique and future directions. *Statistics in Medicine*, 28:3049–3067, 2009. with discussion.

M. Plummer. *JAGS Version 2.0.0 manual (April 26, 2010)*, 2010a. URL http://mcmc-jags.sourceforge.net.

M. Plummer. *rjags: Bayesian graphical models using MCMC*, 2010b. URL http://mcmc-jags.sourceforge.net. R package version 2.0.0-2.

M. Plummer, N. Best, K. Cowles, and K. Vines. *coda: Output analysis and diagnostics for MCMC*, 2010. URL http://cran.r-project.org/web/packages/coda/index.html. R package version 0.13-5.

J. M. Ponciano, M. L. Taper, B. Dennis, and S. R. Lele. Hierarchical models in ecology: confidence intervals, hypothesis testing, and model selection using data cloning. *Ecology*, 90:356–362, 2009.

P. Sólymos. *dclone: Data Cloning and MCMC Tools for Maximum Likelihood Methods*, 2010. URL http://cran.r-project.org/packages=dclone. R package version 1.2-0.

D. Spiegelhalter, A. Thomas, N. Best, and D. Lunn. *OpenBUGS User Manual, Version 3.0.2, September 2007*, 2007. URL http://mathstat.helsinki.fi/openbugs/.

D. J. Spiegelhalter, A. Thomas, N. G. Best, and D. Lunn. WinBUGS version 1.4 users manual. MRC Biostatistics Unit, Cambridge, 2003. URL http://www.mrc-bsu.cam.ac.uk/bugs/.

D. O. Stram and J. W. Lee. Variance components testing in the longitudinal mixed effects model. *Biometrics*, 50:1171–1177, 1994.

S. Sturtz, U. Ligges, and A. Gelman. R2WinBUGS: A package for running WinBUGS from R. *Journal of Statistical Software*, 12(3):1–16, 2005. URL http://www.jstatsoft.org.

A. Thomas, B. O'Hara, U. Ligges, and S. Sturtz. Making BUGS open. *R News*, 6(1):12–17, 2006. URL http://cran.r-project.org/doc/Rnews/.

stringr: modern, consistent string processing

by Hadley Wickham

Abstract String processing is not glamorous, but it is frequently used in data cleaning and preparation. The existing string functions in R are powerful, but not friendly. To remedy this, the **stringr** package provides string functions that are simpler and more consistent, and also fixes some functionality that R is missing compared to other programming languages.

Introduction

Strings are not glamorous, high-profile components of R, but they do play a big role in many data cleaning and preparations tasks. R provides a solid set of string operations, but because they have grown organically over time, they can be inconsistent and a little hard to learn. Additionally, they lag behind the string operations in other programming languages, so that some things that are easy to do in languages like Ruby or Python are rather hard to do in R. The **stringr** package aims to remedy these problems by providing a clean, modern interface to common string operations.

More concretely, **stringr**:

- Processes factors and characters in the same way.
- Gives functions consistent names and arguments.
- Simplifies string operations by eliminating options that you don't need 95% of the time (the other 5% of the time you can use the base functions).
- Produces outputs than can easily be used as inputs. This includes ensuring that missing inputs result in missing outputs, and zero length inputs result in zero length outputs.
- Completes R's string handling functions with useful functions from other programming languages.

To meet these goals, **stringr** provides two basic families of functions:

- basic string operations, and
- pattern matching functions which use regular expressions to detect, locate, match, replace, extract, and split strings.

These are described in more detail in the following sections.

Basic string operations

There are three string functions that are closely related to their base R equivalents, but with a few enhancements:

- `str_c` is equivalent to `paste`, but it uses the empty string ("") as the default separator and silently removes zero length arguments.
- `str_length` is equivalent to `nchar`, but it preserves NA's (rather than giving them length 2) and converts factors to characters (not integers).
- `str_sub` is equivalent to `substr` but it returns a zero length vector if any of its inputs are zero length, and otherwise expands each argument to match the longest. It also accepts negative positions, which are calculated from the left of the last character. The end position defaults to `-1`, which corresponds to the last character.
- `str_str<-` is equivalent to `substr<-`, but like `str_sub` it understands negative indices, and replacement strings not do need to be the same length as the string they are replacing.

Three functions add new functionality:

- `str_dup` to duplicate the characters within a string.
- `str_trim` to remove leading and trailing whitespace.
- `str_pad` to pad a string with extra whitespace on the left, right, or both sides.

Pattern matching

stringr provides pattern matching functions to **detect**, **locate**, **extract**, **match**, **replace**, and **split** strings:

- `str_detect` detects the presence or absence of a pattern and returns a logical vector. Based on `grepl`.
- `str_locate` locates the first position of a pattern and returns a numeric matrix with columns start and end. `str_locate_all` locates all matches, returning a list of numeric matrices. Based on `regexpr` and `gregexpr`.

- `str_extract` extracts text corresponding to the first match, returning a character vector. `str_extract_all` extracts all matches and returns a list of character vectors.
- `str_match` extracts capture groups formed by `()` from the first match. It returns a character matrix with one column for the complete match and one column for each group. `str_match_all` extracts capture groups from all matches and returns a list of character matrices.
- `str_replace` replaces the first matched pattern and returns a character vector. `str_replace_all` replaces all matches. Based on `sub` and `gsub`.
- `str_split_fixed` splits the string into a fixed number of pieces based on a pattern and returns a character matrix. `str_split` splits a string into a variable number of pieces and returns a list of character vectors.

Figure 1 shows how the simple (single match) form of each of these functions work.

Arguments

Each pattern matching function has the same first two arguments, a character vector of `string`s to process and a single `pattern` (regular expression) to match. The replace functions have an additional argument specifying the replacement string, and the split functions have an argument to specify the number of pieces.

Unlike base string functions, **stringr** only offers limited control over the type of matching. The `fixed()` and `ignore.case()` functions modify the pattern to use fixed matching or to ignore case, but if you want to use perl-style regular expressions or to match on bytes instead of characters, you're out of luck and you'll have to use the base string functions. This is a deliberate choice made to simplify these functions. For example, while `grepl` has six arguments, `str_detect` only has two.

Regular expressions

To be able to use these functions effectively, you'll need a good knowledge of regular expressions (Friedl, 1997), which this paper is not going to teach you. Some useful tools to get you started:

- A good reference sheet[1]
- A tool that allows you to interactively test[2] what a regular expression will match
- A tool to build a regular expression[3] from an input string

When writing regular expressions, I strongly recommend generating a list of positive (pattern should match) and negative (pattern shouldn't match) test cases to ensure that you are matching the correct components.

Functions that return lists

Many of the functions return a list of vectors or matrices. To work with each element of the list there are two strategies: iterate through a common set of indices, or use `mapply` to iterate through the vectors simultaneously. The first approach is usually easier to understand and is illustrated in Figure 2.

Conclusion

stringr provides an opinionated interface to strings in R. It makes string processing simpler by removing uncommon options, and by vigorously enforcing consistency across functions. I have also added new functions that I have found useful from Ruby, and over time, I hope users will suggest useful functions from other programming languages. I will continue to build on the included test suite to ensure that the package behaves as expected and remains bug free.

Bibliography

J. E. Friedl. *Mastering Regular Expressions*. O'Reilly, 1997. URL http://oreilly.com/catalog/9781565922570.

Hadley Wickham
Department of Statistics
Rice University
6100 Main St MS#138
Houston TX 77005-1827
USA
hadley@rice.edu

[1] http://www.regular-expressions.info/reference.html
[2] http://gskinner.com/RegExr/
[3] http://www.txt2re.com

```
library(stringr)
strings <- c(" 219 733 8965", "329-293-8753 ", "banana", "595 794 7569",
 "387 287 6718", "apple", "233.398.9187  ", "482 952 3315", "239 923 8115",
 "842 566 4692", "Work: 579-499-7527", "$1000", "Home: 543.355.3679")
phone <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"

# Which strings contain phone numbers?
str_detect(strings, phone)
strings[str_detect(strings, phone)]

# Where in the string is the phone number located?
loc <- str_locate(strings, phone)
loc
# Extract just the phone numbers
str_sub(strings, loc[, "start"], loc[, "end"])
# Or more conveniently:
str_extract(strings, phone)

# Pull out the three components of the match
str_match(strings, phone)

# Anonymise the data
str_replace(strings, phone, "XXX-XXX-XXXX")
```

Figure 1: Simple string matching functions for processing a character vector containing phone numbers (among other things).

```
library(stringr)
col2hex <- function(col) {
  rgb <- col2rgb(col)
  rgb(rgb["red", ], rgb["green", ], rgb["blue", ], max = 255)
}

# Goal replace colour names in a string with their hex equivalent
strings <- c("Roses are red, violets are blue", "My favourite colour is green")

colours <- str_c("\\b", colors(), "\\b", collapse="|")
# This gets us the colours, but we have no way of replacing them
str_extract_all(strings, colours)

# Instead, let's work with locations
locs <- str_locate_all(strings, colours)
sapply(seq_along(strings), function(i) {
  string <- strings[i]
  loc <- locs[[i]]

  # Convert colours to hex and replace
  hex <- col2hex(str_sub(string, loc[, "start"], loc[, "end"]))
  str_sub(string, loc[, "start"], loc[, "end"]) <- hex
  string
})
```

Figure 2: A more complex situation involving iteration through a string and processing matches with a function.

Bayesian Estimation of the GARCH(1,1) Model with Student-t Innovations

by David Ardia and Lennart F. Hoogerheide

Abstract This note presents the R package **bayesGARCH** which provides functions for the Bayesian estimation of the parsimonious and effective GARCH(1,1) model with Student-t innovations. The estimation procedure is fully automatic and thus avoids the tedious task of tuning an MCMC sampling algorithm. The usage of the package is shown in an empirical application to exchange rate log-returns.

Introduction

Research on changing volatility using time series models has been active since the pioneer paper by Engle (1982). From there, ARCH (AutoRegressive Conditional Heteroscedasticity) and GARCH (Generalized ARCH) type models grew rapidly into a rich family of empirical models for volatility forecasting during the 80's. These models are widespread and essential tools in financial econometrics.

In the GARCH(p,q) model introduced by Bollerslev (1986), the conditional variance at time t of the log-return y_t (of a financial asset or a financial index), denoted by h_t, is postulated to be a linear function of the squares of past q log-returns and past p conditional variances. More precisely:

$$h_t \doteq \alpha_0 + \sum_{i=1}^{q} \alpha_i y_{t-i}^2 + \sum_{j=1}^{p} \beta_j h_{t-j},$$

where the parameters satisfy the constraints $\alpha_i \geq 0$ $(i = 0,\ldots,q)$ and $\beta_j \geq 0$ $(j = 1,\ldots,p)$ in order to ensure a positive conditional variance. In most empirical applications it turns out that the simple specification $p = q = 1$ is able to reproduce the volatility dynamics of financial data. This has led the GARCH(1,1) model to become the *workhorse model* by both academics and practitioners. Given a model specification for h_t, the log-returns are then modelled as $y_t = \varepsilon_t h_t^{1/2}$, where ε_t are i.i.d. disturbances. Common choices for ε_t are Normal and Student-t disturbances. The Student-t specification is particularly useful, since it can provide the excess kurtosis in the conditional distribution that is often found in financial time series processes (unlike models with Normal innovations).

Until recently, GARCH models have mainly been estimated using the classical Maximum Likelihood technique. Several R packages provide functions for their estimation; see, e.g. **fGarch** (Wuertz and Chalabi, 2009), **rgarch** (Ghalanos, 2010) and **tseries** (Trapletti and Hornik, 2009). The Bayesian approach offers an attractive alternative which enables small sample results, robust estimation, model discrimination, model combination, and probabilistic statements on (possibly nonlinear) functions of the model parameters.

The package **bayesGARCH** (Ardia, 2007) implements the Bayesian estimation procedure described in Ardia (2008, chapter 5) for the GARCH(1,1) model with Student-t innovations. The approach, based on the work of Nakatsuma (1998), consists of a Metropolis-Hastings (MH) algorithm where the proposal distributions are constructed from auxiliary ARMA processes on the squared observations. This methodology avoids the time-consuming and difficult task, especially for non-experts, of choosing and tuning a sampling algorithm. The program is written in R with some subroutines implemented in C in order to speed up the simulation procedure. The validity of the algorithm as well as the correctness of the computer code have been verified by the method of Geweke (2004).

Model, priors and MCMC scheme

A GARCH(1,1) model with Student-t innovations for the log-returns $\{y_t\}$ may be written via data augmentation (see Geweke, 1993) as

$$\begin{aligned} y_t &= \varepsilon_t \left(\tfrac{\nu-2}{\nu}\omega_t h_t\right)^{1/2} \quad t = 1,\ldots,T \\ \varepsilon_t &\overset{iid}{\sim} \mathcal{N}(0,1) \\ \omega_t &\overset{iid}{\sim} \mathcal{IG}\left(\frac{\nu}{2},\frac{\nu}{2}\right) \\ h_t &\doteq \alpha_0 + \alpha_1 y_{t-1}^2 + \beta h_{t-1}, \end{aligned} \tag{1}$$

where $\alpha_0 > 0$, $\alpha_1, \beta \geq 0$ and $\nu > 2$; $\mathcal{N}(0,1)$ denotes the standard normal distribution; $\mathcal{IG}$ denotes the inverted gamma distribution. The restriction on the degrees of freedom parameter ν ensures the conditional variance to be finite and the restrictions on the GARCH parameters α_0, α_1 and β guarantee its positivity. We emphasize the fact that only positivity constraints are implemented in the MH algorithm; no stationarity conditions are imposed in the simulation procedure.

In order to write the likelihood function, we define the vectors $y \doteq (y_1,\ldots,y_T)'$, $\omega \doteq (\omega_1,\ldots,\omega_T)'$ and $\alpha \doteq (\alpha_0,\alpha_1)'$. We regroup the model parameters into the vector $\psi \doteq (\alpha,\beta,\nu)$. Then, upon defining the $T \times T$ diagonal matrix

$$\Sigma \doteq \Sigma(\psi,\omega) = \operatorname{diag}\left(\left\{\omega_t \tfrac{\nu-2}{\nu} h_t(\alpha,\beta)\right\}_{t=1}^{T}\right),$$

where $h_t(\alpha,\beta) \doteq \alpha_0 + \alpha_1 y_{t-1}^2 + \beta h_{t-1}(\alpha,\beta)$, we can express the likelihood of (ψ,ϖ) as

$$\mathcal{L}(\psi,\varpi \,|\, y) \propto (\det\Sigma)^{-1/2} \exp\left[-\tfrac{1}{2} y'\Sigma^{-1} y\right]. \quad (2)$$

The Bayesian approach considers (ψ,ϖ) as a random variable which is characterized by a prior density denoted by $p(\psi,\varpi)$. The prior is specified with the help of parameters called hyperparameters which are initially assumed to be known and constant. Moreover, depending on the researcher's prior information, this density can be more or less informative. Then, by coupling the likelihood function of the model parameters with the prior density, we can transform the probability density using Bayes' rule to get the posterior density $p(\psi,\varpi \mid y)$ as follows:

$$p(\psi,\varpi \mid y) = \frac{\mathcal{L}(\psi,\varpi \mid y)\, p(\psi,\varpi)}{\int \mathcal{L}(\psi,\varpi \mid y)\, p(\psi,\varpi)\, d\psi d\varpi}. \quad (3)$$

This posterior is a quantitative, probabilistic description of the knowledge about the model parameters after observing the data. For an excellent introduction on Bayesian econometrics we refer the reader to Koop (2003).

We use truncated normal priors on the GARCH parameters α and β

$$p(\alpha) \propto \phi_{\mathcal{N}_2}(\alpha \mid \mu_\alpha, \Sigma_\alpha)\, 1\left\{\alpha \in \mathrm{R}_+^2\right\}$$
$$p(\beta) \propto \phi_{\mathcal{N}_1}(\beta \mid \mu_\beta, \Sigma_\beta)\, 1\{\beta \in \mathrm{R}_+\},$$

where $\mu_\bullet$ and $\Sigma_\bullet$ are the hyperparameters, $1\{\cdot\}$ is the indicator function and $\phi_{\mathcal{N}_d}$ is the d-dimensional normal density.

The prior distribution of vector ϖ conditional on ν is found by noting that the components ϖ_t are independent and identically distributed from the inverted gamma density, which yields

$$p(\varpi \mid \nu) = \left(\frac{\nu}{2}\right)^{\frac{T\nu}{2}} \left[\Gamma\left(\frac{\nu}{2}\right)\right]^{-T} \left(\prod_{t=1}^{T} \varpi_t\right)^{-\frac{\nu}{2}-1} \times \exp\left[-\frac{1}{2}\sum_{t=1}^{T}\frac{\nu}{\varpi_t}\right].$$

We follow Deschamps (2006) in the choice of the prior distribution on the degrees of freedom parameter. The distribution is a translated exponential with parameters $\lambda > 0$ and $\delta \geq 2$

$$p(\nu) = \lambda \exp\left[-\lambda(\nu-\delta)\right] 1\{\nu > \delta\}.$$

For large values of λ, the mass of the prior is concentrated in the neighborhood of δ and a constraint on the degrees of freedom can be imposed in this manner. Normality of the errors is assumed when δ is chosen large. As pointed out by Deschamps (2006), this prior density is useful for two reasons. First, it is potentially important, for numerical reasons, to bound the degrees of freedom parameter away from two to avoid explosion of the conditional variance. Second, we can approximate the normality of the errors while maintaining a reasonably tight prior which can improve the convergence of the sampler.

The joint prior distribution is then formed by assuming prior independence between the parameters, i.e. $p(\psi,\varpi) = p(\alpha)p(\beta)p(\varpi \mid \nu)p(\nu)$.

The recursive nature of the GARCH(1,1) variance equation implies that the joint posterior and the full conditional densities cannot be expressed in closed form. There exists no (conjugate) prior that can remedy this property. Therefore, we cannot use the simple Gibbs sampler and need to rely on a more elaborated Markov Chain Monte Carlo (MCMC) simulation strategy to approximate the posterior density. The idea of MCMC sampling was first introduced by Metropolis et al. (1953) and was subsequently generalized by Hastings (1970). The sampling strategy relies on the construction of a Markov chain with realizations $(\psi^{[0]},\varpi^{[0]}),\ldots,(\psi^{[j]},\varpi^{[j]}),\ldots$ in the parameter space. Under appropriate regularity conditions, asymptotic results guarantee that as j tends to infinity, $(\psi^{[j]},\varpi^{[j]})$ tends in distribution to a random variable whose density is (3). Hence, after discarding a *burn-in* of the first draws, the realized values of the chain can be used to make inference about the joint posterior.

The MCMC sampler implemented in the package **bayesGARCH** is based on the approach of Ardia (2008, chapter 5), inspired from the previous work by Nakatsuma (1998). The algorithm consists of a MH algorithm where the GARCH parameters are updated by blocks (one block for α and one block for β) while the degrees of freedom parameter is sampled using an optimized rejection technique from a translated exponential source density. This methodology has the advantage of being fully automatic. Moreover, in our experience, the algorithm explores the domain of the joint posterior efficiently compared to naive MH approaches or the Griddy-Gibbs sampler of Ritter and Tanner (1992).

Illustration

We apply our Bayesian estimation methods to daily observations of the Deutschmark vs British Pound (DEM/GBP) foreign exchange log-returns. The sample period is from January 3, 1985, to December 31, 1991, for a total of 1974 observations. This data set has been promoted as an informal benchmark for GARCH time series software validation. From this time series, the first 750 observations are used to illustrate the Bayesian approach. The observation window excerpt from our data set is plotted in Figure 1.

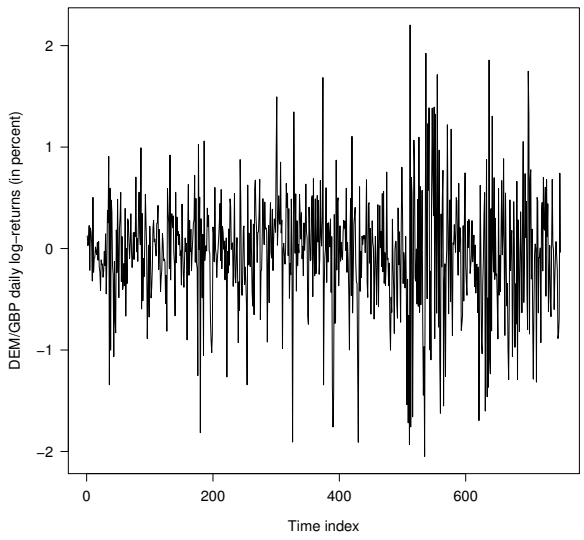


Figure 1: DEM/GBP foreign exchange daily log-returns.

We fit the GARCH(1,1) model with Student-t innovations to the data for this observation window using the `bayesGARCH` function

```
> args(bayesGARCH)

function (y, mu.alpha = c(0, 0),
          Sigma.alpha = 1000 * diag(1,2),
          mu.beta = 0, Sigma.beta = 1000,
          lambda = 0.01, delta = 2,
          control = list())
```

The input arguments of the function are the vector of data, the hyperparameters and the list `control` which can supply any of the following elements:

- `n.chain`: number of MCMC chain(s) to be generated; default `1`.
- `l.chain`: length of each MCMC chain; default `10000`.
- `start.val`: vector of starting values of the chain(s); default `c(0.01,0.1,0.7,20)`. Alternatively, the starting values could be set to the maximum likelihood estimates using the function `fGarch` available in the package **fGarch**, for instance.
- `addPriorConditions`: function which allows the user to add any constraint on the model parameters; default `NULL`, i.e. not additional constraints are imposed.
- `refresh`: frequency of reports; default `10`.
- `digits`: number of printed digits in the reports; default `4`.

As a prior distribution for the Bayesian estimation we take the default values in `bayesGARCH`, which are diffuse priors. We generate two chains for 5000 passes each by setting the `control` parameter values `n.chain = 2` and `l.chain = 5000`.

```
> data(dem2gbp)
> y <- dem2gbp[1:750]
> set.seed(1234)
> MCMC <- bayesGARCH(y, control = list(
          l.chain = 5000, n.chain = 2))

chain:  1  iteration:  10
parameters:  0.0441 0.212 0.656 115
chain:  1  iteration:  20
parameters:  0.0346 0.136 0.747 136
...
chain:  2  iteration:  5000
parameters:  0.0288 0.190 0.754 4.67
```

The function outputs the MCMC chains as an object of the class `"mcmc"` from the package **coda** (Plummer et al., 2010). This package contains functions for post-processing the MCMC output; see Plummer et al. (2006) for an introduction. Note that **coda** is loaded automatically with **bayesGARCH**.

A trace plot of the MCMC chains (i.e. a plot of iterations vs. sampled values) can be generated using the function `traceplot`; the output is displayed in Figure 2.

Convergence of the sampler (using the diagnostic test of Gelman and Rubin (1992)), acceptance rates and autocorrelations in the chains can be computed as follows:

```
> gelman.diag(MCMC)

       Point est. 97.5% quantile
alpha0       1.02           1.07
alpha1       1.01           1.05
beta         1.02           1.07
nu           1.02           1.06

Multivariate psrf

1.02

> 1 - rejectionRate(MCMC)

alpha0 alpha1   beta     nu
 0.890  0.890  0.953  1.000

> autocorr.diag(MCMC)

       alpha0 alpha1  beta    nu
Lag 0   1.000  1.000 1.000 1.000
Lag 1   0.914  0.872 0.975 0.984
Lag 5   0.786  0.719 0.901 0.925
Lag 10  0.708  0.644 0.816 0.863
Lag 50  0.304  0.299 0.333 0.558
```

The convergence diagnostic shows no evidence against convergence for the last 2500 iterations (only

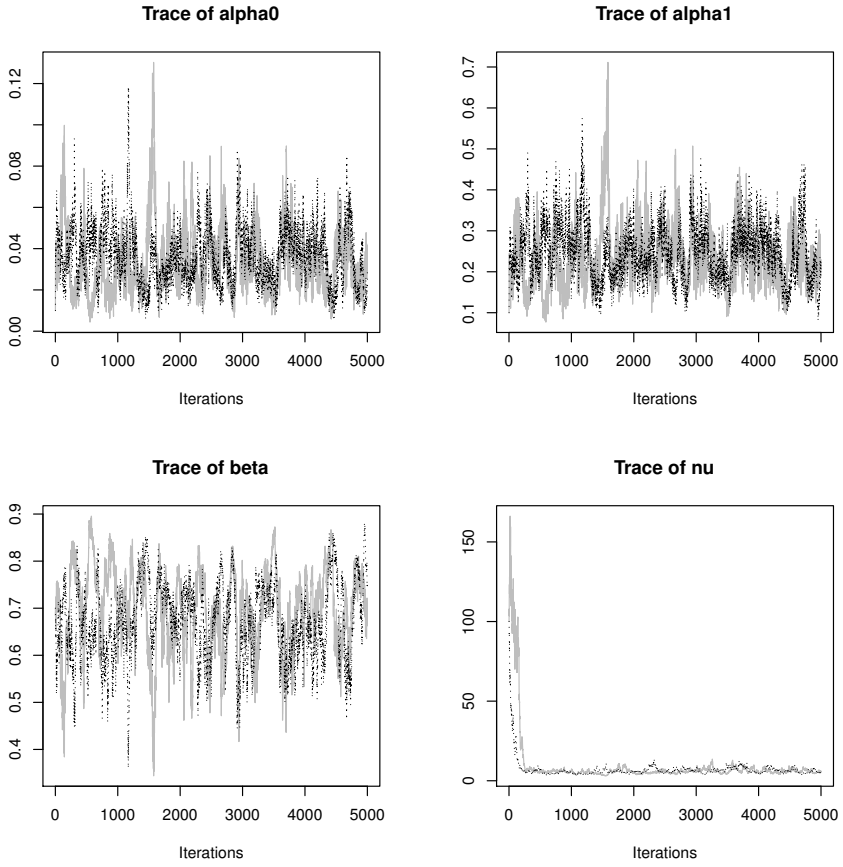


Figure 2: Trace plot of the two MCMC chains (in black and gray) for the four model parameters generated by the MH algorithm.

the second half of the chain is used by default in `gelman.diag`) since the scale reduction factor is smaller than 1.2; see Gelman and Rubin (1992) for details. The MCMC sampling algorithm reaches very high acceptance rates ranging from 89% for vector α to 95% for β suggesting that the proposal distributions are close to the full conditionals. The rejection technique used to generate ν allows a new value to be drawn at each pass in the MH algorithm.

The one-lag autocorrelations in the chains range from 0.87 for parameter α_1 to 0.98 for parameter ν. Using the function `formSmpl`, we discard the first 2500 draws from the overall MCMC output as a burn in period, keep only every second draw to diminish the autocorrelation, and merge the two chains to get a final sample length of 2500.

```
> smpl <- formSmpl(MCMC, l.bi = 2500,
                   batch.size = 2)

n.chain   : 2
l.chain   : 5000
l.bi      : 2500
batch.size: 2
smpl size : 2500
```

Basic posterior statistics can be easily obtained with the `summary` method available for `mcmc` objects.

```
> summary(smpl)

Iterations = 1:2500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 2500

1. Empirical mean and standard deviation
   for each variable, plus standard error
   of the mean:

         Mean     SD Naive SE Time-series SE
alpha0 0.0345 0.0138 0.000277        0.00173
alpha1 0.2360 0.0647 0.001293        0.00760
beta   0.6832 0.0835 0.001671        0.01156
nu     6.4019 1.5166 0.030333        0.19833

2. Quantiles for each variable:

         2.5%   25%    50%    75%   97.5%
alpha0 0.0126 0.024 0.0328 0.0435  0.0646
alpha1 0.1257 0.189 0.2306 0.2764  0.3826
beta   0.5203 0.624 0.6866 0.7459  0.8343
nu     4.2403 5.297 6.1014 7.2282 10.1204
```

The marginal distributions of the model parameters can be obtained by first transforming the output into a matrix and then using the function `hist`.

Marginal posterior densities are displayed in Figure 3. We clearly notice the asymmetric shape of the histograms; this is especially true for parameter ν. This is also reflected by the differences between the posterior means and medians. These results should warn us against the abusive use of asymptotic justifications. In the present case, even 750 observations do not suffice to justify the asymptotic symmetric normal approximation for the parameter estimator's distribution.

Probabilistic statements on nonlinear functions of the model parameters can be straightforwardly obtained by simulation from the joint posterior sample. In particular, we can test the covariance stationarity condition and estimate the density of the unconditional variance when this condition is satisfied. Under the GARCH(1,1) specification, the process is covariance stationary if $\alpha_1 + \beta < 1$, as shown by Bollerslev (1986, page 310). The term $(\alpha_1 + \beta)$ is the degree of persistence in the autocorrelation of the squares which controls the intensity of the clustering in the variance process. With a value close to one, past shocks and past variances will have a longer impact on the future conditional variance.

To make inference on the persistence of the squared process, we simply use the posterior sample and generate $(\alpha_1^{[j]} + \beta^{[j]})$ for each draw $\psi^{[j]}$ in the posterior sample. The posterior density of the persistence is plotted in Figure 4. The histogram is left-skewed with a median value of 0.923 and a maximum value of 1.050. In this case, the covariance stationarity of the process is supported by the data. The unconditional variance of the GARCH(1,1) model is $\alpha_0/(1-\alpha_1-\beta)$ given that $\alpha_1 + \beta < 1$. Conditionally upon existence, the posterior mean is 0.387 and the 90% credible interval is [0.274,1.378]. The empirical variance is 0.323.

Other probabilistic statements on interesting functions of the model parameters can be obtained using the joint posterior sample. Under specification (1), the conditional kurtosis is $3(\nu-2)/(\nu-4)$ provided that $\nu > 4$. Using the posterior sample, we estimate the posterior probability of existence for the conditional kurtosis to be 0.994. Therefore, the existence is clearly supported by the data. Conditionally upon existence, the posterior mean of the kurtosis is 8.21, the median is 5.84 and the 95% confidence interval is [4.12,15.81], indicating heavier tails than for the normal distribution. The positive skewness of the posterior for the conditional kurtosis is caused by a couple of very large values (the maximum simulated value is 404.90). These correspond to draws with ν slightly larger than 4. Note that if one desires to rule out large values for the conditional kurtosis beforehand, then one can set $\delta > 4$ in the prior for ν. For example, the choice $\delta = 4.5$ would guarantee the kurtosis to be smaller than 15.

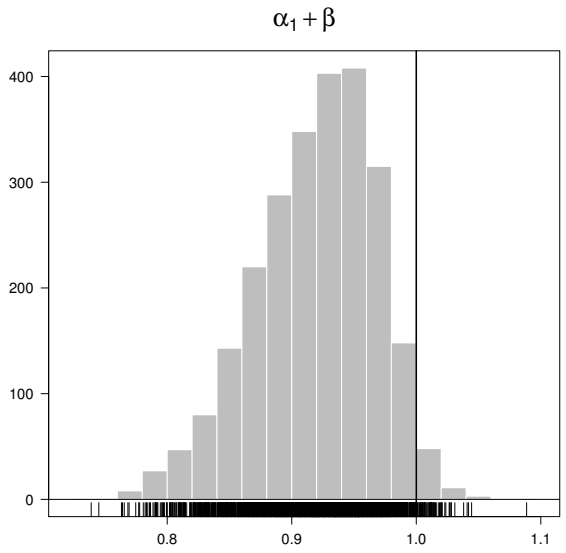


Figure 4: Posterior density of the persistence. The histogram is based on 2500 draws from the joint posterior distribution.

Prior restrictions and normal innovations

The control parameter `addPriorConditions` can be used to impose any type of constraints on the model parameters ψ during the estimation. For instance, to ensure the estimation of a covariance stationary GARCH(1,1) model, the function should be defined as

```
> addPriorConditions <- function(psi)
+   psi[2] + psi[3] < 1
```

Finally, we can impose normality of the innovations in a straightforward manner by setting the hyperparameters $\lambda = 100$ and $\delta = 500$ in the `bayesGARCH` function.

Practical advice

The estimation strategy implemented in **bayesGARCH** is fully automatic and does not require any tuning of the MCMC sampler. This is certainly an appealing feature for practitioners. The generation of the Markov chains is however time consuming and estimating the model over several datasets on a daily basis can therefore take a significant amount of time. In this case, the algorithm can be easily parallelized, by running a single chain on several processors. This can be easily achieved with the package **foreach** (REvolution Computing, 2010), for instance. Also, when the estimation is repeated over updated time series (i.e. time series with more recent observations), it is wise to start the algorithm using the posterior mean or median of the parameters obtained at the previous estimation step. The

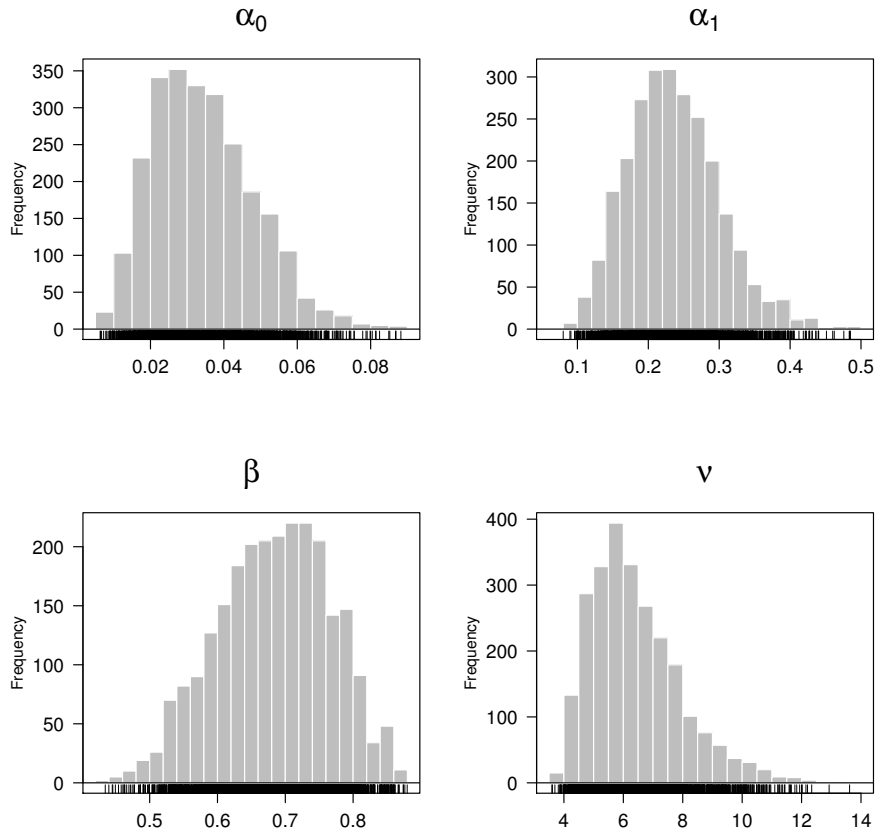


Figure 3: Marginal posterior distributions of the model parameters. This histograms are based on 2500 draws from the joint posterior sample.

impact of the starting values (burn-in phase) is likely to be smaller and thus the convergence faster.

Finally, note that as any MH algorithm, the sampler can get stuck at a given value, so that the chain does not move anymore. However, the sampler uses Taylor-made candidate densities that are especially constructed at each step, so it is almost impossible for this MCMC sampler to get stuck at a given value for many subsequent draws. For example, for our data set we still obtain posterior results that are almost equal to the results that we obtained for the reasonable default initial values `c(0.01,0.1,0.7,20)`, even if we take the very poor initial values `c(0.1,0.01,0.4,50)`. In the unlikely case that such ill behaviour does occur, one could scale the data (to have standard deviation 1), or run the algorithm with different initial values or a different random seed.

Summary

This note presented the Bayesian estimation of the GARCH(1,1) model with Student-*t* innovations using the R package **bayesGARCH**. We illustrated the use of the package with an empirical application to foreign exchange rate log-returns.

Acknowledgements

The authors acknowledge two anonymous reviewers and the associate editor, Martyn Plummer, for helpful comments that have led to improvements of this note. David Ardia is grateful to the Swiss National Science Foundation (under grant #FN PB FR1-121441) for financial support. Any remaining errors or shortcomings are the authors' responsibility.

Bibliography

D. Ardia. *Financial Risk Management with Bayesian Estimation of GARCH Models: Theory and Applications*, volume 612 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, Germany, June 2008. ISBN 978-3-540-78656-6. URL http://www.springer.com/economics/econometrics/book/978-3-540-78656-6.

D. Ardia. *bayesGARCH: Bayesian Estimation of the GARCH(1,1) Model with Student-t Innovations in R*,

2007. URL http://CRAN.R-project.org/package=bayesGARCH. R package version 1.00-05.

T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3): 307–327, Apr. 1986.

P. J. Deschamps. A flexible prior distribution for Markov switching autoregressions with Student-t errors. *Journal of Econometrics*, 133(1):153–190, July 2006.

R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica*, 50(4):987–1008, July 1982.

A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, Nov. 1992.

J. F. Geweke. Getting it right: Joint distribution tests of posterior simulators. *Journal of the American Statistical Association*, 99(467):799–804, Sept. 2004.

J. F. Geweke. Bayesian treatment of the independent Student-t linear model. *Journal of Applied Econometrics*, 8(S1):S19–S40, Dec. 1993.

A. Ghalanos. *rgarch: Flexible GARCH modelling in R*, 2010. URL http://r-forge.r-project.org/projects/rgarch.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, Apr. 1970.

G. Koop. *Bayesian Econometrics*. Wiley-Interscience, London, UK, 2003. ISBN 0470845678.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

T. Nakatsuma. A Markov-chain sampling algorithm for GARCH models. *Studies in Nonlinear Dynamics and Econometrics*, 3(2):107–117, July 1998. URL http://www.bepress.com/snde/vol3/iss2/algorithm1/. Algorithm nr.1.

M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: Convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11, Mar. 2006.

M. Plummer, N. Best, K. Cowles, and K. Vines. *coda: Output analysis and diagnostics for MCMC*, 2010. URL http://CRAN.R-project.org/package=coda. R package version 0.13-5.

REvolution Computing. *foreach: Foreach looping construct for R*, 2009. URL http://CRAN.R-project.org/package=foreach.

C. Ritter and M. A. Tanner. Facilitating the Gibbs sampler: The Gibbs stopper and the Griddy-Gibbs sampler. *Journal of the American Statistical Association*, 87(419):861–868, Sept. 1992.

A. Trapletti and K. Hornik. *tseries: Time series analysis and computational finance*, 2009. URL http://CRAN.R-project.org/package=tseries.

D. Wuertz and Y. Chalabi. *fGarch: Rmetrics - Autoregressive Conditional Heteroskedastic Modelling*, 2009. URL http://CRAN.R-project.org/package=fGarch.

David Ardia
University of Fribourg, Switzerland
david.ardia@unifr.ch

Lennart F. Hoogerheide
Erasmus University Rotterdam, The Netherlands

cudaBayesreg: Bayesian Computation in CUDA

by Adelino Ferreira da Silva

Abstract Graphical processing units are rapidly gaining maturity as powerful general parallel computing devices. The package **cudaBayesreg** uses GPU–oriented procedures to improve the performance of Bayesian computations. The paper motivates the need for devising high-performance computing strategies in the context of fMRI data analysis. Some features of the package for Bayesian analysis of brain fMRI data are illustrated. Comparative computing performance figures between sequential and parallel implementations are presented as well.

A functional magnetic resonance imaging (fMRI) data set consists of time series of volume data in 4D space. Typically, volumes are collected as slices of 64 x 64 voxels. The most commonly used functional imaging technique relies on the blood oxygenation level dependent (BOLD) phenomenon (Sardy, 2007). By analyzing the information provided by the BOLD signals in 4D space, it is possible to make inferences about activation patterns in the human brain. The statistical analysis of fMRI experiments usually involve the formation and assessment of a statistic image, commonly referred to as a Statistical Parametric Map (SPM). The SPM summarizes a statistic indicating evidence of the underlying neuronal activations for a particular task. The most common approach to SPM computation involves a univariate analysis of the time series associated with each voxel. Univariate analysis techniques can be described within the framework of the general linear model (GLM) (Sardy, 2007). The GLM procedure used in fMRI data analysis is often said to be "massively univariate", since data for each voxel are independently fitted with the same model. Bayesian methodologies provide enhanced estimation accuracy (Friston et al., 2002). However, since (non-variational) Bayesian models draw on Markov Chain Monte Carlo (MCMC) simulations, Bayesian estimates involve a heavy computational burden.

The programmable Graphic Processor Unit (GPU) has evolved into a highly parallel processor with tremendous computational power and very high memory bandwidth (NVIDIA Corporation, 2010b). Modern GPUs are built around a scalable array of multithreaded streaming multiprocessors (SMs). Current GPU implementations enable scheduling thousands of concurrently executing threads. The *Compute Unified Device Architecture* (CUDA) (NVIDIA Corporation, 2010b) is a software platform for massively parallel high-performance computing on NVIDIA manycore GPUs. The CUDA programming model follows the standard single-program multiple-data (SPMD) model. CUDA greatly simplifies the task of parallel programming by providing thread management tools that work as extensions of conventional C/C++ constructions. Automatic thread management removes the burden of handling the scheduling of thousands of lightweight threads, and enables straightforward programming of the GPU cores.

The package **cudaBayesreg** (Ferreira da Silva, 2010a) implements a Bayesian multilevel model for the analysis of brain fMRI data in the CUDA environment. The statistical framework in **cudaBayesreg** is built around a Gibbs sampler for multilevel/hierarchical linear models with a normal prior (Ferreira da Silva, 2010c). Multilevel modeling may be regarded as a generalization of regression methods in which regression coefficients are themselves given a model with parameters estimated from data (Gelman, 2006). As in SPM, the Bayesian model fits a linear regression model at each voxel, but uses uses multivariate statistics for parameter estimation at each iteration of the MCMC simulation. The Bayesian model used in **cudaBayesreg** follows a two–stage Bayes prior approach to relate voxel regression equations through correlations between the regression coefficient vectors (Ferreira da Silva, 2010c). This model closely follows the Bayesian multilevel model proposed by Rossi, Allenby and McCulloch (Rossi et al., 2005), and implemented in **bayesm** (Rossi and McCulloch., 2008). This approach overcomes several limitations of the classical SPM methodology. The SPM methodology traditionally used in fMRI has several important limitations, mainly because it relies on classical hypothesis tests and p–values to make statistical inferences in neuroimaging (Friston et al., 2002; Berger and Sellke, 1987; Vul et al., 2009). However, as is often the case with MCMC simulations, the implementation of this Bayesian model in a sequential computer entails significant time complexity. The CUDA implementation of the Bayesian model proposed here has been able to reduce significantly the runtime processing of the MCMC simulations. The main contribution for the increased performance comes from the use of separate threads for fitting the linear regression model at each voxel in parallel.

Bayesian multilevel modeling

We are interested in the following Bayesian multilevel model, which has been analyzed by Rossi

et al. (2005), and has been implemented as `rhierLinearModel` in **bayesm**. Start out with a general linear model, and fit a set of m voxels as,

$$y_i = X_i\beta_i + \epsilon_i, \quad \epsilon_i \overset{iid}{\sim} N\left(0, \sigma_i^2 I_{n_i}\right), \quad i = 1, \ldots, m. \tag{1}$$

In order to tie together the voxels' regression equations, assume that the $\{\beta_i\}$ have a common prior distribution. To build the Bayesian regression model we need to specify a prior on the $\{\beta_i\}$ coefficients, and a prior on the regression error variances $\{\sigma_i^2\}$. Following Ferreira da Silva (2010c), specify a normal regression prior with mean $\Delta' z_i$ for each β,

$$\beta_i = \Delta' z_i + \nu_i, \quad \nu_i \overset{iid}{\sim} N\left(0, V_\beta\right), \tag{2}$$

where z is a vector of n_z elements, representing characteristics of each of the m regression equations.

The prior (2) can be written using the matrix form of the multivariate regression model for k regression coefficients,

$$B = Z\Delta + V \tag{3}$$

where B and V are $m \times k$ matrices, Z is a $m \times n_z$ matrix, Δ is a $n_z \times k$ matrix. Interestingly, the prior (3) assumes the form of a second–stage regression, where each column of Δ has coefficients which describes how the mean of the k regression coefficients varies as a function of the variables in z. In (3), Z assumes the role of a prior design matrix.

The proposed Bayesian model can be written down as a sequence of conditional distributions (Ferreira da Silva, 2010c),

$$\begin{aligned} y_i &\mid X_i, \beta_i, \sigma_i^2 \\ \beta_i &\mid z_i, \Delta, V_\beta \\ \sigma_i^2 &\mid \nu_i, s_i^2 \\ V_\beta &\mid \nu, V \\ \Delta &\mid V_\beta, \bar{\Delta}, A. \end{aligned} \tag{4}$$

Running MCMC simulations on the set of full conditional posterior distributions (4), the full posterior for all the parameters of interest may then be derived.

GPU computation

In this section, we describe some of the main design considerations underlying the code implementation in **cudaBayesreg**, and the options taken for processing fMRI data in parallel. Ideally, the GPU is best suited for computations that can be run on numerous data elements simultaneously in parallel (NVIDIA Corporation, 2010b). Typical textbook applications for the GPU involve arithmetic on large matrices, where the same operation is performed across thousands of elements at the same time. Since the Bayesian model of computation outlined in the previous Section does not fit well in this framework, some design options had to be assumed in order to properly balance optimization, memory constraints, and implementation complexity, while maintaining numerical correctness. Some design requirements for good performance on CUDA are as follows (NVIDIA Corporation, 2010a): (i) the software should use a large number of threads; (ii) different execution paths within the same thread block (warp) should be avoided; (iii) inter-thread communication should be minimized; (iv) data should be kept on the device as long as possible; (v) global memory accesses should be coalesced whenever possible; (vi) the use of shared memory should be preferred to the use of global memory. We detail below how well these requirements have been met in the code implementation. The first requirement is easily met by **cudaBayesreg**. On the one hand, fMRI applications typically deal with thousands of voxels. On the other hand, the package uses three constants which can be modified to suit the available device memory, and the computational power of the GPU. Specifically, `REGDIM` specifies the maximum number of regressions (voxels), `OBSDIM` specifies the maximum length of the time series observations, and `XDIM` specifies the maximum number of regression coefficients. Requirements (ii) and (iii) are satisfied by **cudaBayesreg** as well. Each thread executes the same code independently, and no inter-thread communication is required. Requirement (iv) is optimized in **cudaBayesreg** by using as much constant memory as permitted by the GPU. Data that do not change between MCMC iterations are kept in constant memory. Thus, we reduce expensive memory data transfers between host and device. For instance, the matrix of voxel predictors X (see (1)) is kept in constant memory. Requirement (v) is insufficiently met in **cudaBayesreg**. For maximum performance, memory accesses to global memory must be coalesced. However, different fMRI data sets and parameterizations may generate data structures with highly variable dimensions, thus rendering coalescence difficult to implement in a robust manner. Moreover, GPU devices of *Compute Capability* 1.x impose hard memory coalescing restrictions. Fortunately, GPU devices of *Compute Capability* 2.x are expected to lift some of the most taxing memory coalescing constraints. Finally requirement (vi) is not met by the available code. The current kernel implementation does not use shared memory. The relative complexity of the Bayesian computation performed by each thread compared to the conventional arithmetic load assigned to the thread has prevented us from exploiting shared memory operations. The task assigned to the kernel may be subdivided to reduce kernel complexity. However, this option may easily compromise other optimization requirements, namely thread independence. As detailed in the next paragraph, our option has been to keep the computational design simple, by assigning the whole of the

univariate regression to the kernel.

The computational model has been specified as a grid of thread blocks of dimension 64 in which a separate thread is used for fitting a linear regression model at each voxel in parallel. Maximum efficiency is expected to be achieved when the total number of required threads to execute in parallel equals the number of voxels in the fMRI data set, after appropriate masking has been done. However, this approach typically calls for the parallel execution of several thousands of threads. To keep computational resources low, while maintaining significant high efficiency it is generally preferable to process fMRI data slice-by-slice. In this approach, slices are processed in sequence. Voxels in slices are processed in parallel. Thus, for slices of dimension 64×64, the required number of parallel executing threads does not exceed 4096 at a time. The main computational bottleneck in sequential code comes from the necessity of performing Gibbs sampling using a (temporal) univariate regression model for all voxel time series. We coded this part of the MCMC computation as device code, i.e. a kernel to be executed by the CUDA threads. CUDA threads execute on the GPU device that operates as a coprocessor to the host running the MCMC simulation. Following the proposed Bayesian model (4), each thread implements a Gibbs sampler to draw from the posterior of a univariate regression with a conditionally conjugate prior. The host code is responsible for controlling the MCMC simulation. At each iteration, the threads perform one Gibbs iteration for all voxels in parallel, to draw the threads' estimators for the regression coefficients β_i as specified in (4). In turn, the host, based on the simulated β_i values, draws from the posterior of a multivariate regression model to estimate V_β and Δ. These values are then used to drive the next iteration.

The bulk of the MCMC simulations for Bayesian data analysis is implemented in the kernel (device code). Most currently available RNGs for the GPU tend to be have too high time- and space-complexity for our purposes. Therefore, we implemented and tested three different random number generators (RNGs) in device code, by porting three well-known RNGs to device code. Marsaglia's multicarry RNG (Marsaglia, 2003) follows the R implementation, is the fastest one, and is used by default; Brent's RNG (Brent, 2007) has higher quality but is not-so-fast; Matsumoto's Mersenne Twister (Matsumoto and Nishimura, 1998) is slower than the others. In addition, we had to ensure that different threads receive different random seeds. We generated random seeds for the threads by combining random seeds generated by the host with the threads' unique identification numbers. Random deviates from the normal (Gaussian) distribution and chi-squared distribution had to be implemented in device code as well. Random deviates from the normal distribution were generated using the Box-Muller method. In a similar vein, random deviates from the chi-squared distribution with ν number of degrees of freedom, $\chi^2(\nu)$, were generated from gamma deviates, $\Gamma(\nu/2, 1/2)$, following the method of Marsaglia and Tsang specified in (Press et al., 2007).

The next Sections provide details on how to use **cudaBayesreg** (Ferreira da Silva, 2010b) for fMRI data analysis. Two data sets, which are included and documented in the complementary package **cudaBayesregData** (Ferreira da Silva, 2010b), have been used in testing: the 'fmri' and the 'swrfM' data sets. We performed MCMC simulations on these data sets using three types of code implementations for the Bayesian multilevel model specified before: a (sequential) R-language version, a (sequential) C-language version, and a CUDA implementation. Comparative runtimes for 3000 iterations in these three situations, for the data sets 'fmri' and 'swrfM', are as follows.

```
Runtimes in  seconds for 3000 iterations:


      slice R-code C-code CUDA
fmri      3   1327    224   22
swrfM    21   2534    309   41
```

Speed-up factors between the sequential versions and the parallel CUDA implementation are summarized next.

```
Comparative speedup factors:


      C-vs-R CUDA-vs-C CUDA-vs-R
fmri     6.0      10.0      60.0
swrfM    8.2       7.5      61.8
```

In these tests, the C-implementation provided, approximately, a $7.6\times$ mean speedup factor relative to the equivalent R implementation. The CUDA implementation provided a $8.8\times$ mean speedup factor relative to the equivalent C implementation. Overall, the CUDA implementation yielded a significant $60\times$ speedup factor. The tests were performed on a notebook equipped with a (low–end) graphics card: a 'GeForce 8400M GS' NVIDIA device. This GPU device has just 2 multiprocessors, *Compute Capability* 1.1, and delivers single–precision performance. The compiler flags used in compiling the source code are detailed in the package's `Makefile`. In particular, the optimization flag `-O3` is set there. It is worth noting that the CUDA implementation in **cudaBayesreg** affords much higher speedups. First, the CUDA implementation may easily be modified to process all voxels of a fMRI volume in parallel, instead of processing data in a slice-by-slice basis. Second, GPUs with 16 multiprocessors and 512 CUDA cores and *Compute Capability* 2.0 are now available.

Experiments using the fmri test dataset

The data set 'fmri.nii.gz' is available from the FMRIB/FSL site (www.fmrib.ox.ac.uk/fsl). This data set is from an auditory–visual experiment. Auditory stimulation was applied as an alternating "boxcar" with 45*s*-on-45*s*-off and visual stimulation was applied as an alternating "boxcar" with 30*s*-on-30*s*-off. The data set includes just 45 time points and 5 slices from the original 4D data. The file 'fmri_filtered_func_data.nii' included in **cudaBayesregData** was obtained from 'fmri.nii.gz' by applying FSL/FEAT pre-preprocessing tools. For input/output of NIFTI formatted fMRI data sets **cudaBayesreg** depends on the **R** package **oro.nifti** (Whitcher et al., 2010). The following code runs the MCMC simulation for slice 3 of the `fmri` dataset, and saves the result.

```
> require("cudaBayesreg")
> slicedata <- read.fmrislice(fbase = "fmri",
+     slice = 3, swap = TRUE)
> ymaskdata <- premask(slicedata)
> fsave <- "/tmp/simultest1.sav"
> out <- cudaMultireg.slice(slicedata,
+     ymaskdata, R = 3000, keep = 5,
+     nu.e = 3, fsave = fsave, zprior = FALSE)
```

We may extract the posterior probability (PPM) images for the visual (`vreg=2`) and auditory (`vreg=4`) stimulation as follows (see Figures 1 and 2).

```
> post.ppm(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 2,
+     col = heat.colors(256))
```

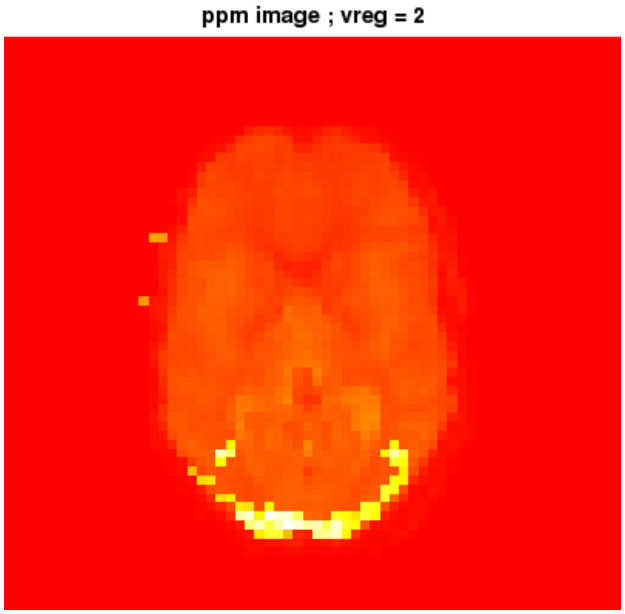


Figure 1: PPM images for the visual stimulation

```
> post.ppm(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 4,
+     col = heat.colors(256))
```

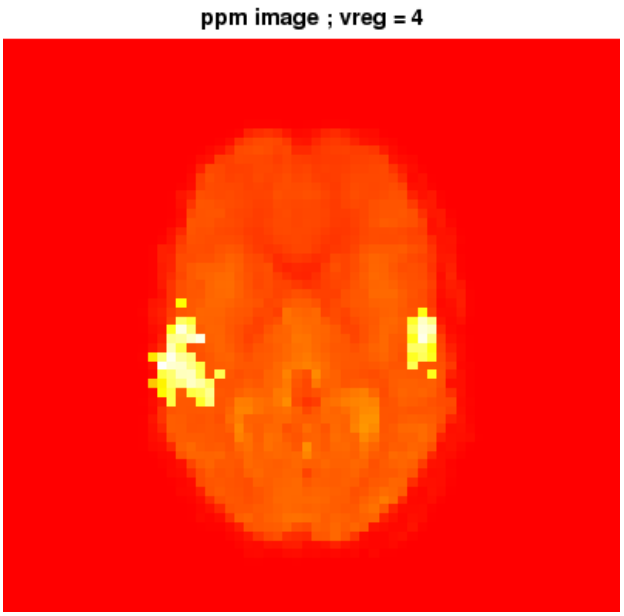


Figure 2: PPM images for the auditory stimulation

To show the fitted time series for a (random) active voxel, as depicted in Figure 3, we use the code:

```
> post.tseries(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 2)

range pm2: -1.409497 1.661774
```

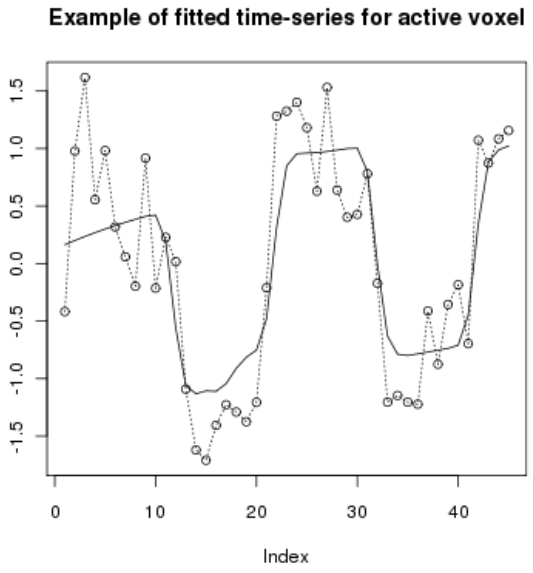


Figure 3: Fitted time–series for an active voxel

Summary statistics for the posterior mean values of regression coefficient `vreg=2`, are presented next. The same function plots the histogram of the posterior distribution for `vreg=2`, as represented in Figure 4.

```
> post.simul.hist(out = out, vreg = 2)
```

```
Call:
        density.default(x = pm2)

Data: pm2 (1525 obs.);        Bandwidth 'bw' = 0.07947

       x                 y
 Min.   :-1.6479   Min.   :0.0000372
 1st Qu.:-0.7609   1st Qu.:0.0324416
 Median : 0.1261   Median :0.1057555
 Mean   : 0.1261   Mean   :0.2815673
 3rd Qu.: 1.0132   3rd Qu.:0.4666134
 Max.   : 1.9002   Max.   :1.0588599
[1] "active range:"
[1] 0.9286707 1.6617739
[1] "non-active range:"
[1] -1.4094965  0.9218208
hpd (95%)=               -0.9300451 0.9286707
```

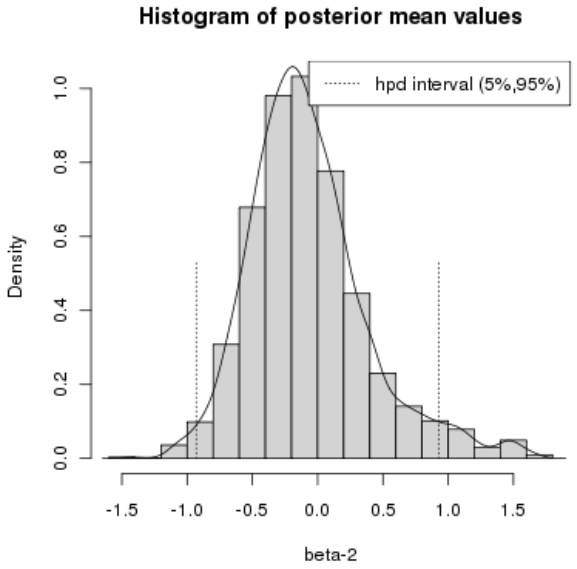


Figure 4: Histogram of the posterior distribution of the regression coefficient β_2 (slice 3).

An important feature of the Bayesian model used in **cudaBayesreg** is the shrinkage induced by the hyperprior ν on the estimated parameters. We may assess the adaptive shrinkage properties of the Bayesian multilevel model for two different values of ν as follows.

```
> nu2 <- 45
> fsave2 <- "/tmp/simultest2.sav"
> out2 <- cudaMultireg.slice(slicedata,
+     ymaskdata, R = 3000, keep = 5,
+     nu.e = nu2, fsave = fsave2,
+     zprior = F)
> vreg <- 2
> x1 <- post.shrinkage.mean(out = out,
+     slicedata$X, vreg = vreg,
+     plot = F)
> x2 <- post.shrinkage.mean(out = out2,
+     slicedata$X, vreg = vreg,
+     plot = F)
> par(mfrow = c(1, 2), mar = c(4,
+     4, 1, 1) + 0.1)
> xlim = range(c(x1$beta, x2$beta))
> ylim = range(c(x1$yrecmean, x2$yrecmean))
> plot(x1$beta, x1$yrecmean, type = "p",
+     pch = "+", col = "violet",
+     ylim = ylim, xlim = xlim,
+     xlab = expression(beta), ylab = "y")
> legend("topright", expression(paste(nu,
+     "=3")), bg = "seashell")
> plot(x2$beta, x2$yrecmean, type = "p",
+     pch = "+", col = "blue", ylim = ylim,
+     xlim = xlim, xlab = expression(beta),
+     ylab = "y")
> legend("topright", expression(paste(nu,
+     "=45")), bg = "seashell")
> par(mfrow = c(1, 1))
```

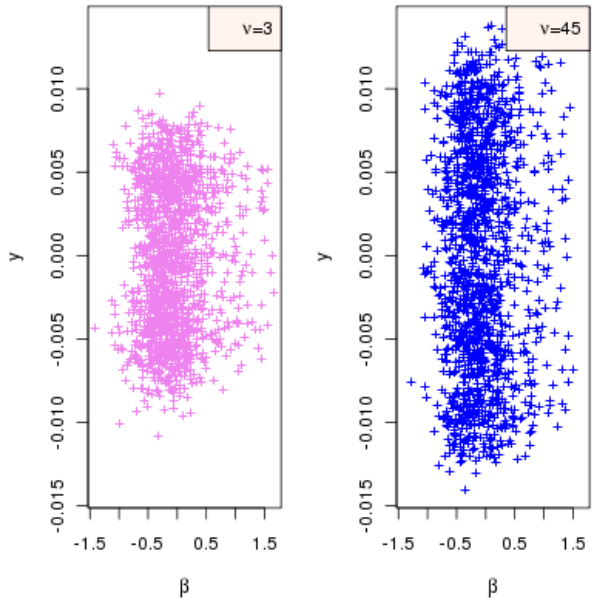


Figure 5: Shrinkage assessment: variability of mean predictive values for $\nu = 3$ and $\nu = 45$.

Experiments using the SPM auditory dataset

In this Section, we exemplify the analysis of the random effects distribution Δ, following the specification of cross-sectional units (group information) in the Z matrix of the statistical model. The Bayesian multilevel statistical model allows for the analysis of random effects through the specification of the Z matrix for the prior in (2). The dataset with prefix `swrfM` (argument `fbase="swrfM"`) in the package's data directory, include mask files associated with the partition of the fMRI dataset 'swrfM' in 3 classes: cerebrospinal fluid (CSF), gray matter (GRY) and white matter (WHT). As before, we begin by loading the data and running the simulation. This time, however, we call `cudaMultireg.slice` with the argument `zprior=TRUE`. This argument will launch `read.Zsegslice`, that reads the segmented images (CSF/GRY/WHT) to build the Z matrix.

```
> fbase <- "swrfM"
> slice <- 21
> slicedata <- read.fmrislice(fbase = fbase,
+     slice = slice, swap = TRUE)
> ymaskdata <- premask(slicedata)
> fsave3 <- "/tmp/simultest3.sav"
> out <- cudaMultireg.slice(slicedata,
+     ymaskdata, R = 3000, keep = 5,
+     nu.e = 3, fsave = fsave3,
+     zprior = TRUE)
```

We confirm that areas of auditory activation have been effectively selected by displaying the PPM image for regression variable `vreg=2`.

```
> post.ppm(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 2,
+     col = heat.colors(256))
```

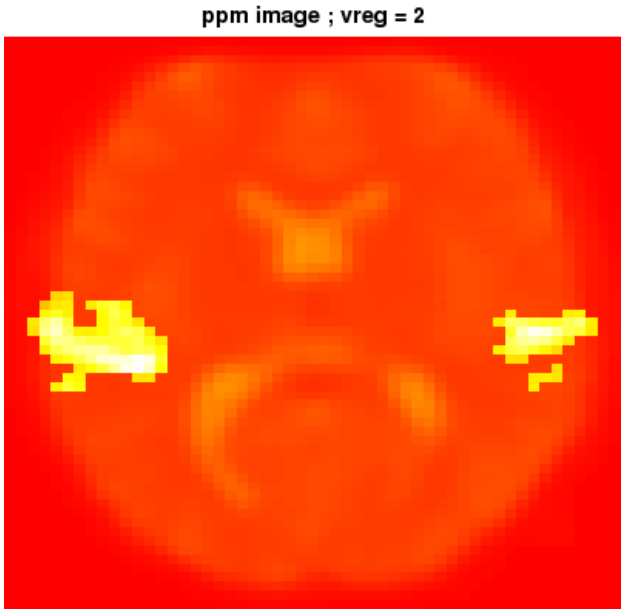


Figure 6: PPM images for the auditory stimulation

Plots of the draws of the mean of the random effects distribution are presented in Figure 7.

```
> post.randeff(out)
```

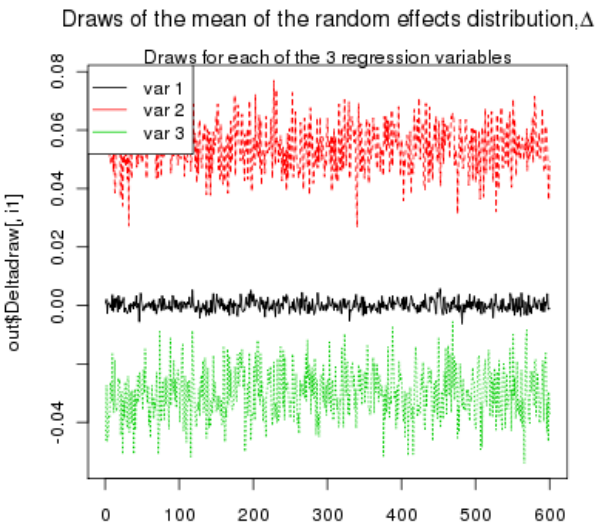


Figure 7: Draws of the mean of the random effects distribution

Random effects plots for each of the 3 classes are obtained by calling,

```
> post.randeff(out, classnames = c("CSF",
+     "GRY", "WHT"))
```

Plots of the random effects associated with the 3 classes are depicted in Figures 8–10.

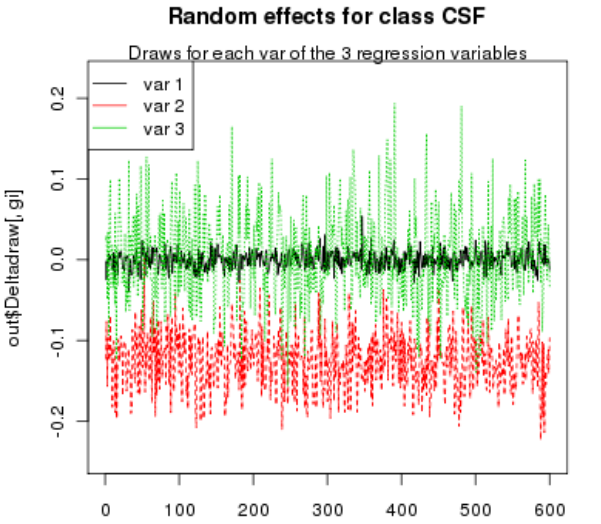


Figure 8: Draws of the random effects distribution for class CSF

Figure 9: Draws of the random effects distribution for class GRY

```
> post.randeff(out, classnames = c("CSF",
+     "GRY", "WHT"))
```

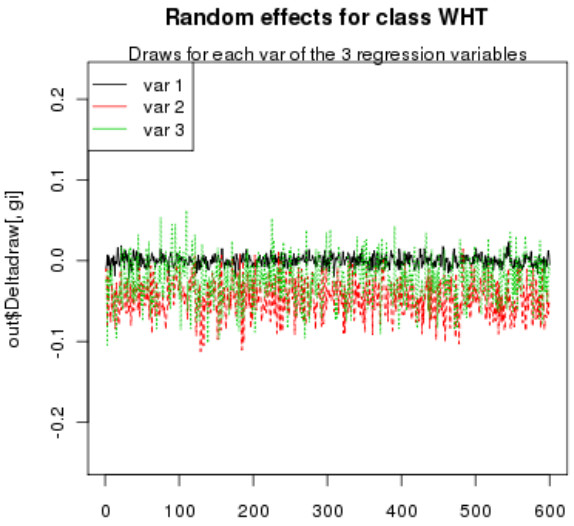


Figure 10: Draws of the random effects distribution for class WHT

Conclusion

The CUDA implementation of the Bayesian model in **cudaBayesreg** has been able to reduce significantly the runtime processing of MCMC simulations. For the applications described in this paper, we have obtained speedup factors of 60× compared to the equivalent sequential R code. The results point out the enormous potential of parallel high-performance computing strategies on manycore GPUs.

Bibliography

J. Berger and T. Sellke. Testing a point null hypothesis: the irreconciability of p values and evidence. *Journal of the American Statistical Association*, 82: 112–122, 1987.

R. P. Brent. Some long-period random number generators using shifts and xors. *ANZIAM Journal*, 48 (CTAC2006):C188–C202, 2007. URL http://wwwmaths.anu.edu.au/~brent.

A. R. Ferreira da Silva. *cudaBayesreg: CUDA Parallel Implementation of a Bayesian Multilevel Model for fMRI Data Analysis*, 2010a. URL http://CRAN.R-project.org/package=cudaBayesreg. R package version 0.3-8.

A. R. Ferreira da Silva. *cudaBayesregData: Data sets for the examples used in the package cudaBayesreg*, 2010b. URL http://CRAN.R-project.org/package=cudaBayesreg. R package version 0.3-8.

A. R. Ferreira da Silva. A Bayesian multilevel model for fMRI data analysis. *Comput. Methods Programs Biomed.*, in press, 2010c.

K. J. Friston, W. Penny, C. Phillips, S. Kiebel, G. Hinton, and J. Ashburner. Classical and Bayesian Inference in Neuroimaging: Theory. *NeuroImage*, 16: 465– 483, 2002.

A. Gelman. Multilevel (hierarchical) modeling: What it can and cannot do. *Technometrics*, 48(3):432–435, Aug. 2006.

G. Marsaglia. Random number generators. *Journal of Modern Applied Statistical Methods*, 2, May 2003.

M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, Jan. 1998.

NVIDIA Corporation. *CUDA C Best Practices Guide Version 3.2*, Aug. 2010a. URL http://www.nvidia.com/CUDA.

NVIDIA Corporation. *NVIDIA CUDA Programming Guide, Version 3.2*, Aug. 2010b. URL http://www.nvidia.com/CUDA.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes, The Art of Scientific Computing*. CUP, third edition, 2007.

P. Rossi and R. McCulloch. *bayesm: Bayesian Inference for Marketing/Micro-econometrics*, 2008. URL http://faculty.chicagogsb.edu/peter.rossi/research/bsm.html. R package version 2.2-2.

P. E. Rossi, G. Allenby, and R. McCulloch. *Bayesian Statistics and Marketing*. John Wiley and Sons, 2005.

G. E. Sardy. *Computing Brain Activity Maps from fMRI Time-Series Images*. Cambridge University Press, 2007.

E. Vul, C. Harris, P. Winkielman, and H. Pashler. Puzzlingly high correlations in fMRI studies of emotion, personality, and social cognition. *Perspectives on Psychological Science*, 4(3):274–290, 2009.

B. Whitcher, V. Schmid, and A. Thornton. *oro.nifti: Rigorous - NIfTI Input / Output*, 2010. URL http://CRAN.R-project.org/package=oro.nifti. R Package Version 0.1.4.

Adelino Ferreira da Silva
Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Portugal
afs@fct.unl.pt

binGroup: A Package for Group Testing

by Christopher R. Bilder, Boan Zhang, Frank Schaarschmidt and Joshua M. Tebbs

Abstract When the prevalence of a disease or of some other binary characteristic is small, group testing (also known as pooled testing) is frequently used to estimate the prevalence and/or to identify individuals as positive or negative. We have developed the **binGroup** package as the first package designed to address the estimation problem in group testing. We present functions to estimate an overall prevalence for a homogeneous population. Also, for this setting, we have functions to aid in the very important choice of the group size. When individuals come from a heterogeneous population, our group testing regression functions can be used to estimate an individual probability of disease positivity by using the group observations only. We illustrate our functions with data from a multiple vector transfer design experiment and a human infectious disease prevalence study.

Introduction

Group testing, where individuals are composited into pools to screen for a binary characteristic, has a long history of successful application in areas such as human infectious disease detection, veterinary screening, drug discovery, and insect vector pathogen transmission (Pilcher et al., 2005; Peck, 2006; Remlinger et al., 2006; Tebbs and Bilder, 2004). Group testing works well in these settings because the prevalence is usually small and individual specimens (e.g., blood, urine, or cattle ear notches) can be composited without loss of diagnostic test accuracy. Group testing is performed often by assigning each individual to a group and testing every group for a positive or negative outcome of the binary characteristic. Using these group responses alone, estimates of overall prevalence or subject-specific probabilities of positivity can be found. When further individual identification of the binary characteristic is of interest, re-testing of specimens within positive groups can be performed to decode the individual positives from the negatives. There are other variants to how group testing is applied, and some will be discussed in this paper. A recent review of group testing for estimation and identification is found in Hughes-Oliver (2006).

Our **binGroup** package (Zhang et al., 2010) is the first dedicated to the group testing estimation problem within homogeneous or heterogeneous populations. We also provide functions to determine the optimal group size based on prior knowledge of what the overall prevalence may be. All of our functions have been written in familiar formats to those where individual testing is used (e.g., `binom.confint()` in **binom** (Dorai-Raj, 2009) or `glm()` in **stats**).

Homogeneous populations

Group testing has been used traditionally in settings where one overall prevalence of a binary characteristic within a homogeneous population is of interest. Typically, one assumes that each individual is independent and has the same probability p of the characteristic, so that p is the overall prevalence. In the next section, we will consider the situation where individuals have different probabilities of positivity. Here, we let θ denote the probability that a group of size s is positive. One can show then $p = 1 - (1-\theta)^{1/s}$ when diagnostic testing is perfect. This equation plays a central role in making estimates and inferences about individuals when only the group responses are known and each individual is within only one group.

We have written two functions to calculate a confidence interval for p. First, the `bgtCI()` function calculates intervals for p when a common group size is used throughout the sample. For example, Ornaghi et al. (1999) estimate the probability that the female *Delphacodes kuscheli* (planthopper) transfers the Mal Rio Cuarto (MRC) virus to maize crops. In stage 4 of the experiment, $n = 24$ enclosed maize plants each had $s = 7$ planthopper vectors placed on them for forty-eight hours and there were $y = 3$ plants that tested positive for the MRC virus after two months. The 95% confidence interval for the probability of transmission p is calculated by

```
> bgtCI(n = 24, y = 3, s = 7,
+   conf.level = 0.95,
+   alternative = "two.sided",
+   method = "Score")

 95 percent Score confidence interval:
  [ 0.006325, 0.05164 ]
 Point estimate: 0.0189
```

where the score (Wilson) interval was used. While the score interval is usually one of the best in terms of coverage (Tebbs and Bilder, 2004), other intervals calculated by `bgtCI()` include the Clopper-Pearson, the asymptotic second-order corrected, and the Wald. The maximum likelihood estimate for p is $1 - (1 - 3/24)^{1/7} = 0.0189$.

Group testing is applied usually with equally sized groups. When a common group size is not used, perhaps due to physical or design constraints, our `bgtvs()` function can calculate the exact interval proposed by Hepworth (1996, equation 5). The arguments to `bgtvs()` include the following vectors: `s`, the different group sizes occurring in the design; `n`,

the corresponding numbers of groups for each group size; and `y`, the corresponding numbers of observed positive groups for each group size. Note that the algorithm becomes computationally expensive when the number of different group sizes is more than three.

One of the most important design considerations is the choice of the group size. Choosing a group size that is too small may result in few groups testing positive, so more tests are used than necessary. Choosing a group size that is too large may result in almost all groups testing positive, which leads to a poor estimate of p. As a rule of thumb, one tries to choose a group size so that about half of the groups test positive. More formally, one can choose an s that minimizes the mean square error (MSE) for a fixed n and a prior estimate of p (Swallow, 1985). If we use a prior prevalence estimate of 0.0189, 24 groups, and a maximum possible group size of 100, our `estDesign()` function finds the optimal choice of s to be 43:

```
> estDesign(n = 24, smax = 100, p.tr = 0.0189)
group size s with minimal mse(p) = 43

$varp [1] 3.239869e-05

$mse [1] 3.2808e-05

$bias [1] 0.0006397784

$exp [1] 0.01953978
```

The function provides the corresponding variance, MSE, bias, and expected value for the maximum likelihood estimator of p. While $s = 43$ is optimal for this example, large group sizes can not necessarily be used in practice (e.g., dilution effects may prevent using a large group size), but this can still be used as a goal.

Our other functions for homogeneous population settings include `bgtTest()`, which calculates a p-value for a hypothesis test involving p. Also, `bgtPower()` calculates the power of the hypothesis test. Corresponding to `bgtPower()`, the `nDesign()` and `sDesign()` functions calculate the power with increasing n or s, respectively, with `plot.bgtDesign()` providing a plot. These functions allow researchers to design their own experiment in a similar manner to that described in Schaarschmidt (2007).

Heterogeneous populations

When covariates for individuals are available, we can model the probability of positivity as with any binary regression model. However, the complicating aspect here is that only the group responses may be available. Also, if both group responses and responses from re-tests are available, the correlation between these responses makes the analysis more difficult. Vansteelandt et al. (2000) and Xie (2001) have both proposed ways to fit these models. Vansteelandt et al. (2000) use a likelihood function written in terms of the initial group responses and maximize it to obtain the maximum likelihood estimates of the model parameters. This fitting procedure can not be used when re-tests are available. Xie (2001) writes the likelihood function in terms of the unobserved individual responses and uses the EM algorithm for estimation. This approach has an advantage over Vansteelandt et al. (2000) because it can be used in more complicated settings such as when re-tests are available or when individuals appear in multiple groups (e.g., matrix or array-based pooling). However, while Xie's fitting procedure is more general, it can be very slow to converge for some group testing protocols.

The `gtreg()` function fits group testing regression models in situations where individuals appear in only one group and no re-tests are performed. The function call is very similar to that of `glm()` in the **stats** package. Additional arguments include sensitivity and specificity of the group test; group numbers for the individuals, and specification of either the Vansteelandt or Xie fitting methods. Both model-fitting methods will produce approximately the same estimates and corresponding standard errors.

We illustrate the `gtreg()` function with data from Vansteelandt et al. (2000). The data were obtained through a HIV surveillance study of pregnant women in rural parts of Kenya. For this example, we model the probability that a women is HIV positive using the covariates age and highest attained education level (treated as ordinal). The data structure is

```
> data(hivsurv)
> tail(hivsurv[,c(3,5,6:8)], n = 7)
    AGE  EDUC HIV gnum groupres
422  29     3   1   85        1
423  17     2   0   85        1
424  18     2   0   85        1
425  18     2   0   85        1
426  22     3   0   86        0
427  30     2   0   86        0
428  34     3   0   86        0
```

Each individual within a group (`gnum` is the group number) is given the same group response (`groupres`) within the data set. For example, individual #422 is positive (1) for HIV, and this leads to all individuals within group #85 to have a positive group response. Note that the individual HIV responses are known here because the purpose of the original study was to show group testing works as well as individual testing (Verstraeten et al., 1998). Continuing, the `gtreg()` function fits the model and the fit is summarized with `summary()`:

```
> fit1 <- gtreg(formula = groupres ~ AGE + EDUC,
+   data = hivsurv, groupn = gnum, sens = 0.99,
+   spec = 0.95, linkf =  "logit",
+   method = "Vansteelandt")
```

```
> summary(fit1)

Call: gtreg(formula = groupres ~ AGE + EDUC,
 data = hivsurv, groupn = gnum, sens = 0.99,
 spec = 0.95, linkf = "logit",
 method = "Vansteelandt")

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1811  -0.9384  -0.8219   1.3299   1.6696

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.99039    1.59911  -1.870   0.0615 .
AGE         -0.05163    0.06748  -0.765   0.4443
EDUC         0.73621    0.43885   1.678   0.0934 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*'
 0.05 '.' 0.1 ' ' 1

    Null deviance: 191.4 on 427 degrees of freedom
Residual deviance: 109.4 on 425 degrees of freedom
AIC: 115.4

Number of iterations in optim(): 138
```

The results from `gtreg()` are stored in `fit1` here, which has a `"gt"` class type. The estimated model can be written as

$$\text{logit}(\hat{p}_{ik}) = -2.99 - 0.0516 Age_{ik} + 0.7362 Educ_{ik}$$

where $\hat{p}_{ik}$ is the estimated probability that the ith individual in the kth group is positive. In addition to the `summary.gt()` function, method functions to find residuals and predicted values are available.

We have also written a function `sim.g()` that simulates group test responses for a given binary regression model. The options within it allow for a user-specified set of covariates or one covariate that is simulated from a gamma distribution. Individuals are randomly put into groups of a specified size by the user. The function can also be used to simulate group testing data from a homogeneous population by specifying zero coefficients for covariates.

One of the most important innovations in group testing is the development of matrix or array-based pooling (Phatarfod and Sudbury, 1994; Kim et al., 2007). In this setting, specimens are placed into a matrix-like grid so that they can be pooled within each row and within each column. Potentially positive individuals occur at the intersection of positive rows and columns. If identification of these positive individuals is of interest, individual re-testing can be done on specimens at these intersections. With the advent of high-throughput screening, matrix pooling has become easier to perform because pooling and testing is done with minimal human intervention.

The `gtreg.mp()` function fits a group testing regression model in a matrix pooling setting. The row and column group responses can be used alone to fit the model. If individual re-testing is performed on the positive row and column intersections, these re-tests can be included when fitting the model. Note that the speed of model convergence can be improved by including re-tests. Within `gtreg.mp()`, we implement the EM algorithm given by Xie (2001) for matrix pooling settings where individual re-tests may or may not be performed. Due to the complicated response nature of matrix pooling, this algorithm involves using Gibbs sampling for the E-step in order to approximate the conditional expected values of a positive individual response.

Through personal communication with Minge Xie, we discovered that while he suggested the model fitting procedure could be used for matrix pooling, he had not implemented it; therefore, to our knowledge, this is the first time group testing regression models for a matrix pooling setting have been put into practice. Zhang and Bilder (2009) provide a technical report on the model fitting details. We hope that the `gtreg.mp()` function will encourage researchers to include covariates when performing matrix pooling rather than assume one common p, as has been done in the past.

The `sim.mp()` function simulates matrix pooling data. In order to simulate the data for a 5×6 and a 4×5 matrix, we can implement the following:

```
> set.seed(9128)
> sa1a <- sim.mp(par = c(-7,0.1), n.row = c(5,4),
+   linkf = "logit", n.col = c(6,5), sens = 0.95,
+   spec = 0.95)
> sa1 <- sa1a$dframe
> head(sa1)
       x col.resp row.resp coln rown arrayn retest
1 29.961        0        0    1    1      1     NA
2 61.282        0        1    1    2      1     NA
3 34.273        0        1    1    3      1     NA
4 46.190        0        0    1    4      1     NA
5 39.438        0        1    1    5      1     NA
6 45.880        1        0    2    1      1     NA
```

where `sa1` contains the column, row, and re-test responses along with one covariate `x`. The `coln`, `rown`, and `arrayn` variables are the column, row, and array numbers, respectively, for the responses. The covariate is simulated using the default gamma distribution with shape parameter 20 and scale parameter 2 (a user-specified matrix of covariates can also be used with the function). The `par` argument gives the coefficients in the model of $\text{logit}(p_{ijk}) = -7 + 0.1 x_{ijk}$ where x_{ijk} and p_{ijk} are the covariate and positivity probability, respectively, for the individual in row i, column j, and array k. We fit a model to the data using the following:

```
> fit1mp <- gtreg.mp(formula = cbind(col.resp,
+   row.resp) ~ x, data = sa1, coln = coln,
+   rown = rown, arrayn = arrayn, sens = 0.95,
+   spec = 0.95, linkf = "logit", n.gibbs = 2000)
> coef(fit1mp)
(Intercept)           x
-6.23982684  0.08659878
```

The coefficients are similar to the ones used to simulate the data. Methods to summarize the model's fit and to perform predictions are also available.

Conclusion

Group testing is used in a vast number of applications where a binary characteristic is of interest and individual specimens can be composited. Our package combines together the most often used and recommended confidence intervals for p. Also, our package makes the regression methods of Vansteelandt et al. (2000) and Xie (2001) easily accessible for the first time. We hope this will encourage researchers to take into account potentially important covariates in a group testing setting.

We see the current form of the **binGroup** package as a beginning rather than an end to meeting researcher needs. There are many additions that would be further helpful to researchers. For example, there are a number of re-testing protocols, such as halving (Gastwirth and Johnson, 1994) or sub-dividing positive groups of any size (Kim et al., 2007), that could be implemented, but would involve a large amount of new programming due to the complex nature of the re-testing. Also, the **binGroup** package does not have any functions solely for individual identification of a binary characteristic. For example, the optimal group size to use for identification alone is usually different to the optimal group size to use when estimating p. Given these desirable extensions, we encourage others to send us their functions or write new functions of their own. We would be willing to work with anyone to include them within the **binGroup** package. This would enable all researchers to have one group testing package rather than many small packages with much duplication.

Acknowledgements

This research is supported in part by Grant R01 AI067373 from the National Institutes of Health.

Bibliography

S. Dorai-Raj. **binom**: *Binomial Confidence Intervals for Several Parameterizations*, 2009. URL http://cran.r-project.org/web/packages/binom. R package version 1.0-4.

J. Gastwirth and W. Johnson. Screening with cost-effective quality control: Potential applications to HIV and drug testing. *Journal of the American Statistical Association*, 89:972–981, 1994.

G. Hepworth. Exact confidence intervals for proportions estimated by group testing. *Biometrics*, 52: 1134–1146, 1996.

J. Hughes-Oliver. Pooling experiments for blood screening and drug discovery. In *Screening: Methods for Experimentation in Industry, Drug Discovery, and Genetics*, edited by A. Dean and S. Lewis, New York: Springer.

H. Kim, M. Hudgens, J. Dreyfuss, D. Westreich, and C. Pilcher. Comparison of group testing algorithms for case identification in the presence of test error. *Biometrics*, 63:1152–1163, 2007.

J. Ornaghi, G. March, G. Boito, A. Marinelli, A. Beviacqua, J. Giuggia, and S. Lenardon. Infectivity in natural populations of *Delphacodes kuscheli* vector of 'Mal Rio Cuarto' virus. *Maydica*, 44:219–223, 1999.

C. Peck. Going after BVD. *Beef*, 42:34–44, 2006.

R. Phatarfod and A. Sudbury. The use of a square array scheme in blood testing. *Statistics in Medicine*, 13:2337–2343, 1994.

C. Pilcher, S. Fiscus, T. Nguyen, E. Foust, L. Wolf, D. Williams, R. Ashby, J. O'Dowd, J. McPherson, B. Stalzer, L. Hightow, W. Miller, J. Eron, M. Cohen, and P. Leone. Detection of acute infections during HIV testing in North Carolina. *New England Journal of Medicine*, 352:1873–1883, 2005.

K. Remlinger, J. Hughes-Oliver, S. Young, and R. Lam. Statistical design of pools using optimal coverage and minimal collision. *Technometrics*, 48: 133–143, 2006.

W. Swallow. Group testing for estimating infection rates and probabilities of disease transmission. *Phytopathology*, 75:882–889, 1985.

F. Schaarschmidt. Experimental design for one-sided confidence intervals or hypothesis tests in binomial group testing. *Communications in Biometry and Crop Science*, 2:32–40, 2007.

J. Tebbs and C. Bilder. Confidence interval procedures for the probability of disease transmission in multiple-vector-transfer designs. *Journal of Agricultural, Biological, and Environmental Statistics*, 9: 75–90, 2004.

S. Vansteelandt, E. Goetghebeur, and T. Verstraeten. Regression models for disease prevalence with diagnostic tests on pools of serum samples. *Biometrics*, 56:1126–1133, 2000.

T. Verstraeten, B. Farah, L. Duchateau, and R. Matu. Pooling sera to reduce the cost of HIV surveillance: a feasibility study in a rural Kenyan district. *Tropical Medicine and International Health*, 3: 747–750, 1998.

M. Xie. Regression analysis of group testing samples. *Statistics in Medicine*, 20:1957–1969, 2001.

B. Zhang and C. Bilder. The EM algorithm for group testing regression models under matrix pooling. Technical Report, University of Nebraska-Lincoln, Department of Statistics. URL http://www.chrisbilder.com/grouptesting.

B. Zhang, C. Bilder, and F. Schaarschmidt. **binGroup**: *Evaluation and experimental design for binomial group testing*, 2010. URL http://cran.r-project.org/web/packages/binGroup. R package version 1.0-5.

Christopher R. Bilder
Department of Statistics
University of Nebraska-Lincoln, Lincoln, NE
chris@chrisbilder.com

Boan Zhang
Department of Statistics
University of Nebraska-Lincoln, Lincoln, NE
boan.zhang@huskers.unl.edu

Frank Schaarschmidt
Institut für Biostatistik
Leibniz Universität Hannover, Germany
schaarschmidt@biostat.uni-hannover.de

Joshua M. Tebbs
Department of Statistics
University of South Carolina, Columbia, SC
tebbs@stat.sc.edu

The RecordLinkage Package: Detecting Errors in Data

by Murat Sariyar and Andreas Borg

Abstract Record linkage deals with detecting homonyms and mainly synonyms in data. The package **RecordLinkage** provides means to perform and evaluate different record linkage methods. A stochastic framework is implemented which calculates weights through an EM algorithm. The determination of the necessary thresholds in this model can be achieved by tools of extreme value theory. Furthermore, machine learning methods are utilized, including decision trees (**rpart**), bootstrap aggregating (**bagging**), ada boost (**ada**), neural nets (**nnet**) and support vector machines (**svm**). The generation of record pairs and comparison patterns from single data items are provided as well. Comparison patterns can be chosen to be binary or based on some string metrics. In order to reduce computation time and memory usage, blocking can be used. Future development will concentrate on additional and refined methods, performance improvements and input/output facilities needed for real-world application.

Introduction

When dealing with data from different sources that stem from and represent one realm, it is likely that homonym and especially synonym errors occur. In order to reduce these errors either different data files are linked or one data file is deduplicated. Record linkage is the task of reducing such errors through a vast number of different methods. These methods can be divided into two classes. One class consists of stochastic methods based on the framework of Fellegi and Sunter (1969). The other class comprises non-stochastic methods from the machine learning context. Methods from both classes need preliminary steps in order to generate data pairs from single data items. These record pairs are then transformed into comparison patterns. An exemplary comparison pattern is of the form $\gamma = (1,0,1,0,1,0,0,0)$ where only agreement and non-agreement of eight attributes are evaluated. This transformation and other preprocessing steps are described in the next section. Stochastic record linkage is primarily defined by the assumption of a probability model concerning probabilities of agreement of attributes conditional on the matching status of the underlying data pair. Machine learning methods reduce the problem of record linkage to a classification problem.

The package **RecordLinkage** is designed to facilitate the application of record linkage in R. The idea for this package evolved whilst using R for record linkage of data stemming from a German cancer registry. An evaluation of different methods thereafter lead to a large number of functions and data structures. The organisation of these functions and data structures as an R package eases the evaluation of record linkage methods and facilitates the application of record linkage to different data sets. These are the main goals behind the package described in this paper.

RecordLinkage is available from our project home page on R-Forge[1] as well as from CRAN.

Data preprocessing

First steps in data preprocessing usually include standardization of data, for example conversion of accented characters or enforcing a well-defined date format. However, such methods are not included in **RecordLinkage**. In the package, the user is responsible for providing the data in the form the package expects it.

Data must reside in a data frame where each row holds one record and columns represent attributes. The package includes two example data sets, `RLdata500` and `RLdata10000`, which differ in the number of records.[2] The following example shows the structure of `RLdata500`.

```
> library(RecordLinkage)
> data(RLdata500)
> RLdata500[1:5, ]

  fname_c1 fname_c2 lname_c1 lname_c2   by bm bd
1  CARSTEN     <NA>    MEIER     <NA> 1949  7 22
2     GERD     <NA>    BAUER     <NA> 1968  7 27
3   ROBERT     <NA> HARTMANN     <NA> 1930  4 30
4   STEFAN     <NA>    WOLFF     <NA> 1957  9  2
5     RALF     <NA>  KRUEGER     <NA> 1966  1 13
```

The fields in this data set are first name and family name, each split into a first and second component, and the date of birth, with separate components for day, month and year.

Column names are reused for comparison patterns (see below). If a record identifier other than the row number is desired, it should be included as a separate column instead of using `row.names`.

[1] http://r-forge.r-project.org/projects/recordlinkage/

[2] The data were created randomly from German name statistics and have no relation to existing persons.

Building comparison patterns

We include two functions for the creation of comparison patterns from data sets: `compare.dedup` for deduplication of a single data set and `compare.linkage` for linking two data sets together. In the case of three or more data sets, iteratively two of them are linked and replaced by the data set which is the result of the linkage. This leads to $n-1$ linkages for n data sets.

Both compare functions return an object of class `"RecLinkData"` which includes, among other components, the resulting comparison patterns as component `pairs`. In the following, such an object will be referred to as a *data object*.

```
> rpairs <- compare.dedup(RLdata500,
+   identity = identity.RLdata500)
> rpairs$pairs[1:5, ]

  id1 id2 fname_c1 fname_c2 lname_c1 lname_c2 by
1   1   2        0       NA        0       NA  0
2   1   3        0       NA        0       NA  0
3   1   4        0       NA        0       NA  0
4   1   5        0       NA        0       NA  0
5   1   6        0       NA        0       NA  0
  bm bd is_match
1  1  0        0
2  0  0        0
3  0  0        0
4  0  0        0
5  1  0        0
```

The printed slice of `datapairs$pairs` shows the structure of comparison patterns: The row numbers of the underlying records are followed by their corresponding comparison vector. Note that a missing value in either of two records results in a `NA` in the corresponding column of the comparison pattern. The classification procedures described in the following sections treat these `NA`s as zeros by default. If this behaviour is undesired, further preprocessing by the user is necessary. Column `is_match` denotes the true matching status of a pair (0 means non-match, 1 means match). It can be set externally by using the optional argument `identity` of the compare functions.[3] This allows evaluation of record linkage procedures based on labeled data as a gold standard.

Blocking

Blocking is the reduction of the amount of data pairs through focusing on specified agreement patterns.

Unrestricted comparison yields comparison patterns for all possible data pairs: $n(n-1)/2$ for deduplication of n records, $n \cdot m$ for linking two data sets with n and m records. Blocking is a common strategy to reduce computation time and memory consumption by only comparing records with equal values for a subset of attributes, called blocking fields. A blocking specification can be supplied to the `compare` functions via the argument `blockfld`. The most simple specification is a vector of column indices denoting the attributes on which two records must agree (possibly after applying a phonetic code, see below) to appear in the output. Combining several such specifications in a list leads to the union of the sets obtained by the individual application of the specifications. In the following example, two records must agree in either the first component of the first name or the complete date of birth to appear in the resulting set of comparison patterns.

```
> rpairs <- compare.dedup(RLdata500,
+   blockfld = list(1, 5:7),
+   identity = identity.RLdata500)
> rpairs$pairs[c(1:3, 1203:1204), ]

     id1 id2 fname_c1 fname_c2 lname_c1 lname_c2
1     17 119        1       NA        0       NA
2     61 106        1       NA        0       NA
3     61 175        1       NA        0       NA
1203  37  72        0       NA        0       NA
1204  44 339        0       NA        0       NA
     by bm bd is_match
1     0  0  0        0
2     0  0  1        0
3     0  0  1        0
1203  1  1  1        1
1204  1  1  1        0
```

Phonetic functions and string comparators

Phonetic functions and string comparators are similar, yet distinct approaches to dealing with typographical errors in character strings. A phonetic function maps words in a natural language to strings representing their pronunciation (the phonetic code). The aim is that words which sound similar enough get the same phonetic code. Obviously one needs different phonetic functions for different languages.[4] Package **RecordLinkage** includes the popular Soundex algorithm for English and a German language algorithm introduced by Michael (1999), implemented through functions `soundex` and `pho_h` respectively. The argument `phonetic` of the compare functions controls the application of the phonetic function, which defaults to `pho_h` and can be set by argument `phonfun`. Typically, an integer vector is supplied which specifies the indices of the data columns for which a phonetic code is to be computed before comparison with other records. Note that the phonetic function, if requested, is applied before the blocking process, therefore the equality restrictions imposed by blocking apply to phonetic codes. Consider, for example, a call with arguments `phonetic = 1:4` and `blockfld = 1`. In this case, the

[3] See documentation for `compare.*` for details.

[4] For this reason, problems may arise when records in one file stem from individuals from different nationalities.

actual blocking criterion is agreement on phonetic code of the first attribute.

String comparators measure the similarity between strings, usually with a similarity measure in the range $[0,1]$, where 0 denotes maximal dissimilarity and 1 equality. This allows 'fuzzy' comparison patterns as displayed in the following example.[5]

```
> rpairsfuzzy <- compare.dedup(RLdata500,
+   blockfld = c(5, 6), strcmp = TRUE)
> rpairsfuzzy$pairs[1:5, ]

  id1 id2  fname_c1 fname_c2  lname_c1 lname_c2
1 357 414 1.0000000       NA 1.0000000       NA
2 389 449 0.6428571       NA 0.0000000       NA
3 103 211 0.7833333       NA 0.5333333       NA
4   6 328 0.4365079       NA 0.4444444       NA
5  37  72 0.9750000       NA 0.9500000       NA
  by bm        bd is_match
1  1  1 0.7000000       NA
2  1  1 0.6666667       NA
3  1  1 0.0000000       NA
4  1  1 0.0000000       NA
5  1  1 1.0000000       NA
```

Controlling the application of string comparators works in the same manner as for phonetic functions, via the arguments `strcmp` and `strcmpfun`. The algorithms by Winkler (1990) (function `jarowinkler`) and one based on the edit distance by Levenshtein (function `levenshteinSim`) are included in the package. String comparison and phonetic encoding cannot be used simultaneously on one attribute but can be applied to different attributes within one set of comparison patterns, as in the function call `compare.dedup(RLdata500, phonetic = 1:4, strcmp = 5:7)`. We refer to the reference manual for further information.

Stochastic record linkage

Theory

Stochastic record linkage relies on the assumption of conditional probabilities concerning comparison patterns. The probabilities of the random vector $\gamma = (\gamma_1,...,\gamma_n)$ having value $\tilde{\gamma} = (\tilde{\gamma}_1,...,\tilde{\gamma}_n)$ conditional on the match status Z are defined by

$$u_{\tilde{\gamma}} = P(\gamma = \tilde{\gamma} \mid Z = 0),\ m_{\tilde{\gamma}} = P(\gamma = \tilde{\gamma} \mid Z = 1),$$

where $Z = 0$ stands for a non-match and $Z = 1$ for a match. In the Fellegi-Sunter model these probabilities are used to compute weights of the form

$$w_{\tilde{\gamma}} = \log\left(\frac{P(\gamma = \tilde{\gamma} \mid Z = 1)}{P(\gamma = \tilde{\gamma} \mid Z = 0)}\right).$$

These weights are used in order to discern between matches and non-matches.

There are several ways of estimating the probabilities involved in this model. In **RecordLinkage** an EM algorithm is used as a promising method for reliable estimations. The backbone of this algorithm is described by Haber (1984). We extended and implemented this algorithm in C in order to improve its performance. Without a proper stochastic model, the probabilities have to be fixed manually. If only weights are to be computed without relying on the assumptions of probabilities, then simple methods like the one implemented by Contiero et al. (2005) are suitable. Further details of these methods are also found in Sariyar et al. (2009).

Weight calculation based on the EM algorithm and the method by Contiero et al. (2005) are implemented by functions `emWeights` and `epiWeights`. Both take a data set object as argument and return a copy with the calculated weights stored in additional components. Calling `summary` on the result shows the distribution of weights in histogram style. This information can be helpful for determining classification thresholds, e.g. by identifying clusters of record pairs with high or low weights as non-matches or matches respectively.

```
> rpairs <- epiWeights(rpairs)
> summary(rpairs)

Deduplication Data Set

500 records
1221 record pairs

49 matches
1172 non-matches
0 pairs with unknown status


Weight distribution:

[0.15,0.2] (0.2,0.25] (0.25,0.3] (0.3,0.35]
      1011          0         89         30
(0.35,0.4] (0.4,0.45] (0.45,0.5] (0.5,0.55]
        29          8          7          1
(0.55,0.6] (0.6,0.65] (0.65,0.7] (0.7,0.75]
        14         19         10          2
(0.75,0.8]
         1
```

Discernment between matches and non-matches is achieved by means of computing weight thresholds. In the Fellegi-Sunter model, thresholds are computed via specification of destined (and feasible) values of homonym and synonym errors so that the amount of doubtable cases is minimized. In the package three auspicious variants for determining the threshold are implemented. The most common practice is to determine thresholds by clerical review, either a single threshold which separates links and non-links or separate thresholds for links

[5] Blocking is used in this example for the purpose of reducing computation time.

and non-links which define a range of doubtable cases between them. **RecordLinkage** supports this by the function `getPairs`, which shows record pairs aligned in two consecutive lines along with their weight (see the following example). When appropriate thresholds are found, classification is performed with `emClassify` or `epiClassify`, which take as arguments the data set object and one or two classification thresholds.

```
> tail(getPairs(rpairs, 0.6, 0.5))

      Weight  id fname_c1 fname_c2 lname_c1
25 0.5924569 266    KARIN     <NA>     HORN
26           437   KARINW     <NA>     HORN
27 0.5924569 395  GISOELA     <NA>     BECK
28           404   GISELA     <NA>     BECK
29 0.5067013 388   ANDREA     <NA>    WEBER
30           408   ANDREA     <NA>  SCHMIDT
   lname_c2   by bm bd
25     <NA> 2002  6  4
26     <NA> 2002  6  4
27     <NA> 2003  4 16
28     <NA> 2003  4 16
29     <NA> 1945  5 20
30     <NA> 1945  2 20

> result <- epiClassify(rpairs, 0.55)
```

The result is an object of class `"RecLinkResult"`, which differs from the data object in having a component `prediction` that represents the classification result. Calling `summary` on such an object shows error measures and a table comparing true and predicted matching status.[6]

```
> summary(result)

Deduplication Data Set

[...]

46 links detected
0 possible links detected
1175 non-links detected

alpha error: 0.061224
beta error: 0.000000
accuracy: 0.997543


Classification table:

           classification
true status    N    P    L
      FALSE 1172    0    0
      TRUE     3    0   46
```

One alternative to threshold determination by clerical review needs labeled training data on which the threshold for the whole data set is computed by minimizing the number of wrongly classified pairs. After weights have been calculated for these data, the classification threshold can be obtained by calling `optimalThreshold` on the training data object.

The other alternative for determining thresholds is an unsupervised procedure based on concepts of extreme value statistics. A mean excess plot is generated on which the interval representing the relevant area for false match rates is to be determined. Based on the assumption that this interval corresponds to a fat tail of the empirical weights distribution, the generalized Pareto distribution is used to compute the threshold discerning matches and non-matches. Details of this latter procedure will be found in a forthcoming paper which is still under review.

A function `getParetoThreshold` is included in the package which encapsulates the necessary steps. Called on a data set for which weights have been calculated, it brings up a mean excess plot of the weights. By clicking on the graph, the boundaries of the weight range in which matches and non-matches presumably overlap are selected. This area is usually discernible as a relatively long, approximately linear section in the middle region of the mean excess graph. This corresponds to the assumption that the generalized Pareto distribution is applicable. The data sets in the package provide only weak support for this assumption, especially because of their limited size. Figure 1 shows an example plot where the appropriate limits are displayed as dashed lines. The return value is a threshold which can be used with `emClassify` or `epiClassify`, depending on the type of weights. We refer to the package vignette *Classifying record pairs by means of Extreme Value Theory* for an example application.

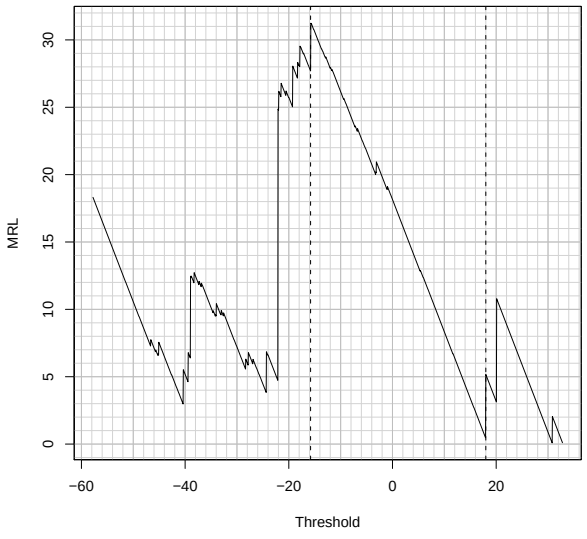


Figure 1: Example mean excess plot of weights with selected limits.

[6] True status is denoted by `TRUE` and `FALSE`, classification result by `"N"` (non-link),`"L"` (link) and `"P"` (possible link). To save space, some output is omitted, marked by '`[...]`' in this and some of the following examples

Machine learning methods

Record Linkage can be understood as a classification problem with comparison pattern γ as input and the matching status variable Z as output. With this view, a vast range of machine learning procedures, such as clustering methods, decision trees or support vector machines, becomes available for deduplicating or linking personal data. The following describes the application of machine learning to record linkage as it is implemented in the package.

Unsupervised classification

Unsupervised classification methods are attractive as they eliminate the necessity to provide representative training data, which can turn out to be a time-consuming task, involving either manual review or the finding of an adequate mechanism to generate artificial data. Motivated by this advantage, unsupervised clustering is incorporated into the package by means of the `classifyUnsup` function, which uses k-means clustering via `kmeans` or bagged clustering via `bclust` from package **e1071** (Dimitriadou et al., 2009). It must be noted that the quality of the resulting classification varies significantly for different data sets and poor results can occur. The following example shows the invocation of k-means clustering and the resulting classification.

```
> summary(classifyUnsup(rpairs, method = "kmeans"))

Deduplication Data Set

[...]

62 links detected
0 possible links detected
1159 non-links detected

alpha error: 0.000000
beta error: 0.011092
accuracy: 0.989353


Classification table:

           classification
true status    N    P    L
      FALSE 1159    0   13
      TRUE     0    0   49
```

Supervised classification

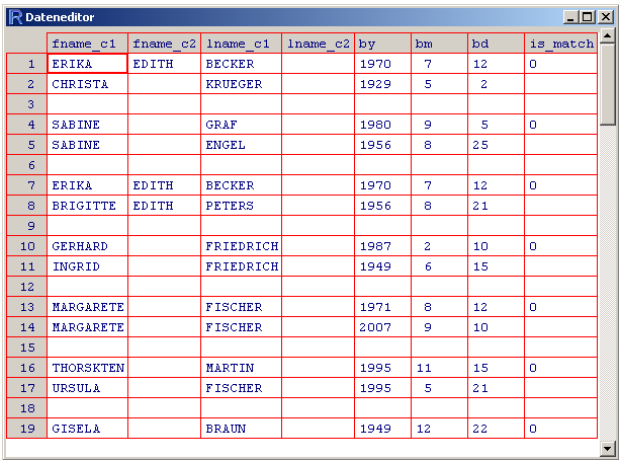
Training data for calibrating supervised classification methods can be obtained in a variety of ways in the context of this package. An important distinction is to be made between cases where additional training data with known true identity status are available and those where a training set has to be determined from the data on which linkage is performed. In the former case, one can provide a distinct set of records which are compared by one of the compare functions (see above) to obtain a training set. In the latter case, two approaches exist, first through what we call a *minimal training set*, second through unsupervised classification.

To construct a minimal training set, comparison patterns in a data set are grouped according to their configuration of agreement values. For every present configuration, one representative is randomly chosen. Naturally, this procedure is only feasible for binary comparisons (agreement or disagreement coded as 1 and 0).

In our experience, the use of supervised classification with a minimal training set can yield results similar to those that might be achieved with randomly sampled training sets of a substantially larger size. Their small magnitude allows minimal training sets to be classified by clerical review with manageable effort.

Two functions in **RecordLinkage** facilitate the use of minimal training sets. A set with the defined properties can be assembled by calling `getMinimalTrain` on a data object. Calling `editMatch` on the result opens an edit window which prints each record pair on two consecutive lines and allows for setting its matching status (as displayed in Figure 2). In the following example, 17 comparison patterns are selected randomly as a minimal training set.

```
> minTrain <- getMinimalTrain(rpairs)
> minTrain <- editMatch(minTrain)
```


Dateneditor

	fname_c1	fname_c2	lname_c1	lname_c2	by	bm	bd	is_match
1	ERIKA	EDITH	BECKER		1970	7	12	0
2	CHRISTA		KRUEGER		1929	5	2	
3								
4	SABINE		GRAF		1980	9	5	0
5	SABINE		ENGEL		1956	8	25	
6								
7	ERIKA	EDITH	BECKER		1970	7	12	0
8	BRIGITTE	EDITH	PETERS		1956	8	21	
9								
10	GERHARD		FRIEDRICH		1987	2	10	0
11	INGRID		FRIEDRICH		1949	6	15	
12								
13	MARGARETE		FISCHER		1971	8	12	0
14	MARGARETE		FISCHER		2007	9	10	
15								
16	THORSKTEN		MARTIN		1995	11	15	0
17	URSULA		FISCHER		1995	5	21	
18								
19	GISELA		BRAUN		1949	12	22	0

Figure 2: Edit window for clerical review

The second approach to obtaining training data when no labelled data are available, provided by function `genSamples`, uses unsupervised clustering in a manner similar to `classifyUnsup`. Instead of directly classifying all the data, a subset is extracted and classified by bagged clustering. It can then be used to calibrate a supervised classifier which is ultimately applied to the remaining set. Arguments to `genSamples` are the original data set, the number of non-matches to appear in the training set and the

desired ratio of matches to non-matches. For example, the call `genSamples(datapairs, num.non = 200, des.mprop = 0.1)` tries to build a training set with 200 non-matches and 20 matches. The return value is a list with two disjoint sets named `train` and `valid`.

In an scenario where different record linkage approaches are evaluated, it is useful to split a data set into training and validation sets. Such a split can be performed by `splitData`. The return value is a list with components `train` and `valid` as in the case of `genSamples`. The most basic usage is to specify a fraction of patterns that is drawn randomly as a training set using the argument `prop`. For example, `splitData(rpairs, prop = 0.1)` selects one tenth of the data for training. By setting `keep.mprop = TRUE` it can be enforced that the original ratio of matches to non-matches is retained in the resulting sets. Another possibility is to set the desired number of non-matches and the match ratio through arguments `num.non` and `mprop` as with `genSamples()`.

Classification functions

All classification methods share two interface functions: `trainSupv` for calibrating a classifier, `classifySupv` for classifying new data. At least two arguments are required for `trainSupv`, the data set on which to train and a string representing the classification method. Currently, the supported methods are:

`"rpart"` Recursive partitioning trees, provided by package **rpart** (Therneau et al., 2009).

`"bagging"` Bagging of decision trees, provided by package **ipred** (Peters and Hothorn, 2009).

`"ada"` Stochastic boosting, provided by package **ada** (Culp et al., 2006).

`"svm"` Support vector machines, provided by package **e1071** (Dimitriadou et al., 2009).

`"nnet"` Single-hidden-layer neural networks, provided by package **e1071** (ibid.).

Of the further arguments, `use.pred` is noteworthy. Setting it to `TRUE` causes `trainSupv` to treat the result of a previous prediction as outcome variable. This has to be used if the training data stem from a run of `genSamples`. Another application is to obtain a training set by classifying a fraction of the data set by the process of weight calculation and manual setting of thresholds. In order to train a supervised classifier with this training set, one would have to set `use.pred = TRUE`.

The return value is an object of class `"RecLinkClassif"`. Classification of new data is carried out by passing it and a `"RecLinkData"` object to `classifySupv`. The following example shows the application of bagging based on the minimal training set `minTrain` from the example above.

```
> model <- trainSupv(minTrain, method = "bagging")
> result <- classifySupv(model, newdata = rpairs)
> summary(result)

Deduplication Data Set

[...]

53 links detected
0 possible links detected
1168 non-links detected

alpha error: 0.020408
beta error: 0.004266
accuracy: 0.995086


Classification table:

           classification
true status    N    P    L
      FALSE 1167    0    5
      TRUE     1    0   48
```

Discussion

During its development, the functionalities of the package **RecordLinkage** have already been used by the authors for a period of about one year, individual functions even before that. Thanks to this process of parallel development and usage, bugs were detected early and new requirements that became apparent were accounted for by implementing new features. Therefore, the package is generally in a usable state. However, there is still potential for future improvements, mainly regarding data handling and performance.

RecordLinkage was developed mainly as a tool for empirical evaluation of record linkage methods, i.e. the main focus of our research was on the performance of particular record linkage methods. Only in one case, the evaluation of a deterministic record linkage procedure used in a German cancer registry, were we actually interested in detecting duplicates. Development of the package followed the needs of this detection process. But many other desirable improvements concerning data processing and input/ouput facilities are needed for a real-world record linkage process, such as:

- The possibility to use a database connection for data input and output.
- Improved print facilities for clerical review, such as highlighting of agreement or disagreement in record pairs.

- The possibility to output a deduplicated data set based on the matching result. This is not a trivial task as it requires choosing the most likely record from each group of matching items which can be accomplished by means of linear programming.

Another important future task is performance enhancement. When handling large data sets of about 10^6 or more record pairs, high memory consumption often leads to errors or destabilizes the computer system R is run on. We are currently working on code changes to avoid unnecessary copying of objects and plan to incorporate these in future releases. Another possibility, which needs further evaluation, is to use more sophisticated ways of storing data, such as a database or one of the various R packages for large data sets. The current disadvantage of R concerning performance and memory can be compensated by an elaborated blocking strategy.

As a general improvement, it is intended to support the usage of algorithms not considered in the package through a generic interface for supervised or unsupervised classification.

We are aware that **RecordLinkage** lacks an important subtask of record linkage, the standardization of records. However, this is an area that itself would need extensive research efforts. Moreover, the appropriate procedures depend heavily on the data to be linked. It is therefore left to the users to tailor standardization methods fitting the requirements of their data. We refer to R's capabilities to handle regular expressions and to the CRAN task view *NaturalLanguageProcessing*[7].

Bibliography

P. Contiero et al. The Epilink record linkage software. *Methods Inf Med.*, 44(1):66–71, 2005.

M. Culp, K. Johnson, and G. Michailidis. *ada: Performs boosting algorithms for a binary response*, 2006. URL http://www.stat.lsa.umich.edu/~culpm/math/ada/img.html. R package version 2.0-1.

E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2009. URL http://CRAN.R-project.org/package=e1071. R package version 1.5-19.

I. Fellegi and A. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

E. Haber. AS207: Fitting a general log-linear model. *Applied Statistics*, 33:358–362, 1984.

J. Michael. Doppelgänger gesucht – ein Programm für kontextsensitive phonetische Textumwandlung. *c't*, 17(25):252–261, 1999. URL http://www.heise.de/ct/ftp/99/25/252/.

A. Peters and T. Hothorn. *ipred: Improved Predictors*, 2009. URL http://CRAN.R-project.org/package=ipred. R package version 0.8-7.

M. Sariyar, A. Borg, and K. Pommerening. Evaluation of record linkage methods for iterative insertions. *Methods Inf Med.*, 48(5):429–437, 2009.

T. M. Therneau, B. Atkinson, and B. Ripley (R port). *rpart: Recursive Partitioning*, 2009. URL http://CRAN.R-project.org/package=rpart. R package version 3.1-45.

W. E. Winkler. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pages 354–369, 1990. URL www.amstat.org/sections/srms/proceedings/papers/1990_056.pdf.

Murat Sariyar
Institute of Medical Biostatistics, Epidemiology and Informatics
Mainz
Germany
sariyar@imbei.uni-mainz.de

Andreas Borg
Institute of Medical Biostatistics, Epidemiology and Informatics
Mainz
Germany
borg@imbei.uni-mainz.de

[7] http://cran.r-project.org/view=NaturalLanguageProcessing

spikeslab: Prediction and Variable Selection Using Spike and Slab Regression

by Hemant Ishwaran, Udaya B. Kogalur and J. Sunil Rao

Abstract Weighted generalized ridge regression offers unique advantages in correlated high-dimensional problems. Such estimators can be efficiently computed using Bayesian spike and slab models and are effective for prediction. For sparse variable selection, a generalization of the elastic net can be used in tandem with these Bayesian estimates. In this article, we describe the R-software package **spikeslab** for implementing this new spike and slab prediction and variable selection methodology.

The expression *spike and slab*, originally coined by Mitchell and Beauchamp (1988), refers to a type of prior used for the regression coefficients in linear regression models (see also Lempers (1971)). In Mitchell and Beauchamp (1988), this prior assumed that the regression coefficients were mutually independent with a two-point mixture distribution made up of a uniform flat distribution (the slab) and a degenerate distribution at zero (the spike). In George and McCulloch (1993) a different prior for the regression coefficient was used. This involved a scale (variance) mixture of two normal distributions. In particular, the use of a normal prior was instrumental in facilitating efficient Gibbs sampling of the posterior. This made spike and slab variable selection computationally attractive and heavily popularized the method.

As pointed out in Ishwaran and Rao (2005), normal-scale mixture priors, such as those used in George and McCulloch (1993), constitute a wide class of models termed *spike and slab models*. Spike and slab models were extended to the class of rescaled spike and slab models (Ishwaran and Rao, 2005). Rescaling was shown to induce a non-vanishing penalization effect, and when used in tandem with a continuous bimodal prior, confers useful model selection properties for the posterior mean of the regression coefficients (Ishwaran and Rao, 2005, 2010).

Recently, Ishwaran and Rao (2010) considered the geometry of generalized ridge regression (GRR), a method introduced by Hoerl and Kennard to overcome ill-conditioned regression settings (Hoerl and Kennard, 1970a,b). This analysis showed that GRR possesses unique advantages in high-dimensional correlated settings, and that weighted GRR (WGRR) regression, a generalization of GRR, is potentially even more effective. Noting that the posterior mean of the regression coefficients from a rescaled spike and slab model is a type of WGRR estimator, they showed that this WGRR estimator, referred to as the Bayesian model averaged (BMA) estimator, when coupled with dimension reduction, yielded low test-set mean-squared-error when compared to the elastic net (Zou and Hastie, 2005).

Additionally, Ishwaran and Rao (2010) introduced a generalization of the elastic net, which they coined the gnet (short for generalized elastic net). The gnet is the solution to a least-squares penalization problem in which the penalization involves an overall ℓ_1-regularization parameter (used to impose sparsity) and a unique ℓ_2-regularization parameter for each variable (these latter parameters being introduced to combat multicollinearity). To calculate the gnet, a lasso-type optimization is used by fixing the ℓ_2-regularization parameters at values determined by finding the closest GRR to the BMA. Like the BMA, the gnet is highly effective for prediction. However, unlike the BMA, which is obtained by model averaging, and therefore often contains many small coefficient values, the gnet is much sparser, making it more attractive for variable selection.

The gnet and BMA estimators represent attractive solutions for modern day high-dimensional data settings. These problems often involve correlated variables, in part due to the nature of the data, and in part due to an artifact of the dimensionality [see Cai and Lv (2007); Fan and Lv (2008) for a detailed discussion about high-dimensional correlation]. The BMA is attractive because it addresses the issue of correlation by drawing upon the properties of WGRR estimation, a strength of the Bayesian approach, while the gnet achieves sparse variable selection by drawing upon the principle of soft-thresholding, a powerful frequentist regularization concept.

Because high-dimensional data is becoming increasingly common, it would be valuable to have user friendly software for computing the gnet and BMA estimator. With this in mind, we have developed an R package **spikeslab** for implementing this methodology (Ishwaran, Kogalur and Rao, 2010).

The main purpose of this article is to describe this package. Because this new spike and slab approach may be unfamiliar to users in the R-community, we start by giving a brief high-level description of the algorithm [for further details readers should however consult Ishwaran and Rao (2010)]. We then highlight some of the package's key features, illustrating its use in both low- and high-dimensional settings.

The spike and slab algorithm

The **spikeslab** R package implements the rescaled spike and slab algorithm described in Ishwaran and Rao (2010). This algorithm involves three key steps:

1. Filtering (dimension reduction).
2. Model Averaging (BMA).
3. Variable Selection (gnet).

Step 1 filters all but the top nF variables, where n is the sample size and $F > 0$ is the user specified fraction. Variables are ordered on the basis of their absolute posterior mean coefficient value, where the posterior mean is calculated using Gibbs sampling applied to an approximate rescaled spike and slab posterior. Below is a toy-illustration of how filtering works [p is the total number of variables and $(V_{(k)})_{k=1}^{p}$ are the ordered variables]:

$$\underbrace{V_{(1)},\ldots,V_{([nF])}}_{\text{retain these variables}} \qquad \underbrace{V_{([nF]+1)},\ldots,V_{(p)}}_{\text{filter these variables}}.$$

The value for F is set using the option `bigp.smalln.factor`, which by default is set to the value $F = 1$. The use of an approximate posterior in the filtering step is needed in high dimensions. This yields an ultra-fast Gibbs sampling procedure, with each step of the Gibbs sampler requiring $O(np)$ operations. Thus, computational effort for the filtering step is linear in both dimensions.

Step 2 fits a rescaled spike and slab model using only those variables that are not filtered in Step 1. Model fitting is implemented using a Gibbs sampler. Computational times are generally rapid as the number of variables at this point are a fraction of the original size, p. A blocking technique is used to further reduce computational times. The posterior mean of the regression coefficients, which we refer to as the BMA, is calculated and returned. This (restricted) BMA is used as an estimator for the regression coefficients.

Step 3 calculates the gnet. In the optimization, the gnet's ℓ_2-regularization parameters are fixed (these being determined from the restricted BMA obtained in Step 2) and its solution path with respect to its ℓ_1-regularization parameter is calculated using the **lars** R package (Hastie and Efron, 2007) [a package dependency of **spikeslab**]. The `lars` wrapper is called with `type="lar"` to produce the full LAR path solution (Efron et al., 2004). The gnet is defined as the model in this path solution minimizing the AIC criterion. Note importantly, that cross-validation is not used to optimize the ℓ_1-regularization parameter. The gnet estimator is generally very stable, a property that it inherits from the BMA, and thus even simple model selection methods such as AIC work quite well in optimizing its path solution. This is different than say the elastic net (Zou and Hastie, 2005) where cross-validation is typically used to determine its regularization parameters (often this involves a double optimization over both the ℓ_1- and ℓ_2-regularization parameters). This is an important feature which reduces computational times in big-p problems.

Low-dimensional settings

Although the spike and slab algorithm is especially adept in high-dimensional settings, it can be used effectively in classical settings as well. In these low-dimensional scenarios when $p < n$, the algorithm is implemented by applying only Steps 2 and 3 (i.e., Step 1 is skipped). The default setting for **spikeslab**, in fact, assumes a low-dimensional scenario.

As illustration, we consider the benchmark diabetes data ($n = 442$, $p = 64$) used in Efron et al. (2004) and which is an example dataset included in the package. The response Y is a quantitative measure of disease progression for patients with diabetes. The data includes 10 baseline measurements for each patient, in addition to 45 interactions and 9 quadratic terms, for a total of 64 variables for each patient. The following code implements a default analysis:

```
data(diabetesI, package = "spikeslab")
set.seed(103608)
obj <- spikeslab(Y ~ . , diabetesI)
print(obj)
```

The `print` call outputs a basic summary of the analysis, including a list of the selected variables and their parameter estimates (variables selected are those having nonzero gnet coefficient estimates):

	bma	gnet	bma.scale	gnet.scale
bmi	24.076	23.959	506.163	503.700
ltg	23.004	22.592	483.641	474.965
map	14.235	12.894	299.279	271.089
hdl	-11.495	-10.003	-241.660	-210.306
sex	-7.789	-6.731	-163.761	-141.520
age.sex	6.523	5.913	137.143	124.322
bmi.map	3.363	4.359	70.694	91.640
glu.2	2.185	3.598	45.938	75.654
age.ltg	1.254	0.976	26.354	20.528
bmi.2	1.225	1.837	25.754	38.622
age.map	0.586	0.928	12.322	19.515
age.2	0.553	0.572	11.635	12.016
sex.map	0.540	0.254	11.349	5.344
glu	0.522	0.628	10.982	13.195
age.glu	0.417	0.222	8.757	4.677

In interpreting the table, we note the following:

(i) The first column with the heading *bma* lists the coefficient estimates for the BMA estimator obtained from Step 2 of the algorithm. These values are given in terms of the standardized covariates (mean of 0 and variance of 1).

(ii) The second column with the heading *gnet* lists the coefficient estimates for gnet obtained from Step 3 of the algorithm. These values are also given in terms of the standardized covariates.

(iii) The last two columns are the BMA and gnet estimators given in terms of the original scale of the variables. These columns are used for prediction, while the first two columns are useful for assessing the relative importance of variables.

Note that all columns can be extracted from the spike and slab object, `obj`, if desired.

Stability analysis

Even though the gnet accomplishes the goal of variable selection, it is always useful to have a measure of stability of a variable. The wrapper `cv.spikeslab` can be used for this purpose.

The call to this wrapper is very simple. Here we illustrate its usage on the diabetes data:

```
y <- diabetesI[,  1]
x <- diabetesI[, -1]
cv.obj <- cv.spikeslab(x = x, y = y, K = 20)
```

This implements 20-fold validation (the number of folds is set by using the option `K`). The gnet estimator is fit using the training data and its test-set mean-squared-error (MSE) for its entire solution-path is determined. As well, for each fold, the optimal gnet model is determined by minimizing test-set error. The average number of times a variable is selected in this manner defines its stability (this is recorded in percentage as a value from 0%-100%). Averaging the gnet's test-set MSE provides an estimate of its MSE as a function of the number of variables.

The gnet's coefficient values (estimated using the full data) and its stability values can be obtained from the `cv.obj` using the following commands:

```
cv.stb <- as.data.frame(cv.obj$stability)
gnet <- cv.stb$gnet
stability <- cv.stb$stability
```

Figure 1 (top) plots the gnet's cross-validated MSE curve as a function of the model size. The plot was produced with the command

```
plot(cv.obj, plot.type = "cv")
```

Close inspection (confirmed by considering the object, `cv.obj`) shows that the optimal model size is somewhere between 9 and 17, agreeing closely with our previous analysis. The bottom plot shows how gnet coefficient estimates vary in terms of their stability values (obtained by plotting `gnet` versus `stability`). There are 10 variables having stability values greater than 80%.

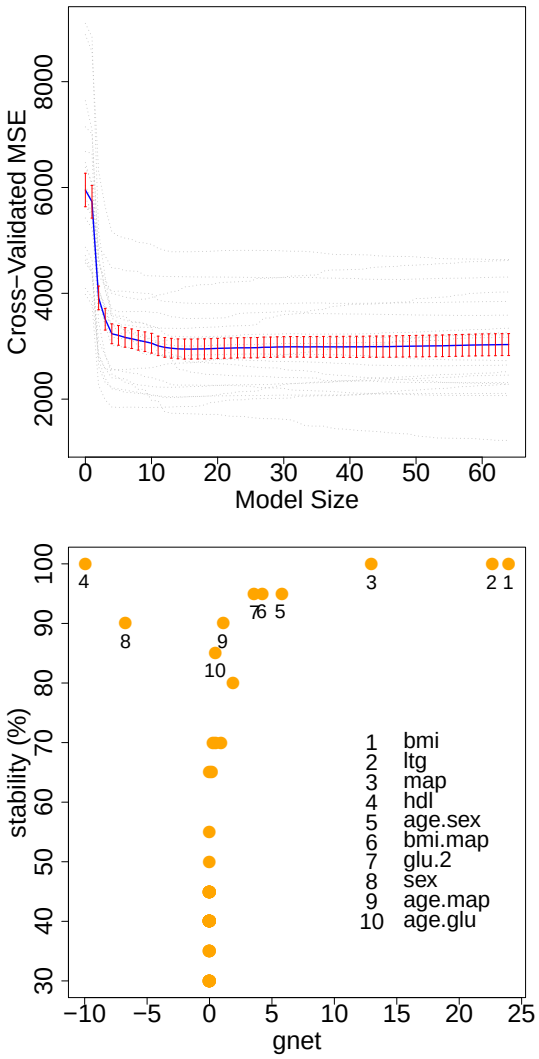


Figure 1: Stability analysis for diabetes data.

High-dimensional settings

To analyze $p \gg n$ data, users should use the option `bigp.smalln=TRUE` in the call to `spikeslab`. This will invoke the full spike and slab algorithm including the filtering step (Step 1) which is crucial to success in high-dimensional settings (note that $p \geq n$ for this option to take effect). This three-step algorithm is computationally efficient, and because the bulk of the computations are linear in p, the algorithm should scale effectively to very large p-problems. However, in order to take full advantage of its speed, there are a few simple, but important rules to keep in mind.

First, users should avoid using the formula and data-frame call to `spikeslab` when p is large. Instead they should pass the x-covariate matrix and y-response vector directly. This avoids the tremendous overhead required to parse formula in R.

Second, the final model size of the BMA and gnet are controlled by two key options; these must be set properly to keep computations manageable. These

options are: `bigp.smalln.factor` and `max.var`. The first option restricts the number of filtered variables in Step 1 of the algorithm to be no larger than Fn, where $F > 0$ is the value `bigp.smalln.factor`. The default setting $F = 1$ should be adequate in most scenarios [one exception is when n is very large (but smaller than p); then F should be decreased to some value $0 < F < 1$]. The second option, `max.var`, restricts the number of selected variables in *both* Steps 1 and 3 of the algorithm. Its function is similar to `bigp.smalln.factor`, although unlike `bigp.smalln.factor`, it directly controls the size of gnet. The default value is `max.var=500`. In most examples, it will suffice to work with `bigp.smalln.factor`.

Thus, if x is the x-matrix, y is the y-response vector, and f and m are the desired settings for `bigp.smalln.factor` and `max.var`, then a generic call in high-dimensional settings would look like:

```
obj <- spikeslab(x=x, y=y, bigp.smalln = TRUE,
     bigp.small.n.factor = f, max.var = m)
```

Although `spikeslab` has several other options, most users will not need these and the above call should suffice for most examples. However, if computational times are a still of concern even after tuning f and m, users may consider changing the default values of `n.iter1` and `n.iter2`. The first controls the number of burn-in iterations used by the Gibbs sampler, and the second controls the number of Gibbs sampled values following burn-in (these latter values are used for inference and parameter estimation). The default setting is 500 in both cases. Decreasing these values will decrease computational times, but accuracy will suffer. Note that if computational times are not a concern, then both values could be increased to 1000 (but not much more is needed) to improve accuracy.

As illustration, we used a simulation with $n = 100$ and $p = 2000$. The data was simulated independently in blocks of size 40. Within each block, the x-variables were drawn from a 50-dimensional multivariate normal distribution with mean zero and equicorrelation matrix with $\rho = 0.95$. With probability 0.9, all regression coefficients within a block were set to zero, otherwise with probability 0.1, all regression coefficients were set to zero except for the first 10 coefficients, which were each assigned a randomly chosen value from a standard normal distribution. Random noise ε was simulated independently from a $N(0,\sigma^2)$ distribution with $\sigma = 0.4$.

The top plot in Figure 2 displays the path solution for the gnet. Such a plot can be produced by a call to the **lars** wrapper `plot.lars` using the `gnet.obj` obtained from the `spikeslab` call. As `gnet.obj` is a **lars**-type object it is fully interpretable by the **lars** package, and thus it can be parsed by the packages' various wrappers. For convenience, the path solution can be produced by a direct call to `plot`; a typical call being:

```
obj <- spikeslab(x=x, y=y, bigp.smalln = TRUE)
plot(obj, plot.type = "path")
```

Actually Figure 2 was not produced by a call to `plot` but in fact was obtained by slightly modifying the `plot.lars` wrapper so as to display the paths of a variable color coded by its true coefficient value (blue for truly zero and red for truly nonzero). We did this in order to facilitate comparison to the lasso. The lasso path (obtained using the LAR-solution) is displayed in the bottom plot of Figure 2. Notice how in contrast to gnet, the path solution for the lasso has a wiggly "spaghetti"-like shape and that many of the truly nonzero coefficients are hard to identify because of this. This a direct consequence of the high-correlation in the x-variables of this example. This correlation creates instability in the lasso-LAR solution, and this ultimately impacts its performance.

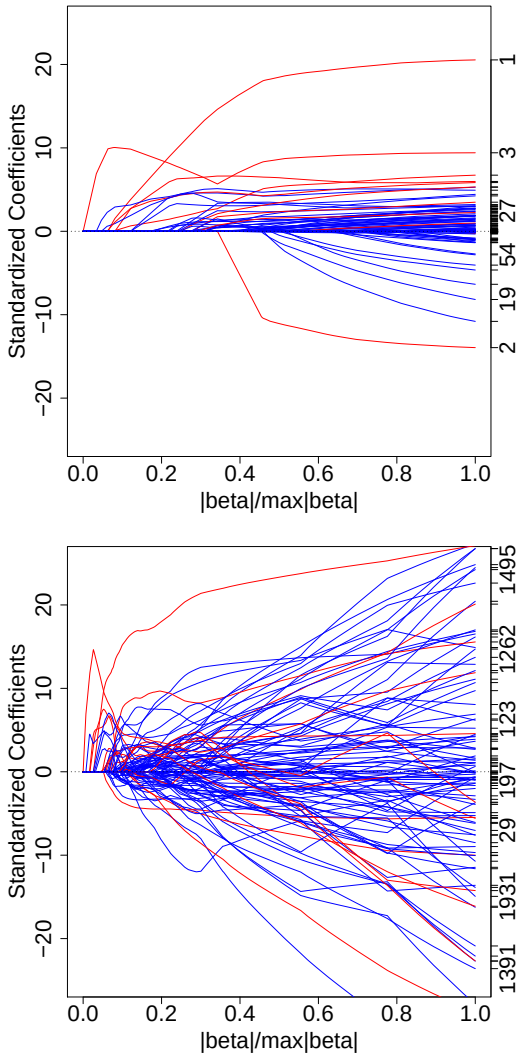


Figure 2: Path solutions for the gnet (top) and the lasso (bottom) from a correlated high-dimensional simulation ($n = 100$ and $p = 2000$). Blue and red lines correspond to truly zero and truly nonzero coefficient values, respectively.

Summary

The gnet incorporates the strength of Bayesian WGRR estimation with that of frequentist soft thresholding. These combined strengths make it an effective tool for prediction and variable selection in correlated high-dimensional settings. If variable selection is not of concern, and the key issue is accurate prediction, than the BMA may be preferred. Both the gnet and BMA can be computed using the **spikeslab** R package. This package is computationally efficient, and scales effectively even to massively large p-problems.

As one example of this scalability, we added 100,000 noise variables to the diabetes data set and then made a call to `cv.spikeslab` with the added options `bigp.smalln = TRUE`, `max.var = 100` and `parallel = TRUE` (as before we used $K = 20$ fold validation). The `parallel` option invokes parallel processing that is implemented via the package **snow** (Tierney et al., 2008) [note that sending in an integer for the option `parallel` sets the number of socket clusters on the local machine on which the session is being initiated; in our example we actually used `parallel = 8`]. The **snow** package should be loaded prior to making the `cv.spikeslab` call.

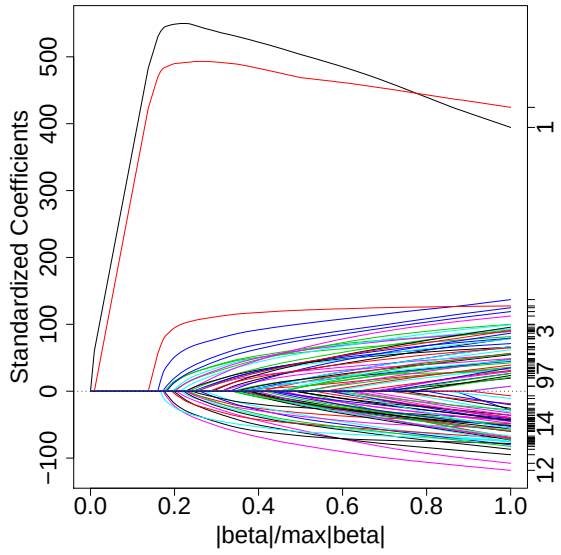


Figure 3: Path solution for the gnet for diabetes data with $p = 100,000$ noise variables.

Figure 3 displays the gnet's path solution (obtained using the full data). While only 4 variables have path-profiles that clearly stand out, impressively these variables are the top 4 from our previous analysis. The gnet estimates (scaled to standardized covariates), its averaged cv-estimates, and the stability values for the top 15 variables were:

```
              gnet     gnet.cv stability
ltg     20.1865498 20.14414604       100
bmi     18.7600433 22.79892835       100
map      4.7111022  4.83179363        95
hdl     -2.5520177 -2.66839785        95
bmi.2    3.6204750  0.46308305        40
x.61746  0.0000000 -0.74646210        35
x.42036  4.8342736  0.41993669        30
x.99041  0.0000000 -0.70183515        30
x.82308  5.2728011  0.75420320        25
glu      1.3105751  0.16714059        25
x.46903  0.0000000 -0.65188451        25
x.57061  0.0000000  0.73203633        25
x.99367 -2.7695621 -0.22110463        20
tch      0.2542299  0.14837708        20
x.51837  0.0000000 -0.09707276        20
```

Importantly, note that the top 4 variables have greater than or equal to 95% stability (variables starting with "x." are noise variables). It is also interesting that 3 other non-noise variables, "bmi.2", "glu", and "tch" were in the top 15 variables. In fact, when we inspected the 100 variables that passed the filtering step of the algorithm (applied to the full data), we found that 10 were from the original 64 variables, and 6 were from the top 15 variables from our earlier analysis. This demonstrates stability of the filtering algorithm even in ultra-high dimensional problems.

Finally, we remark that in illustrating the **spikeslab** package in this article, we focused primarily on the `spikeslab` wrapper, which is the main entry point to the package. Other available wrappers include `predict.spikeslab` for prediction on test data, and `sparsePC.spikeslab`. The latter implements variable selection for multiclass gene expression data (Ishwaran and Rao, 2010).

In future work we plan to extend the rescaled spike and slab methodology to high-dimensional generalized linear models. At that time we will introduce a corresponding wrapper.

Acknowledgements

This work was partially funded by the National Science Foundation grant DMS-0705037. We thank Vincent Carey and two anonymous referees for helping to substantially improve this manuscript.

Bibliography

T.T. Cai. and J. Lv. Discussion: The Dantzig selector: statistical estimation when p is much larger than n. *Ann. Statist.*, 35:2365, 2007.

B. Efron, T. Hastie, I. Johnstone and R. Tibshirani. Least angle regression (with discussion). *Ann. Statist.*, 32:407–499, 2004.

J. Fan and J. Lv. Sure independence screening for ultra-high dimensional feature space. *J. Royal Statist. Soc. B*, 70(5):849–911, 2008.

E.I. George and R.E. McCulloch. Variable selection via Gibbs sampling. *J. Amer. Stat. Assoc.*, 88:881–889, 1993.

T. Hastie and B. Efron. *lars: Least Angle Regression, Lasso and Forward Stagewise*, 2010. R package version 0.9-7.

A.E. Hoerl and R.W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

A.E. Hoerl and R.W. Kennard. Ridge regression: Applications to nonorthogonal problems (Corr: V12 p723). *Technometrics*, 12:69–82, 1970.

H. Ishwaran and J.S. Rao. Spike and slab variable selection: frequentist and Bayesian strategies. *Ann. Statist.*, 33:730–773, 2005.

H. Ishwaran and J.S. Rao. Generalized ridge regression: geometry and computational solutions when p is larger than n. *Manuscript*, 2010.

H. Ishwaran, U.B. Kogalur and J.S. Rao. *spikeslab: Prediction and variable selection using spike and slab regression*, 2010. R package version 1.1.2.

F. B. Lempers. *Posterior Probabilities of Alternative Linear Models*. Rotterdam University Press, Rotterdam, 1971.

T.J. Mitchell and J.J. Beauchamp. Bayesian variable selection in linear regression. *J. Amer. Stat. Assoc.*, 83:1023–1036, 1988.

L. Tierney, A. J. Rossini, N. Li and H. Sevcikova *snow: Simple Network of Workstations*, 2008. R package version 0.3-3.

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *J. Royal Statist. Soc. B*, 67(2):301–320, 2005.

Hemant Ishwaran
Dept. Quantitative Health Sciences
Cleveland Clinic
Cleveland, OH 44195
hemant.ishwaran@gmail.com

Udaya B. Kogalur
Dept. Quantitative Health Sciences, Cleveland Clinic
Cleveland, OH 44195
kogalurshear@gmail.com

J. Sunil Rao
Dept. of Epidemiology and Public Health
School of Medicine, University of Miami
Miami, FL 33136
rao.jsunil@gmail.com

What's New?

by Kurt Hornik and Duncan Murdoch

Abstract We discuss how news in R and add-on packages can be disseminated. R 2.10.0 has added facilities for computing on news in a common plain text format. In R 2.12.0, the Rd format has been further enhanced so that news can be very conveniently recorded in Rd, allowing for both improved rendering and the development of new news services.

The R development community has repeatedly discussed how to best represent and disseminate the news in R and R add-on packages. The GNU Coding Standards (http://www.gnu.org/prep/standards/html_node/NEWS-File.html) say

> In addition to its manual, the package should have a file named 'NEWS' which contains a list of user-visible changes worth mentioning. In each new release, add items to the front of the file and identify the version they pertain to. Don't discard old items; leave them in the file after the newer items. This way, a user upgrading from any previous version can see what is new.
>
> If the 'NEWS' file gets very long, move some of the older items into a file named 'ONEWS' and put a note at the end referring the user to that file.

For a very long time R itself used a three-layer (series, version, category) plain text 'NEWS' file recording individual news items in itemize-type lists, using 'o' as item tag and aligning items to a common column (as quite commonly done in open source projects).

R 2.4.0 added a function `readNEWS()` (eventually moved to package **tools**) for reading the R 'NEWS' file into a news hierarchy (a series list of version lists of category lists of news items). This makes it possible to compute on the news (e.g. changes are displayed daily in the RSS feeds available from http://developer.r-project.org/RSSfeeds.html), and to transform the news structure into sectioning markup (e.g. when creating an HTML version for reading via the on-line help system). However, the items were still plain text: clearly, it would be useful to be able to hyperlink to resources provided by the on-line help system (Murdoch and Urbanek, 2009) or other URLs, e.g., those providing information on bug reports. In addition, it has always been quite a nuisance to add LaTeX markup to the news items for the release notes in the R Journal (and its predecessor, R News). The R Core team had repeatedly discussed the possibilities of moving to a richer markup system (and generating plain text, HTML and LaTeX from this), but had been unable to identify a solution that would be appropriate in terms of convenience as well as the effort required.

R add-on packages provide news-type information in a variety of places and plain text formats, with a considerable number of 'NEWS' files in the package top-level source directory or its 'inst' subdirectory, and in the spirit of a one- or two-layer variant of the R news format. Whereas all these files can conveniently be read by "users", the lack of standardization implies that reliable computation on the news items is barely possible, even though this could be very useful. For example, upgrading packages could optionally display the news between the available and installed versions of packages (similar to the `apt-listchanges` facility in Debian-based Linux systems). One could also develop repository-level services extracting and disseminating the news in all packages in the repository (or suitable groups of these, such as task views (Zeileis, 2005)) on a regular basis. In addition, we note that providing the news in plain text limits the usefulness in package web pages and for possible inclusion in package reference manuals (in PDF format).

R 2.10.0 added a function `news()` (in package **utils**) to possibly extract the news for R or add-on packages into a simple (plain text) data base, and use this for simple queries. The data base is a character frame with variables `Version`, `Category`, `Date` and `Text`, where the last contains the entry texts read, and the other variables may be `NA` if they were missing or could not be determined. These variables can be used in the queries. E.g., we build a data base of all currently available R news entries:

```
> db <- news()
```

This has 2950 news entries, distributed according to version as follows:

```
> with(db, table(Version))

Version
         2.0.1  2.0.1 patched          2.1.0
            56             27            223
         2.1.1  2.1.1 patched         2.10.0
            56             31            153
       2.10.1 2.10.1 patched         2.11.0
            43             35            132
       2.11.1          2.2.0          2.2.1
            28            188             63
 2.2.1 patched          2.3.0          2.3.1
            18            250             38
 2.3.1 patched          2.4.0          2.4.1
            28            225             60
 2.4.1 patched          2.5.0          2.5.1
            15            208             50
 2.5.1 patched          2.6.0          2.6.1
            16            177             31
         2.6.2  2.6.2 patched          2.7.0
```

```
          53           14          186
       2.7.1        2.7.2  2.7.2 patched
          66           45            6
       2.8.0        2.8.1  2.8.1 patched
         132           59           27
       2.9.0        2.9.1        2.9.2
         124           56           26
2.9.2 patched
             5
```

(This shows "versions" such as '2.10.1 patched', which correspond to news items in the time interval between the last patchlevel release and the next minor or major releases of R (in the above, 2.10.1 and 2.11.0), respectively. These are not valid version numbers, and are treated as the corresponding patchlevel version in queries.) To extract all bug fixes since R 2.11.0 that include a PR number from a formal bug report, we can query the news db as follows:

```
> items <- news(Version >= "2.11.0" &
+               grepl("^BUG", Category) &
+               grepl("PR#", Text),
+               db = db)
```

This finds 22 such items, the first one being

```
> writeLines(strwrap(items[1, "Text"]))

The C function mkCharLenCE now no longer
reads past 'len' bytes (unlikely to be a
problem except in user code). (PR#14246)
```

Trying to extract the news for add-on packages involved a trial-and-error process to develop a reasonably reliable default reader for the 'NEWS' files in a large subset of available packages, and subsequently documenting the common format (requirements). This reader looks for version headers and splits the news file into respective chunks. It then tries to read the news items from the chunks, after possibly splitting according to category. For each chunk found, it records whether (it thinks) it successfully processed the chunk. To assess how well this reader actually works, we apply it to all 'NEWS' files in the CRAN packages. As of 2010-09-11, there are 427 such files for a "news coverage" of about 20%. For 67 files, no version chunks are found. For 40 files, all chunks are marked as bad (indicating that we found some version headers, but the files really use a different format). For 60 files, some chunks are bad, with the following summary of the frequencies of bad chunks:

```
   Min.  1st Qu.   Median     Mean  3rd Qu.
0.005747 0.057280 0.126800 0.236500 0.263800
   Max.
0.944400
```

Finally, for 260 files, no chunks are marked as bad, suggesting a success rate of about 61 percent. Given the lack of formal standardization, this is actually "not bad for a start", but certainly not good enough. Clearly, package maintainers could check readability for themselves and fix incompatibilities with the default format, or switch to it (or contribute readers for their own formats). But ideally, maintainers could employ a richer format allowing for more reliable computations and better processing.

In R 2.12.0, the new Rd format was enhanced in several important ways. The `\subsection` and `\newcommand` macros were added, to allow for subsections within sections, and user-defined macros. One can now convert R objects to fragments of Rd code, and process these. And finally, the rendering (of text in particular) is more customizable. Given the availability of `\subsection`, one can now conveniently map one- and two-layer hierarchies of item lists into sections (and subsections) of Rd `\itemize` lists. Altogether, the new Rd format obviously becomes a very attractive (some might claim, the canonical) format for maintaining R and package news information, in particular given that most R developers will be familiar with the Rd format and that Rd can be rendered as text, HTML (for on-line help) and LaTeX (for PDF manuals).

R 2.12.0 itself has switched to the Rd format for recording its news. To see the effect, run `help.start()`, click on the NEWS link and note that e.g. bug report numbers are now hyperlinked to the respective pages in the bug tracking system. The conversion was aided by the (internal) utility function `news2Rd()` in package **tools** which converted the legacy plain text 'NEWS' to Rd, but of course required additional effort enriching the markup.

R 2.12.0 is also aware of 'inst/NEWS.Rd' files in add-on packages, which are used by `news()` in preference to the plain text 'NEWS' and 'inst/NEWS' files. For 'NEWS' files in a format which can successfully be handled by the default reader, package maintainers can use `tools:::news2Rd(dir, "NEWS.Rd")`, possibly with additional argument `codify = TRUE`, with `dir` a character string specifying the path to a package's root directory. Upon success, the 'NEWS.Rd' file can be further improved and then be moved to the 'inst' subdirectory of the package source directory. For now, the immediate benefits of switching to the new format is that `news()` will work more reliably, and that the package news web pages on CRAN will look better.

As packages switch to Rd format for their news, additional services taking advantage of the new format can be developed and deployed. For the near future, we are planning to integrate package news into the on-line help system, and to optionally integrate the news when generating the package reference manuals. A longer term goal is to enhance the package management system to include news computations in `apt-listchanges` style. We hope that repository "news providers" will take advantage of the available infrastructure to enhance their news services (e.g., using news diffs instead of, or in ad-

dition to, the currently employed recursive diff summaries). But perhaps most importantly, we hope that the new format will make it more attractive for package maintainers to actually provide valuable news information.

Bibliography

Duncan Murdoch and Simon Urbanek. The new R help system. *The R Journal*, 1(2):60–65, 2009.

Achim Zeileis. CRAN task views. *R News*, 5(1):39–40, May 2005.

Kurt Hornik
Department of Finance, Accounting and Statistics
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

useR! 2010

by Katharine Mullen

Abstract A summary of the useR! 2010 conference.

The R user conference, useR! 2010, took place on the Gaithersburg, Maryland, USA campus of the National Institute of Standards and Technology (NIST) July 21-23 2010. Following the five previous useR! conferences (held in Austria, Germany, Iowa, and France), useR! 2010 focused on

- R as the 'lingua franca' of data analysis and statistical computing,
- providing a venue to discuss and exchange ideas on the use of R for statistical computations, data analysis, visualization, and exciting applications in various fields and
- providing an overview of the new features of the rapidly evolving R project.

The conference drew over 450 R users hailing from 29 countries. The technical program was composed of 167 contributed presentations, seven invited lectures, a poster session, and a panel discussion on 'Challenges of Bringing R into Commercial Environments'. The social program of the conference included several receptions and a dinner at the National Zoo.

Program, organizing and conference committees

Organization of the conference was thanks to individuals participating in the following committees:

Program Committee:
Louis Bajuk-Yorgan, Dirk Eddelbuettel, John Fox, Virgilio Gómez-Rubio, Richard Heiberger, Torsten Hothorn, Aaron King, Jan de Leeuw, Nicholas Lewin-Koh, Andy Liaw, Uwe Ligges, Martin Mächler, Katharine Mullen, Heather Turner, Ravi Varadhan, H. D. Vinod, John Verzani, Alan Zaslavsky, Achim Zeileis

Organizing Committee:
Kevin Coakley, Nathan Dodder, David Gil, William Guthrie, Olivia Lau, Walter Liggett, John Lu, Katharine Mullen, Jonathon Phillips, Antonio Possolo, Daniel Samarov, Ravi Varadhan

R Conference Committee:
Torsten Hothorn, Achim Zeileis

User-contributed presentations

The diversity of interests in the R community was reflected in the themes of the user-contributed sessions. The themes were:

- Bioinformatics workflows
- Biostatistics (three sessions)
- Biostatistics workflows
- Business intelligence
- Cloud computing
- Commercial applications (two sessions)
- Computer experiments and simulation
- Data manipulation and classification
- Data mining, machine learning
- Finance and resource allocation
- fMRI
- Fene expressions, genetics
- Grid computing
- Graphical User Interfaces (two sessions)
- High-performance-computing
- Interfaces
- Lists and objects
- Longitudinal data analysis
- MetRology
- Optimization
- Parallel computing
- Pedagogy (three sessions)
- Real-time computing
- Reproducible research and generating reports
- R social networks
- RuG panel discussion
- Social sciences
- Spatio-temporal data analysis
- Spreadsheets and RExcel
- Time series
- Visualization

In addition to sessions on these themes, research was presented in a poster session and in Kaleidescope sessions aimed at a broad audience.

Tutorials

The day before the official start of the conference, on July 20, the following nineteen 3-hour tutorials were given by R experts:

- **Douglas Bates:** Fitting and evaluating mixed models using lme4
- **Peter Danenberg and Manuel Eugster:** Literate programming with Roxygen
- **Karin Groothuis-Oudshoorn and Stef van Buuren:** Handling missing data in R with MICE

- **Frank Harrell Jr:** Statistical presentation graphics
- **François Husson and Julie Josse:** Exploratory data analysis with a special focus on clustering and multiway methods
- **Uwe Ligges:** My first R package
- **Daniel Samarov, Errol Strain and Elaine McVey:** R for Eclipse
- **Jing Hua Zhao:** Genetic analysis of complex traits
- **Alex Zolot:** Work with R on Amazon's Cloud
- **Karim Chine:** Elastic-R, a google docs-like portal for data analysis in the cloud
- **Dirk Eddelbuettel:** Introduction to high-performance computing with R
- **Michael Fay:** Interval censored data analysis
- **Virgilio Gómez-Rubio:** Applied spatial data analysis with R
- **Frank Harrell Jr:** Regression modeling strategies using the R package rms
- **Olivia Lau:** A crash course in R programming
- **Friedrich Leisch:** Sweave - Writing dynamic and reproducible documents
- **John Nash:** Optimization and related nonlinear modelling computations in R
- **Brandon Whitcher, Pierre Lafaye de Micheaux, Bradley Buchsbaum and Jörg Polzehl:** Medical image analysis for structural and functional MRI

Invited lectures

The distinguished invited lecturers were:

- **Mark S. Handcock:** Statistical Modeling of Networks in R
- **Frank E. Harrell Jr:** Information Allergy
- **Friedrich Leisch:** Reproducible Statistical Research in Practice
- **Uwe Ligges:** Prospects and Challenges for CRAN - with a glance on 64-bit Windows binaries
- **Richard M. Stallman:** Free Software in Ethics and in Practice
- **Luke Tierney:** Some possible directions for the R engine
- **Diethelm Würtz:** The Hull, the Feasible Set, and the Risk Surface: A Review of the Portfolio Modeling Infrastructure in R/Rmetrics

In addition, **Antonio Possolo** (Chief of the Statistical Engineering Division at NIST) began the conference with a rousing speech to welcome participants.

Conference-related information and thanks

The conference webpage http://www.R-project.org/useR-2010 makes available abstracts and slides associated with presentations, as well as links to video of plenary sessions. Questions regarding the conference may be addressed to useR-2010@R-project.org.

Many thanks to all those who contributed to useR! 2010. The talks, posters, ideas, and spirit of cooperation that R users from around the world brought to Gaithersburg made the conference a great success.

Katharine Mullen
Structure Determination Methods Group
Ceramics Division
National Institute of Standards and Technology (NIST)
100 Bureau Drive, M/S 8520
Gaithersburg, MD, 20899, USA
Email: Katharine.Mullen@nist.gov

Forthcoming Events: useR! 2011

The seventh international R user conference, *useR! 2011*, will take place at the University of Warwick, Coventry, 16–18 August 2011.

Following previous *useR!* conferences, this meeting of the R user community will provide a platform for R users to discuss and exchange ideas of how R can be used for statistical computation, data analysis and visualization.

The program will consist of invited talks, user-contributed sessions and pre-conference tutorials.

Invited Talks

The invited talks represent the spectrum of interest, ranging from important technical developments to exciting applications of R, presented by experts in the field:

- *Adrian Bowman*: Modelling Three-dimensional Surfaces in R.
- *Lee Edlefsen*: High Performance Computing in R.
- *Ulrike Grömping*: Design of Experiments in R.
- *Wolfgang Huber*: From Genomes to Phenotypes.
- *Brian Ripley*: The R Development Process.
- *Jonathan Rougier*: Calibration and Prediction for Stochastic Dynamical Systems Arising in Environmental Science.
- *Simon Urbanek*: R Graphics: Supercharged - Recent Advances in Visualization and Analysis of Large Data in R.
- *Brandon Whitcher*: Quantitative Analysis of Medical Imaging Data in R.

User-contributed Sessions

In the contributed sessions, presenters will share innovative and interesting uses of R, covering topics such as:

- Bayesian statistics
- Bioinformatics
- Chemometrics and computational physics
- Data mining
- Econometrics and finance
- Environmetrics and ecological modeling
- High performance computing
- Imaging
- Interfaces with other languages/software
- Machine learning
- Multivariate statistics
- Nonparametric statistics
- Pharmaceutical statistics
- Psychometrics
- Spatial statistics
- Statistics in the social and political sciences
- Teaching
- Visualization and graphics

Abstracts may be submitted for the poster session, which will be a major social event on the evening of the first day of the conference, or for the main program of talks. Submissions for contributed talks will be considered for the following types of session:

- *useR! Kaleidoscope*: These sessions give a broad overview of the many different applications of R and should appeal to a wide audience.
- *useR! Focus Session*: These sessions cover topics of special interest and may be more technical.

The scientific program is organized by members of the program committee, comprising Ramón Díaz-Uriarte, John Fox, Romain François, Robert Gramacy, Paul Hewson, Torsten Hothorn, Kate Mullen, Brian Peterson, Thomas Petzoldt, Anthony Rossini, Barry Rowlingson, Carolin Strobl, Stefan Theussl, Heather Turner, Hadley Wickham and Achim Zeileis.

In addition to the main program of talks and new for *useR! 2011*, all participants are invited to present a *Lightning Talk*, for which no abstract is required. These talks provide a 5-minute platform to speak on any R-related topic and should particularly appeal to those new to R. A variation of the pecha kucha[1] and ignite[2] formats will be used in which speakers must provide 15 slides to accompany their talk and each slide will be shown for 20 seconds.

Pre-conference Tutorials

Before the start of the official program, the following half-day tutorials will be offered on Monday, 15 August:

- *Douglas Bates*: Fitting and evaluating mixed models using **lme4**
- *Roger Bivand and Edzer Pebesma*: Handling and analyzing spatio-temporal data in R
- *Marine Cadoret and Sébastien Lê*: Analysing categorical data in R
- *Stephen Eglen*: Emacs Speaks Statistics
- *Andrea Foulkes*: High-dimensional data methods with R
- *Frank E Harrell Jr*: Regression modeling strategies using the R package **rms**
- *Søren Højsgaard*: Graphical models and Bayesian networks with R
- *G. Jay Kerns*: Introductory probability and statistics using R
- *Max Kuhn*: Predictive modeling with R and the **caret** package

[1] http://en.wikipedia.org/wiki/Pecha_Kucha
[2] http://en.wikipedia.org/wiki/Ignite_(event)

- *Nicholas Lewin Koh*: Everyday R: Statistical consulting with R
- *Fausto Molinari, Enrico Branca, Francesco De Filippo and Rocco Claudio Cannizzaro*: **R-Adamant**: Applied financial analysis and risk management
- *Martin Morgan*: Bioconductor for the analysis of high-throughput genomic data
- *Paul Murrell*: Introduction to grid graphics
- *Giovanni Petris*: State space models in R
- *Shivani Rao*: Topic modeling of large datasets with R using Amazon EMR
- *Karline Soetaert and Thomas Petzoldt*: Simulating differential equation models in R
- *Antony Unwin*: Graphical data analysis
- *Brandon Whitcher, Jörg Polzehl, Karsten Tabelow*: Medical image analysis for MRI

Location & Surroundings

Participants will be well catered for at the University of Warwick, with meals and accommodation available on campus. The university is in the city of Coventry which offers a number of places to visit such as the cathedral and city art gallery. In the heart of England, the city is within easy reach of other attractions such as Warwick's medieval castle and the new Royal Shakespeare and Swan Theatres in Stratford-upon-Avon (Shakespeare's birthplace).

Further Information

A web page offering more information on *useR! 2011*, including details regarding registration and abstract submission, is available at http://www.R-project.org/useR-2011/

We hope to meet you in Coventry!

The organizing committee:
John Aston, Julia Brettschneider, David Firth, Ashley Ford, Ioannis Kosmidis, Tom Nichols, Elke Thönnes and Heather Turner
useR-2011@R-project.org

Changes in R

From version 2.11.1 patched to version 2.12.1

by the R Core Team

CHANGES IN R VERSION 2.12.1

NEW FEATURES

- The DVI/PDF reference manual now includes the help pages for all the standard packages: **splines**, **stats4** and **tcltk** were previously omitted (intentionally).
- `http://www.rforge.net` has been added to the default set of repositories known to `setRepositories()`.
- xz-utils has been updated to version 5.0.0.
- `reshape()` now makes use of `sep` when forming names during reshaping to wide format. (PR#14435)
- `legend()` allows the length of lines to be set by the end user *via* the new argument `seg.len`.
- New reference class utility methods `copy()`, `field()`, `getRefClass()` and `getClass()` have been added.
- When a character value is used for the `EXPR` argument in `switch()`, a warning is given if more than one unnamed alternative value is given. This will become an error in R 2.13.0.
- `StructTS(type = "BSM")` now allows series with just two seasons. (Reported by Birgit Erni.)

INSTALLATION

- The PDF reference manual is now built as PDF version 1.5 with object compression, which on platforms for which this is not the default (notably MiKTeX) halves its size.
- Variable `FCLIBS` can be set during configuration, for any additional library flags needed when linking a shared object with the Fortran 9x compiler. (Needed with Solaris Studio 12.2.)

BUG FIXES

- `seq.int()` no longer sometimes evaluates arguments twice. (PR#14388)
- The `data.frame` method of `format()` failed if a column name was longer than 256 bytes (the maximum length allowed for an R name).
- `predict(<lm object>, type ="terms", ...)` failed if both `terms` and `interval` were specified. (Reported by Bill Dunlap.)

  Also, if `se.fit = TRUE` the standard errors were reported for all terms, not just those selected by a non-null `terms`.
- The TRE regular expressions engine could terminate R rather than give an error when given certain invalid regular expressions. (PR#14398)
- `cmdscale(eig = TRUE)` was documented to return $n - 1$ eigenvalues but in fact only returned `k`. It now returns all n eigenvalues.

  `cmdscale(add = TRUE)` failed to centre the return configuration and sometimes lost the labels on the points. Its return value was described wrongly (it is always a list and contains component `ac`).
- `promptClass()` in package **methods** now works for reference classes and gives a suitably specialized skeleton of documentation.

  Also, `callSuper()` now works via the `methods()` invocation as well as for initially specified methods.
- `download.file()` could leave the destination file open if the URL was not able to be opened. (PR#14414)
- Assignment of an environment to functions or as an attribute to other objects now works for S4 subclasses of `"environment"`.
- Use of '`[[<-`' for S4 subclasses of `"environment"` generated an infinite recursion from the method. The method has been replaced by internal code.
- In a reference class S4 method, `callSuper()` now works in `initialize()` methods when there is no explicit superclass method.
- '`!`' dropped attributes such as names and dimensions from a length-zero argument. (PR#14424)
- When `list2env()` created an environment it was missing a `PROTECT` call and so was vulnerable to garbage collection.
- `Sweave()` with `keep.source=TRUE` dropped comments at the start and end of code chunks. It could also fail when '`\SweaveInput`' was combined with named chunks.
- The Fortran code used by `nls(algorithm = "port")` could infinite-loop when compiled

with high optimization on a modern version of gcc, and SAFE_FFLAGS is now used to make this less likely. (PR#14427, seen with 32-bit Windows using gcc 4.5.0 used from R 2.12.0.)

- sapply() with default simplify = TRUE and mapply() with default SIMPLIFY = TRUE wrongly simplified language-like results, as, e.g., in mapply(1:2, c(3,7), FUN = function(i,j) call(':',i,j)).
- Backreferences to undefined patterns in [g]sub(pcre = TRUE) could cause a segfault. (PR#14431)
- The format() (and hence the print()) method for class "Date" rounded fractional dates towards zero: it now always rounds them down.
- Reference S4 class creation could generate ambiguous inheritance patterns under very special circumstances.
- '[[<-' turned S4 subclasses of "environment" into plain environments.
- Long titles for help pages were truncated in package indices and a few other places.
- Additional utilities now work correctly with S4 subclasses of "environment" (rm, locking tools and active bindings).
- spec.ar() now also work for the "ols" method. (Reported by Hans-Ruedi Kuensch.)
- The initialization of objects from S4 subclasses of "environment" now allocates a new environment object.
- R CMD check has more protection against (probably erroneous) example or test output which is invalid in the current locale.
- qr.X() with column names and pivoting now also pivots the column names. (PR#14438)
- unit.pmax() and unit.pmin() in package **grid** gave incorrect results when all inputs were of length 1. (PR#14443)
- The parser for 'NAMESPACE' files ignored misspelled directives, rather than signalling an error. For 2.12.x a warning will be issued, but this will be correctly reported as an error in later releases. (Reported by Charles Berry.)
- Fix for subsetting of "raster" objects when only one of i or j is specified.
- grid.raster() in package **grid** did not accept "nativeRaster" objects (like rasterImage() does).
- Rendering raster images in PDF output was resetting the clipping region.
- Rendering of raster images on Cairo X11 device was wrong, particularly when a small image was being scaled up using interpolation.

  With Cairo < 1.6, will be better than before, though still a little clunky. With Cairo >= 1.6, should be sweet as.
- Several bugs fixed in read.DIF(): single column inputs caused errors, cells marked as "character" could be converted to other types, and (in Windows) copying from the clipboard failed.

CHANGES IN R VERSION 2.12.0

NEW FEATURES

- Reading a package's 'CITATION' file now defaults to ASCII rather than Latin-1: a package with a non-ASCII 'CITATION' file should declare an encoding in its 'DESCRIPTION' file and use that encoding for the 'CITATION' file.
- difftime() now defaults to the "tzone" attribute of "POSIXlt" objects rather than to the current timezone as set by the default for the tz argument. (Wish of PR#14182.)
- pretty() is now generic, with new methods for "Date" and "POSIXt" classes (based on code contributed by Felix Andrews).
- unique() and match() are now faster on character vectors where all elements are in the global CHARSXP cache and have unmarked encoding (ASCII). Thanks to Matthew Dowle for suggesting improvements to the way the hash code is generated in 'unique.c'.
- The enquote() utility, in use internally, is exported now.
- .C() and .Fortran() now map non-zero return values (other than NA_LOGICAL) for logical vectors to TRUE: it has been an implicit assumption that they are treated as true.
- The print() methods for "glm" and "lm" objects now insert linebreaks in long calls in the same way that the print() methods for "summary.[g]lm" objects have long done. This does change the layout of the examples for a number of packages, e.g. **MASS**. (PR#14250)
- constrOptim() can now be used with method "SANN". (PR#14245)

  It gains an argument hessian to be passed to optim(), which allows all the ... arguments to be intended for f() and grad(). (PR#14071)

- `curve()` now allows `expr` to be an object of mode `"expression"` as well as `"call"` and `"function"`.

- The `"POSIX[cl]t"` methods for `Axis()` have been replaced by a single method for `"POSIXt"`.

  There are no longer separate `plot()` methods for `"POSIX[cl]t"` and `"Date"`: the default method has been able to handle those classes for a long time. This *inter alia* allows a single date-time object to be supplied, the wish of PR#14016.

  The methods had a different default (`""`) for `xlab`.

- Classes `"POSIXct"`, `"POSIXlt"` and `"difftime"` have generators `.POSIXct()`, `.POSIXlt()` and `.difftime()`. Package authors are advised to make use of them (they are available from R 2.11.0) to proof against planned future changes to the classes.

  The ordering of the classes has been changed, so `"POSIXt"` is now the second class. See the document 'Updating packages for changes in R 2.12.x' on `http://developer.r-project.org` for the consequences for a handful of CRAN packages.

- The `"POSIXct"` method of `as.Date()` allows a timezone to be specified (but still defaults to UTC).

- New `list2env()` utility function as an inverse of `as.list(<environment>)` and for fast multi-`assign()` to existing environment. `as.environment()` is now generic and uses `list2env()` as list method.

- There are several small changes to output which 'zap' small numbers, e.g. in printing quantiles of residuals in summaries from `"lm"` and `"glm"` fits, and in test statistics in `print.anova()`.

- Special names such as `"dim"`, `"names"`, etc, are now allowed as slot names of S4 classes, with `"class"` the only remaining exception.

- File '.Renviron' can have architecture-specific versions such as '.Renviron.i386' on systems with sub-architectures.

- `installed.packages()` has a new argument `subarch` to filter on sub-architecture.

- The `summary()` method for `packageStatus()` now has a separate `print()` method.

- The default `summary()` method returns an object inheriting from class `"summaryDefault"` which has a separate `print()` method that calls `zapsmall()` for numeric/complex values.

- The startup message now includes the platform and if used, sub-architecture: this is useful where different (sub-)architectures run on the same OS.

- The `getGraphicsEvent()` mechanism now allows multiple windows to return graphics events, through the new functions `setGraphicsEventHandlers()`, `setGraphicsEventEnv()`, and `getGraphicsEventEnv()`. (Currently implemented in the `windows()` and `X11()` devices.)

- `tools::texi2dvi()` gains an `index` argument, mainly for use by `R CMD Rd2pdf`.

  It avoids the use of `texindy` by `texinfo`'s `texi2dvi >= 1.157`, since that does not emulate 'makeindex' well enough to avoid problems with special characters (such as `'('`, `'{'`, `'!'`) in indices.

- The ability of `readLines()` and `scan()` to reencode inputs to marked UTF-8 strings on Windows since R 2.7.0 is extended to non-UTF-8 locales on other OSes.

- `scan()` gains a `fileEncoding` argument to match `read.table()`.

- `points()` and `lines()` gain `"table"` methods to match `plot()`. (Wish of PR#10472.)

- `Sys.chmod()` allows argument `mode` to be a vector, recycled along `paths`.

- There are `|`, `&` and `xor()` methods for classes `"octmode"` and `"hexmode"`, which work bitwise.

- Environment variables `R_DVIPSCMD`, `R_LATEXCMD`, `R_MAKEINDEXCMD`, `R_PDFLATEXCMD` are no longer used nor set in an R session. (With the move to `tools::texi2dvi()`, the conventional environment variables `LATEX`, `MAKEINDEX` and `PDFLATEX` will be used. `options("dvipscmd")` defaults to the value of `DVIPS`, then to `"dvips"`.)

- New function `isatty()` to see if terminal connections are redirected.

- `summaryRprof()` returns the sampling interval in component `sample.interval` and only returns in `by.self` data for functions with non-zero self times.

- `print(x)` and `str(x)` now indicate if an empty list `x` is named.

- `install.packages()` and `remove.packages()` with `lib` unspecified and multiple libraries in `.libPaths()` inform the user of the library location used with a message rather than a warning.

- There is limited support for multiple compressed streams on a file: all of `[bgx]zfile()` allow streams to be appended to an existing file, but `bzfile()` reads only the first stream.

- Function `person()` in package **utils** now uses a given/family scheme in preference to first/middle/last, is vectorized to handle an arbitrary number of persons, and gains a `role` argument to specify person roles using a controlled vocabulary (the MARC relator terms).

- Package **utils** adds a new `"bibentry"` class for representing and manipulating bibliographic information in enhanced BibTeX style, unifying and enhancing the previously existing mechanisms.

- A `bibstyle()` function has been added to the **tools** package with default JSS style for rendering `"bibentry"` objects, and a mechanism for registering other rendering styles.

- Several aspects of the display of text help are now customizable using the new `Rd2txt_options()` function. options("help_text_width") is no longer used.

- Added '`\href`' tag to the Rd format, to allow hyperlinks to URLs without displaying the full URL.

- Added '`\newcommand`' and '`\renewcommand`' tags to the Rd format, to allow user-defined macros.

- New `toRd()` generic in the **tools** package to convert objects to fragments of Rd code, and added `"fragment"` argument to `Rd2txt()`, `Rd2HTML()`, and `Rd2latex()` to support it.

- Directory 'R_HOME/share/texmf' now follows the TDS conventions, so can be set as a `texmf` tree ('root directory' in MiKTeX parlance).

- S3 generic functions now use correct S4 inheritance when dispatching on an S4 object. See `?Methods`, section on "Methods for S3 Generic Functions" for recommendations and details.

- `format.pval()` gains a `...` argument to pass arguments such as `nsmall` to `format()`. (Wish of PR#9574)

- `legend()` supports `title.adj`. (Wish of PR#13415)

- Added support for subsetting `"raster"` objects, plus assigning to a subset, conversion to a matrix (of colour strings), and comparisons ('`==`' and '`!=`').

- Added a new `parseLatex()` function (and related functions `deparseLatex()` and `latexToUtf8()`) to support conversion of bibliographic entries for display in R.

- Text rendering of '`\itemize`' in help uses a Unicode bullet in UTF-8 and most single-byte Windows locales.

- Added support for polygons with holes to the graphics engine. This is implemented for the `pdf()`, `postscript()`, `x11(type="cairo")`, `windows()`, and `quartz()` devices (and associated raster formats), but not for `x11(type="Xlib")` or `xfig()` or `pictex()`. The user-level interface is the `polypath()` function in **graphics** and `grid.path()` in **grid**.

- File 'NEWS' is now generated at installation with a slightly different format: it will be in UTF-8 on platforms using UTF-8, and otherwise in ASCII. There is also a PDF version, 'NEWS.pdf', installed at the top-level of the R distribution.

- `kmeans(x, 1)` now works. Further, `kmeans` now returns between and total sum of squares.

- `arrayInd()` and `which()` gain an argument `useNames`. For `arrayInd`, the default is now false, for speed reasons.

- As is done for closures, the default `print` method for the formula class now displays the associated environment if it is not the global environment.

- A new facility has been added for inserting code into a package without re-installing it, to facilitate testing changes which can be selectively added and backed out. See `?insertSource`.

- New function `readRenviron` to (re-)read files in the format of '~/.Renviron' and 'Renviron.site'.

- `require()` will now return `FALSE` (and not fail) if loading the package or one of its dependencies fails.

- `aperm()` now allows argument `perm` to be a character vector when the array has named dimnames (as the results of `table()` calls do). Similarly, `array()` allows `MARGIN` to be a character vector. (Based on suggestions of Michael Lachmann.)

- Package **utils** now exports and documents functions `aspell_package_Rd_files()` and `aspell_package_vignettes()` for spell checking package Rd files and vignettes using Aspell, Ispell or Hunspell.

- Package news can now be given in Rd format, and `news()` prefers these 'inst/NEWS.Rd' files to old-style plain text 'NEWS' or 'inst/NEWS' files.
- New simple function `packageVersion()`.
- The PCRE library has been updated to version 8.10.
- The standard Unix-alike terminal interface declares its name to readline as 'R', so that can be used for conditional sections in '~/.inputrc' files.
- 'Writing R Extensions' now stresses that the standard sections in '.Rd' files (other than '`\alias`', '`\keyword`' and '`\note`') are intended to be unique, and the conversion tools now drop duplicates with a warning.

  The '.Rd' conversion tools also warn about an unrecognized type in a '`\docType`' section.
- `ecdf()` objects now have a `quantile()` method.
- `format()` methods for date-time objects now attempt to make use of a `"tzone"` attribute with `"%Z"` and `"%z"` formats, but it is not always possible. (Wish of PR#14358.)
- `tools::texi2dvi(file, clean = TRUE)` now works in more cases (e.g. where emulation is used and when 'file' is not in the current directory).
- New function `droplevels()` to remove unused factor levels.
- `system(command, intern = TRUE)` now gives an error on a Unix-alike (as well as on Windows) if `command` cannot be run. It reports a non-success exit status from running `command` as a warning.

  On a Unix-alike an attempt is made to return the actual exit status of the command in `system(intern = FALSE)`: previously this had been system-dependent but on POSIX-compliant systems the value return was 256 times the status.
- `system()` has a new argument `ignore.stdout` which can be used to (portably) ignore standard output.
- `system(intern = TRUE)` and `pipe()` connections are guaranteed to be available on all builds of R.
- `Sys.which()` has been altered to return `""` if the command is not found (even on Solaris).
- A facility for defining reference-based S4 classes (in the OOP style of Java, C++, etc.) has been added experimentally to package **methods**; see `?ReferenceClasses`.
- The `predict` method for `"loess"` fits gains an `na.action` argument which defaults to `na.pass` rather than the previous default of `na.omit`.

  Predictions from `"loess"` fits are now named from the row names of `newdata`.
- Parsing errors detected during `Sweave()` processing will now be reported referencing their original location in the source file.
- New `adjustcolor()` utility, e.g., for simple translucent color schemes.
- `qr()` now has a trivial `lm` method with a simple (fast) validity check.
- An experimental new programming model has been added to package **methods** for reference (OOP-style) classes and methods. See `?ReferenceClasses`.
- bzip2 has been updated to version 1.0.6 (bugfix release). '`--with-system-bzlib`' now requires at least version 1.0.6.
- R now provides 'jss.cls' and 'jss.bst' (the class and bib style file for the Journal of Statistical Software) as well as 'RJournal.bib' and 'Rnews.bib', and `R CMD` ensures that the '.bst' and '.bib' files are found by BibTeX.
- Functions using the `TAR` environment variable no longer quote the value when making `system` calls. This allows values such as '`tar --force-local`', but does require additional quotes in, e.g., `TAR = "'/path with spaces/mytar'"`.

DEPRECATED & DEFUNCT

- Supplying the parser with a character string containing both octal/hex and Unicode escapes is now an error.
- File extension '.C' for C++ code files in packages is now defunct.
- `R CMD check` no longer supports configuration files containing Perl configuration variables: use the environment variables documented in 'R Internals' instead.
- The `save` argument of `require()` now defaults to `FALSE` and `save = TRUE` is now deprecated. (This facility is very rarely actually used, and was superseded by the '`Depends`' field of the 'DESCRIPTION' file long ago.)
- `R CMD check -no-latex` is deprecated in favour of '`--no-manual`'.
- `R CMD Sd2Rd` is formally deprecated and will be removed in R 2.13.0.

PACKAGE INSTALLATION

- `install.packages()` has a new argument `libs_only` to optionally pass '`--libs-only`' to `R CMD INSTALL` and works analogously for Windows binary installs (to add support for 64- or 32-bit Windows).

- When sub-architectures are in use, the installed architectures are recorded in the `Archs` field of the 'DESCRIPTION' file. There is a new default filter, `"subarch"`, in `available.packages()` to make use of this.

  Code is compiled in a copy of the 'src' directory when a package is installed for more than one sub-architecture: this avoid problems with cleaning the sources between building sub-architectures.

- `R CMD INSTALL -libs-only` no longer overrides the setting of locking, so a previous version of the package will be restored unless '`--no-lock`' is specified.

UTILITIES

- `R CMD Rprof|build|check` are now based on R rather than Perl scripts. The only remaining Perl scripts are the deprecated `R CMD Sd2Rd` and `install-info.pl` (used only if `install-info` is not found) as well as some maintainer-mode-only scripts.

  NB: because these have been completely rewritten, users should not expect undocumented details of previous implementations to have been duplicated.

  `R CMD` no longer manipulates the environment variables `PERL5LIB` and `PERLLIB`.

- `R CMD check` has a new argument '`--extra-arch`' to confine tests to those needed to check an additional sub-architecture.

  Its check for "Subdirectory 'inst' contains no files" is more thorough: it looks for files, and warns if there are only empty directories.

  Environment variables such as `R_LIBS` and those used for customization can be set for the duration of checking *via* a file '~/.R/check.Renviron' (in the format used by '.Renviron', and with sub-architecture specific versions such as '~/.R/check.Renviron.i386' taking precedence).

  There are new options '`--multiarch`' to check the package under all of the installed sub-architectures and '`--no-multiarch`' to confine checking to the sub-architecture under which `check` is invoked. If neither option is supplied, a test is done of installed sub-architectures and all those which can be run on the current OS are used.

  Unless multiple sub-architectures are selected, the install done by `check` for testing purposes is only of the current sub-architecture (*via* `R CMD INSTALL -no-multiarch`).

  It will skip the check for non-ascii characters in code or data if the environment variables `_R_CHECK_ASCII_CODE_` or `_R_CHECK_ASCII_DATA_` are respectively set to `FALSE`. (Suggestion of Vince Carey.)

- `R CMD build` no longer creates an 'INDEX' file (`R CMD INSTALL` does so), and `-force` removes (rather than overwrites) an existing 'INDEX' file.

  It supports a file '~/.R/build.Renviron' analogously to `check`.

  It now runs build-time '`\Sexpr`' expressions in help files.

- `R CMD Rd2dvi` makes use of `tools::texi2dvi()` to process the package manual. It is now implemented entirely in R (rather than partially as a shell script).

- `R CMD Rprof` now uses `utils::summaryRprof()` rather than Perl. It has new arguments to select one of the tables and to limit the number of entries printed.

- `R CMD Sweave` now runs R with `-vanilla` so the environment setting of `R_LIBS` will always be used.

C-LEVEL FACILITIES

- `lang5()` and `lang6()` (in addition to pre-existing `lang[1-4]()`) convenience functions for easier construction of `eval()` calls. If you have your own definition, do wrap it inside `#ifndef lang5 .... #endif` to keep it working with old and new R.

- Header 'R.h' now includes only the C headers it itself needs, hence no longer includes `errno.h`. (This helps avoid problems when it is included from C++ source files.)

- Headers 'Rinternals.h' and 'R_ext/Print.h' include the C++ versions of 'stdio.h' and 'stdarg.h' respectively if included from a C++ source file.

INSTALLATION

- A C99 compiler is now required, and more C99 language features will be used in the R sources.

- Tcl/Tk >= 8.4 is now required (increased from 8.3).

- System functions `access`, `chdir` and `getcwd` are now essential to configure R. (In practice they have been required for some time.)

- `make check` compares the output of the examples from several of the base packages to reference output rather than the previous output (if any). Expect some differences due to differences in floating-point computations between platforms.

- File 'NEWS' is no longer in the sources, but generated as part of the installation. The primary source for changes is now 'doc/NEWS.Rd'.

- The `popen` system call is now required to build R. This ensures the availability of `system(intern = TRUE)`, `pipe()` connections and printing from `postscript()`.

- The `pkg-config` file 'libR.pc' now also works when R is installed using a sub-architecture.

- R has always required a BLAS that conforms to IE60559 arithmetic, but after discovery of more real-world problems caused by a BLAS that did not, this is tested more thoroughly in this version.

BUG FIXES

- Calls to `selectMethod()` by default no longer cache inherited methods. This could previously corrupt methods used by `as()`.

- The densities of non-central chi-squared are now more accurate in some cases in the extreme tails, e.g. `dchisq(2000, 2, 1000)`, as a series expansion was truncated too early. (PR#14105)

- `pt()` is more accurate in the left tail for `ncp` large, e.g. `pt(-1000, 3, 200)`. (PR#14069)

- The default C function (`R_binary`) for binary ops now sets the S4 bit in the result if either argument is an S4 object. (PR#13209)

- `source(echo=TRUE)` failed to echo comments that followed the last statement in a file.

- S4 classes that contained one of `"matrix"`, `"array"` or `"ts"` and also another class now accept superclass objects in `new()`. Also fixes failure to call `validObject()` for these classes.

- Conditional inheritance defined by argument `test` in `methods::setIs()` will no longer be used in S4 method selection (caching these methods could give incorrect results). See `?setIs`.

- The signature of an implicit generic is now used by `setGeneric()` when that does not use a definition nor explicitly set a signature.

- A bug in `callNextMethod()` for some examples with `"..."` in the arguments has been fixed. See file 'src/library/methods/tests/nextWithDots.R' in the sources.

- `match(x, table)` (and hence `%in%`) now treat `"POSIXlt"` consistently with, e.g., `"POSIXct"`.

- Built-in code dealing with environments (`get()`, `assign()`, `parent.env()`, `is.environment()` and others) now behave consistently to recognize S4 subclasses; `is.name()` also recognizes subclasses.

- The `abs.tol` control parameter to `nlminb()` now defaults to `0.0` to avoid false declarations of convergence in objective functions that may go negative.

- The standard Unix-alike termination dialog to ask whether to save the workspace takes a EOF response as `n` to avoid problems with a damaged terminal connection. (PR#14332)

- Added `warn.unused` argument to `hist.default()` to allow suppression of spurious warnings about graphical parameters used with `plot=FALSE`. (PR#14341)

- `predict.lm()`, `summary.lm()`, and indeed `lm()` itself had issues with residual DF in zero-weighted cases (the latter two only in connection with empty models). (Thanks to Bill Dunlap for spotting the `predict()` case.)

- `aperm()` treated `resize = NA` as `resize = TRUE`.

- `constrOptim()` now has an improved convergence criterion, notably for cases where the minimum was (very close to) zero; further, other tweaks inspired from code proposals by Ravi Varadhan.

- Rendering of S3 and S4 methods in man pages has been corrected and made consistent across output formats.

- Simple markup is now allowed in '`\title`' sections in '.Rd' files.

- The behaviour of `as.logical()` on factors (to use the levels) was lost in R 2.6.0 and has been restored.

- `prompt()` did not backquote some default arguments in the '`\usage`' section. (Reported by Claudia Beleites.)

- `writeBin()` disallows attempts to write 2GB or more in a single call. (PR#14362)

- `new()` and `getClass()` will now work if Class is a subclass of `"classRepresentation"` and should also be faster in typical calls.
- The `summary()` method for data frames makes a better job of names containing characters invalid in the current locale.
- `[[` sub-assignment for factors could create an invalid factor (reported by Bill Dunlap).
- `Negate(f)` would not evaluate argument `f` until first use of returned function (reported by Olaf Mersmann).
- `quietly=FALSE` is now also an optional argument of `library()`, and consequently, `quietly` is now propagated also for loading dependent packages, e.g., in `require(*, quietly=TRUE)`.
- If the loop variable in a `for` loop was deleted, it would be recreated as a global variable. (Reported by Radford Neal; the fix includes his optimizations as well.)
- Task callbacks could report the wrong expression when the task involved parsing new code. (PR#14368)
- `getNamespaceVersion()` failed; this was an accidental change in 2.11.0. (PR#14374)
- `identical()` returned `FALSE` for external pointer objects even when the pointer addresses were the same.
- `L$a@x[] <- val` did not duplicate in a case it should have.
- `tempfile()` now always gives a random file name (even if the directory is specified) when called directly after startup and before the R RNG had been used. (PR#14381)
- `quantile(type=6)` behaved inconsistently. (PR#14383)
- `backSpline(.)` behaved incorrectly when the knot sequence was decreasing. (PR#14386)
- The reference BLAS included in R was assuming that `0*x` and `x*0` were always zero (whereas they could be `NA` or `NaN` in IEC 60559 arithmetic). This was seen in results from `tcrossprod`, and for example that `log(0) %*% 0` gave `0`.
- The calculation of whether text was completely outside the device region (in which case, you draw nothing) was wrong for screen devices (which have [0, 0] at top-left). The symptom was (long) text disappearing when resizing a screen window (to make it smaller). (PR#14391)
- `model.frame(drop.unused.levels = TRUE)` did not take into account `NA` values of factors when deciding to drop levels. (PR#14393)
- `library.dynam.unload` required an absolute path for `libpath`. (PR#14385)

  Both `library()` and `loadNamespace()` now record absolute paths for use by `searchpaths()` and `getNamespaceInfo(ns, "path")`.
- The self-starting model `NLSstClosestX` failed if some deviation was exactly zero. (PR#14384)
- `X11(type = "cairo")` (and other devices such as `png` using cairographics) and which use Pango font selection now work around a bug in Pango when very small fonts (those with sizes between 0 and 1 in Pango's internal units) are requested. (PR#14369)
- Added workaround for the font problem with `X11(type = "cairo")` and similar on Mac OS X whereby italic and bold styles were interchanged. (PR#13463 amongst many other reports.)
- `source(chdir = TRUE)` failed to reset the working directory if it could not be determined – that is now an error.
- Fix for crash of `example(rasterImage)` on `x11(type="Xlib")`.
- Force Quartz to bring the on-screen display up-to-date immediately before the snapshot is taken by `grid.cap()` in the Cocoa implementation. (PR#14260)
- `model.frame` had an unstated 500 byte limit on variable names. (Example reported by Terry Therneau.)
- The 256-byte limit on names is now documented.
- Subassignment by `[`, `[[` or `$` on an expression object with value `NULL` coerced the object to a list.

CHANGES IN R VERSION 2.11.1 patched

NEW FEATURES

- `install.packages()` has a new optional argument `INSTALL_opts` which can be used to pass options to `R CMD INSTALL` for source-package installs.
- `R CMD check` now runs the package-specific tests with `LANGUAGE=en` to facilitate comparison to '.Rout.save' files.

- `sessionInfo()` gives more detailed platform information, including 32/64-bit and the sub-architecture if one is used.

DEPRECATED & DEFUNCT

- The use of Perl configuration variables for `R CMD check` (as previously documented in 'Writing R Extensions') is deprecated and will be removed in R 2.12.0. Use the environment variables documented in 'R Internals' instead.

BUG FIXES

- `R CMD Rd2dvi` failed if run from a path containing space(s). This also affected `R CMD check`, which calls `Rd2dvi`.
- `stripchart()` could fail with an empty factor level. (PR#14317)
- Text help rendering of '`\tabular{}`' has been improved: under some circumstances leading blank columns were not rendered.
- `strsplit(x, fixed=TRUE)` marked UTF-8 strings with the local encoding when no splits were found.
- `weighted.mean(NA, na.rm=TRUE)` and similar now returns `NaN` again, as it did prior to R 2.10.0.
- `R CMD` had a typo in its detection of whether the environment variable `TEXINPUTS` was set (reported by Martin Morgan).
- The command-line parser could mistake '`--file=size...`' for one of the options for setting limits for Ncells or Vcells.
- The internal `strptime()` could corrupt its copy of the timezone which would then lead to spurious warnings. (PR#14338)
- `dir.create(recursive = TRUE)` could fail if one of the components existed but was a directory on a read-only file system. (Seen on Solaris, where the error code returned is not even listed as possible on the man page.)
- The `postscript()` and `pdf()` devices will now allow `lwd` values less than 1 (they used to force such values to be 1).
- Fixed font face for CID fonts in `pdf()` graphics output. (PR#14326)
- `GERaster()` now checks for width or height of zero and does nothing in those cases; previously the behaviour was undefined, probably device-specific, and possibly dangerous.
- `wilcox.test(x, y, conf.int = TRUE)` failed with an unhelpful message if `x` and `y` were constant vectors, and similarly in the one-sample case. (PR#14329)
- Improperly calling `Recall()` from outside a function could cause a segfault. (Reported by Robert McGehee.)
- '`\Sexpr[result=rd]`' in an Rd file added a spurious newline, which was displayed as extra whitespace when rendered.
- `require(save = TRUE)` recorded the names of packages it failed to load.
- `packageStatus()` could return a data frame with duplicate row names which could then not be printed.
- `txtProgressBar(style = 2)` did not work correctly.

  `txtProgressBar(style = 3)` did not display until a non-minimum value was set.
- `contour()` did not display dashed line types properly when contour lines were labelled. (Reported by David B. Thompson.)
- `tools::undoc()` again detects undocumented data objects. Of course, this also affects `R CMD check`.
- `ksmooth(x,NULL)` no longer segfaults.
- `approxfun()`, `approx()`, `splinefun()` and `spline()` could be confused by `x` values that were different but so close as to print identically. (PR#14377)

Changes on CRAN

2010-06-04 to 2010-12-12

by Kurt Hornik and Achim Zeileis

New CRAN task views

OfficialStatistics Topic: Official Statistics & Survey Methodology. Maintainer: Matthias Templ. Packages: **Amelia**, **EVER**, **Hmisc**, **MImix**, **RecordLinkage**, **SDaA**, **SeqKnn**, **StatMatch**, **TeachingSampling**, **VIM**, **cat**, **impute**, **ineq**, **laeken**, **lme4**, **memisc**, **mi**, **micEcon**, **mice**, **missMDA**, **mitools**, **mix**, **nlme**, **norm**, **odfWeave.survey**, **pan**, **pps**, **reweight**, **robCompositions**, **rrcovNA**, **sampfling**, **sampling**, **samplingbook**, **sdcMicro**, **sdcTable**, **simFrame**, **simPopulation**, **spsurvey**, **stratification**, **survey***, **surveyNG**, **urn**, **x12**, **yaImpute**.

ReproducibleResearch Topic: Reproducible Research. Maintainer: Max Kuhn. Packages: **Design***, **Hmisc***, **R.cache**, **R.rsp**, **R2HTML***, **R2PPT**, **R2wd**, **SRPM**, **SweaveListingUtils**, **TeachingSampling**, **animation**, **apsrtable**, **ascii**, **brew**, **cacheSweave**, **cxxPack**, **exams**, **highlight**, **hwriter**, **memisc**, **odfWeave**, **odfWeave.survey**, **pgfSweave**, **quantreg**, **r2lh**, **reporttools**, **rms***, **svSweave**, **tikzDevice**, **xtable***.

(* = core package)

New packages in CRAN task views

Bayesian **Ratings**, **RxCEcolInf**, **SimpleTable**, **spikeslab**.

ChemPhys **OrgMassSpecR**, **plsRglm***, **solaR**.

ClinicalTrials **DoseFinding**, **MCPMod**.

Distributions **mc2d**, **nacopula**, **truncnorm**.

Environmetrics **DSpat**, **SPACECAP**, **secr**.

ExperimentalDesign **DoseFinding**.

Finance **orderbook**, **rrv**, **schwartz97**.

HighPerformanceComputing **gcbd**, **magma**, **rpud**.

MachineLearning **Boruta**, **LiblineaR**, **LogicForest**, **LogicReg**, **RSNNS**, **SDDA**, **hda**, **quantregForest**, **rda**, **rgp**, **sda**.

MedicalImaging **RNiftyReg***, **dpmixsim***, **mritc***.

Optimization **Rcgmin**, **Rsolnp***, **Rvmmin**, **minqa***, **optimx***.

Psychometrics **QuantPsyc**, **REQS**, **catR**, **equate**, **plsRglm**.

Spatial **CompRandFld**, **MarkedPointProcess**, **constrainedKriging**, **divagis**, **geosphere**, **raster***, **sparr**, **sphet**, **vardiag**.

Survival **aster**, **aster2**, **censReg**, **exactRankTests**, **glrt**, **saws**, **simexaft**, **spef**, **splinesurv**, **survPresmooth**.

TimeSeries **MARSS**, **TSAgg**, **mondate**, **pomp**.

gR **catnet**.

(* = core package)

New contributed packages

ACCLMA ACC & LMA Graph Plotting. Authors: Tal Carmi, Liat Gaziel.

AMA Anderson-Moore Algorithm. Authors: Gary Anderson, Aneesh Raghunandan.

AllPossibleSpellings Computes all of a word's possible spellings. Author: Antoine Tremblay, IWK Health Center.

BMS Bayesian Model Averaging Library. Authors: Martin Feldkircher and Stefan Zeugner. In view: *Bayesian*.

BayHap Bayesian analysis of haplotype association using Markov Chain Monte Carlo. Authors: Raquel Iniesta and Victor Moreno.

Benchmarking Benchmark and frontier analysis using DEA and SFA (BeDS). Authors: Peter Bogetoft and Lars Otto.

BlakerCI Blaker's binomial confidence limits. Author: J. Klaschka.

CFL Compensatory Fuzzy Logic. Authors: Pablo Michel Marin Ortega, Kornelius Rohmeyer.

CNVassoc Association analysis of CNV data. Authors: Juan R González, Isaac Subirana.

COSINE COndition SpecIfic sub-NEtwork. Author: Haisu Ma.

COUNT Functions, data and code for count data. Author: Joseph M Hilbe.

Ckmeans.1d.dp Optimal distance-based clustering for one-dimensional data. Authors: Joe Song and Haizhou Wang.

ClustOfVar Clustering of variables. Authors: Marie Chavent and Vanessa Kuentz and Benoit Liquet and Jerome Saracco.

CompQuadForm Distribution function of quadratic forms in normal variables. Author: P. Lafaye de Micheaux.

CompRandFld Composite-likelihood based Analysis of Random Fields. Authors: Simone Padoan, Moreno Bevilacqua. In view: *Spatial*.

DCGL Differential Coexpression Analysis of Gene Expression Microarray Data. Authors: Bao-Hong Liu, Hui Yu.

DECIDE DEComposition of Indirect and Direct Effects. Author: Christiana Kartsonaki.

DMwR Functions and data for "Data Mining with R". Author: Luis Torgo.

DNAtools Tools for analysing forensic genetic DNA data. Authors: Torben Tvedebrink and James Curran.

DeducerExtras Additional dialogs and functions for **Deducer**. Author: Ian Fellows.

DeducerPlugInScaling Reliability and factor analysis plugin. Authors: Alberto Mirisola and Ian Fellows.

DiceView Plot methods for computer experiments design and models. Authors: Yann Richet, Yves Deville, Clement Chevalier.

EMA Easy Microarray data Analysis. Author: EMA group.

EquiNorm Normalize expression data using equivalently expressed genes. Authors: Li-Xuan Qin, Jaya M. Satagopan.

FAwR Functions and Datasets for "Forest Analytics with R". Authors: Andrew Robinson and Jeff Hamann.

FeaLect Scores features for Feature seLection. Author: Habil Zare.

FitARMA FitARMA: Fit ARMA or ARIMA using fast MLE algorithm. Author: A.I. McLeod.

FourierDescriptors Generate images using Fourier descriptors. Author: John Myles White.

GAD General ANOVA Design (GAD): Analysis of variance from general principles. Author: Leonardo Sandrini-Neto & Mauricio G. Camargo.

GPseq Using the generalized Poisson distribution to model sequence read counts from high throughput sequencing experiments. Authors: Sudeep Srivastava, Liang Chen.

GWASExactHW Exact Hardy-Weinburg testing for Genome Wide Association Studies. Author: Ian Painter, GENEVA project, University of Washington.

HDclassif High Dimensionnal Classification. Authors: R. Aidan, L. Berge, C. Bouveyron, S. Girard.

HWEintrinsic Objective Bayesian Testing for the Hardy-Weinberg Equilibrium Problem. Author: Sergio Venturini.

HapEstXXR Haplotype-based analysis of association for genetic traits. Authors: Sven Knueppel and Klaus Rohde.

HumMeth27QCReport Quality control and preprocessing of Illumina's Infinium HumanMethylation27 BeadChip methylation assay. Author: F.M. Mancuso.

IMIS Incremental Mixture Importance Sampling. Authors: Adrian Raftery, Le Bao.

ImpactIV Identifying Causal Effect for Multi-Component Intervention Using Instrumental Variable Method. Author: Peng Ding.

IniStatR Initiation à la Statistique avec R. Authors: Frederic Bertrand, Myriam Maumy-Bertrand.

LMERConvenienceFunctions An assortment of functions to facilitate modeling with linear mixed-effects regression (LMER). Author: Antoine Tremblay.

LPCM Local principal curve methods. Authors: Jochen Einbeck and Ludger Evers.

LSD Lots of Superior Depictions. Authors: Bjoern Schwalb, Achim Tresch, Romain Francois.

LVQTools Learning Vector Quantization Tools. Author: Sander Kelders.

LogicForest Logic Forest. Author: Bethany Wolf. In view: *MachineLearning*.

MARSS Multivariate Autoregressive State-Space Modeling. Authors: Eli Holmes, Eric Ward, and Kellie Wills, NOAA, Seattle, USA. In view: *TimeSeries*.

MCLIME Simultaneous Estimation of the Regression Coefficients and Precision Matrix. Authors: T. Tony Cai, Hongzhe Li, Weidong Liu and Jichun Xie.

MMST Datasets from "Modern Multivariate Statistical Techniques" by Alan Julian Izenman. Author: Keith Halbert.

MOCCA Multi-objective optimization for collecting cluster alternatives. Author: Johann Kraus.

MSToolkit The MSToolkit library for clinical trial design. Authors: Mango Solutions & Pfizer.

MatrixModels Modelling with Sparse And Dense Matrices. Authors: Douglas Bates and Martin Maechler.

MeDiChI MeDiChI ChIP-chip deconvolution library. Author: David J Reiss.

MetabolAnalyze Probabilistic latent variable models for metabolomic data. Authors: Nyamundanda Gift, Isobel Claire Gormley and Lorraine Brennan.

Modalclust Hierarchical Modal Clustering. Authors: Surajit Ray and Yansong Cheng.

MsatAllele Visualizes the scoring and binning of microsatellite fragment sizes. Author: Filipe Alberto.

NCBI2R Navigate and annotate genes and SNPs. Author: Scott Melville.

NetCluster Clustering for networks. Authors: Mike Nowak, Solomon Messing, Sean J Westwood, and Dan McFarland.

NetData Network Data for McFarland's SNA R labs. Authors: Mike Nowak, Sean J Westwood, Solomon Messing, and Dan McFarland.

NetworkAnalysis Statistical inference on populations of weighted or unweighted networks. Author: Cedric E Ginestet.

ORDER2PARENT Estimate parent distributions with data of several order statistics. Author: Cheng Chou.

OSACC Ordered Subset Analysis for Case-Control Studies. Authors: Xuejun Qin, Elizabeth R. Hauser, Silke Schmidt (Center for Human Genetics, Duke University).

OrgMassSpecR Organic Mass Spectrometry. Authors: Nathan G. Dodder, Southern California Coastal Water Research Project (SCCWRP). Contributions from Katharine M. Mullen. In view: *ChemPhys*.

PERregress Regression Functions and Datasets. Author: Peter Rossi.

PKreport A reporting pipeline for checking population pharmacokinetic model assumption. Author: Xiaoyong Sun.

PermAlgo Permutational algorithm to simulate survival data. Authors: Marie-Pierre Sylvestre, Thad Edens, Todd MacKenzie, Michal Abrahamowicz.

PhysicalActivity Process Physical Activity Accelerometer Data. Authors: Leena Choi, Zhouwen Liu, Charles E. Matthews, and Maciej S. Buchowski.

PoMoS Polynomial (ordinary differential equation) Model Search. Authors: Mangiarotti S., Coudret R., Drapeau L.

ProjectTemplate Automates the creation of new statistical analysis projects. Author: John Myles White.

QSARdata Quantitative Structure Activity Relationship (QSAR) Data Sets. Author: Max Kuhn.

QuACN Quantitative Analysis of Complex Networks. Author: Laurin Mueller.

R4dfp 4dfp MRI Image Read and Write Routines. Author: Kevin P. Barry with contributions from Avi Z. Snyder.

RBrownie Continuous and discrete ancestral character reconstruction and evolutionary rate tests. Authors: J. Conrad Stack, Brian O'Meara, Luke Harmon.

RGCCA Regularized Generalized Canonical Correlation Analysis. Author: Arthur Tenenhaus.

RGtk2Extras Data frame editor and dialog making wrapper for **RGtk2**. Authors: Tom Taverner, with contributions from John Verzani and Iago Conde.

RJSONIO Serialize R objects to JSON, JavaScript Object Notation. Author: Duncan Temple Lang.

RMC Functions for fitting Markov models. Author: Scott D. Foster.

RNCBI The java ncbi interface to R. Author: Martin Schumann.

RNCBIAxis2Libs Axis2 libraries for use in the R environment. Author: Martin Schumann.

RNCBIEUtilsLibs EUtils libraries for use in the R environment. Author: Martin Schumann.

RNiftyReg Medical image registration using the NiftyReg library. Authors: Jon Clayden; based on original code by Marc Modat and Pankaj Daga. In view: *MedicalImaging*.

RSNNS Neural Networks in R using the Stuttgart Neural Network Simulator (SNNS). Author: Christoph Bergmeir. In view: *MachineLearning*.

RSearchYJ Search with Yahoo Japan. Author: Yohei Sato.

RWebMA An R interface toWebMA of Yahoo Japan. Author: Yohei Sato.

RWekajars R/Weka interface jars. Author: Kurt Hornik.

RandForestGUI Authors: Rory Michelland, Genevieve Grundmann.

RcmdrPlugin.EHESsampling Tools for sampling in European Health Examination Surveys (EHES). Author: Statistics Norway.

RcmdrPlugin.SensoMineR Graphical User Interface for **SensoMineR**. Authors: F. Husson, J. Josse, S. Le.

RcmdrPlugin.TextMining Rcommander plugin for **tm** package. Author: Dzemil Lusija.

RcppGSL Rcpp integration for GNU GSL vectors and matrices. Authors: Romain Francois and Dirk Eddelbuettel.

Rd2roxygen Convert Rd to roxygen documentation. Authors: Hadley Wickham and Yihui Xie.

Records Record Values and Record Times. Author: Magdalena Chrapek.

Renext Renewal method for extreme values extrapolation. Authors: Yves Deville, IRSN.

Rfit Rank Estimation for Linear Models. Author: John Kloke.

Rramas Matrix population models. Author: Marcelino de la Cruz.

Runiversal Convert R objects to Java variables and XML. Author: Mehmet Hakan Satman.

RunuranGUI A GUI for the UNU.RAN random variate generators. Author: Josef Leydold.

SAPP Statistical Analysis of Point Processes. Author: The Institute of Statistical Mathematics.

SE.IGE Standard errors of estimated genetic parametes. Author: Piter Bijma.

SII Calculate ANSI S3.5-1997 Speech Intelligibility Index. Author: Gregory R. Warnes.

SMCP Smoothed minimax concave penalization (SMCP) method for genome-wide association studies. Author: Jin (Gordon) Liu.

SPOT Sequential Parameter Optimization. Authors: T. Bartz-Beielstein with contributions from J. Ziegenhirt, W. Konen, O. Flasch, P. Koch, M. Zaefferer.

SQUAREM Squared extrapolation methods for accelerating fixed-point iterations. Author: Ravi Varadhan.

SSSR Server Side Scripting with R. Author: Mehmet Hakan Satman.

SamplerCompare A framework for comparing the performance of MCMC samplers. Author: Madeleine Thompson.

SlimPLS SlimPLS multivariate feature selection. Author: Michael Gutkin.

SortableHTMLTables Turns a data frame into an HTML file containing a sortable table. Author: John Myles White.

TEQR Target Equavelence Range Design. Author: M. Suzette Blanchard.

TRIANGG General discrete triangular distribution. Authors: Tristan Senga Kiessé, Francial G. Libengué, Silvio S. Zocchi, Célestin C. Kokonendji.

TSAgg Time series Aggregation. Author: JS Lessels. In view: *TimeSeries*.

ThreeGroups ML Estimator for Baseline-Placebo-Treatment (Three-group) designs. Author: Holger L. Kern.

TilePlot This package performs various analyses of functional gene tiling DNA microarrays for studying complex microbial communities. Author: Ian Marshall.

TreePar Estimating diversification rate changes in phylogenies. Author: Tanja Stadler.

TunePareto Multi-objective parameter tuning for classifiers. Authors: Christoph Muessel, Hans Kestler.

VecStatGraphs2D Vector analysis using graphical and analytical methods in 2D. Authors: Juan Carlos Ruiz Cuetos, Maria Eugenia Polo Garcia, Pablo Garcia Rodriguez.

VecStatGraphs3D Vector analysis using graphical and analytical methods in 3D. Authors: Juan Carlos Ruiz Cuetos, Maria Eugenia Polo Garcia, Pablo Garcia Rodriguez.

WDI Search and download data from the World Bank's World Development Indicators. Author: Vincent Arel-Bundock.

WGCNA Weighted Gene Co-Expression Network Analysis. Authors: Peter Langfelder and Steve Horvath with contributions by Jun Dong, Jeremy Miller, Lin Song, Andy Yip, and Bin Zhang.

WMTregions Exact calculation of WMTR. Author: Pavel Bazovkin.

abc Functions to perform Approximate Bayesian Computation (ABC) using simulated data. Authors: Katalin Csillery, with contributions from Michael Blum and Olivier Francois.

adaptivetau Tau-leaping stochastic simulation. Author: Philip Johnson.

alabama Constrained nonlinear optimization. Author: Ravi Varadhan (with contributions from Gabor Grothendieck).

allan Automated Large Linear Analysis Node. Author: Alan Lee.

andrews Andrews curves. Author: Jaroslav Myslivec.

aratio Assessment ratio analysis. Author: Daniel McMillen.

ares Environment air pollution epidemiology: a library for time series analysis. Authors: Washington Junger and Antonio Ponce de Leon.

aster2 Aster Models. Author: Charles J. Geyer. In view: *Survival*.

bayesDem Graphical User Interface for **bayesTFR**. Author: Hana Sevcikova.

bayesTFR Bayesian Fertility Projection. Authors: Hana Sevcikova, Leontine Alkema, Adrian Raftery.

bbemkr Bayesian bandwidth estimation for multivariate kernel regression with Gaussian error. Authors: Han Lin Shang and Xibin Zhang.

beadarrayMSV Analysis of Illumina BeadArray SNP data including MSV markers. Author: Lars Gidskehaug.

beeswarm The bee swarm plot, an alternative to stripchart. Author: Aron Charles Eklund.

belief Contains basic functions to manipulate belief functions and associated mass assignments. Authors: N. Maillet, B. Charnomordic, S. Destercke.

benchmark Benchmark Experiments Toolbox. Author: Manuel J. A. Eugster.

ber Batch Effects Removal. Author: Marco Giordan.

bfp Bayesian Fractional Polynomials. Author: Daniel Sabanes Bove.

bild BInary Longitudinal Data. Authors: M. Helena Gonçalves, M. Salomé Cabral and Adelchi Azzalini; incorporates Fortran-77 code written by R. Piessens and E. de Doncker in 'Quadpack'.

bisoreg Bayesian Isotonic Regression with Bernstein Polynomials.

bivarRIpower Sample size calculations for bivariate longitudinal data. Authors: W. Scott Comulada and Robert E. Weiss.

cMonkey cMonkey intgrated biclustering algorithm. Authors: David J Reiss, Institute for Systems Biology.

care CAR Estimation, Variable Selection, and Regression. Authors: Verena Zuber and Korbinian Strimmer.

caspar Clustered and sparse regression (CaSpaR). Author: Daniel Percival.

catR Procedures to generate IRT adaptive tests (CAT). Authors: David Magis (U Liege, Belgium), Gilles Raiche (UQAM, Canada). In view: *Psychometrics*.

cccd Class Cover Catch Digraphs. Author: David J. Marchette.

censReg Censored Regression (Tobit) Models. Author: Arne Henningsen. In view: *Survival*.

cgdsr R-Based API for accessing the MSKCC Cancer Genomics Data Server (CGDS). Author: Anders Jacobsen.

charlson Converts listwise icd9 data into comorbidity count and Charlson Index. Author: Vanessa Cox.

cherry Multiple testing methods for exploratory research. Author: Jelle Goeman.

clustsig Significant Cluster Analysis. Authors: Douglas Whitaker and Mary Christman.

colcor Tests for column correlation in the presence of row correlation. Author: Omkar Muralidharan.

compareGroups Descriptives by groups. Authors: Héctor Sanz, Isaac Subirana, Joan Vila.

constrainedKriging Constrained, covariance-matching constrained and universal point or block kriging. Author: Christoph Hofer. In view: *Spatial*.

corrsieve CorrSieve. Author: Michael G. Campana.

costat Time series costationarity determination and tests of stationarity. Authors: Guy Nason and Alessandro Cardinali.

crossmatch The cross-match test. Authors: Ruth Heller, Dylan Small, Paul Rosenbaum.

cudaBayesregData Data sets for the examples used in the package **cudaBayesreg**. Author: Adelino Ferreira da Silva.

cumSeg Change point detection in genomic sequences. Author: Vito M.R. Muggeo.

curvHDR curvHDR filtering. Authors: George Luta, Ulrike Naumann and Matt Wand.

cxxPack R/C++ Tools for Literate Statistical Practice. Author: Dominick Samperi. In view: *ReproducibleResearch*.

dafs Data analysis for forensic scientists. Authors: James Curran, Danny Chang.

dcv Conventional Cross-validation statistics for climate-growth model. Author: Zongshan Li with contributions from Jinlong Zhang.

ddepn Dynamical Deterministic Effects Propagation Networks: Infer signalling networks for time-course RPPA data. Author: Christian Bender.

delftfews delftfews R extensions. Authors: Mario Frasca, Michèl van Leeuwen.

demography Forecasting mortality, fertility, migration and population data. Authors: Rob J Hyndman with contributions from Heather Booth, Leonie Tickle and John Maindonald.

dicionariosIBGE Dictionaries for reading survey microdata from IBGE. Authors: Erick Fonseca, Alexandre Rademaker.

directlabels Direct labels for plots. Author: Toby Dylan Hocking.

discretization Data preprocessing, discretization for classification. Authors: Kim, H. J.

dismo Species distribution modeling. Authors: Robert Hijmans, Steven Phillips, John Leathwick and Jane Elith.

distory Distance Between Phylogenetic Histories. Authors: John Chakerian and Susan Holmes.

divisors Find the divisors of a number. Author: Greg Hirson.

dixon Nearest Neighbour Contingency Table Analysis. Authors: Marcelino de la Cruz Rot and Philip M. Dixon.

dpa Dynamic Path Approach. Author: Emile Chappin.

eVenn A powerful tool to compare lists and draw Venn diagrams. Author: Nicolas Cagnard.

ecoreg Ecological regression using aggregate and individual data. Author: Christopher Jackson.

egarch EGARCH simulation and fitting. Author: Kerstin Konnerth.

emg Exponentially Modified Gaussian (EMG) Distribution. Author: Shawn Garbett.

evaluate Parsing and evaluation tools that provide more details than the default. Author: Hadley Wickham.

expm Matrix exponential. Authors: Vincent Goulet, Christophe Dutang, Martin Maechler, David Firth, Marina Shapira, Michael Stadelmann.

fANCOVA Nonparametric Analysis of Covariance. Author: Xiao-Feng Wang.

factorQR Bayesian quantile regression factor models. Author: Lane Burgette.

fastR Data sets and utilities for Foundations and Applications of Statistics by R Pruim. Author: Randall Pruim.

fmsb Functions for medical statistics book with some demographic data. Author: Minato Nakazawa.

futile.paradigm A framework for working in a functional programming paradigm in R. Author: Brian Lee Yung Rowe.

fwdmsa Forward search for Mokken scale analysis. Author: Wobbe P. Zijlstra.

gMCP A graphical approach to sequentially rejective multiple test procedures. Author: Kornelius Rohmeyer.

galts Genetic algorithms and C-steps based LTS estimation. Author: Mehmet Hakan Satman.

games Statistical Estimation of Game-Theoretic Models. Authors: Curtis S. Signorino and Brenton Kenkel.

gaussDiff Difference measures for multivariate Gaussian probability density functions. Author: Henning Rust.

gcbd GPU/CPU Benchmarking in Debian-based systems. Author: Dirk Eddelbuettel. In view: *HighPerformanceComputing*.

genepi Genetic Epidemiology Design and Inference. Author: Venkatraman E. Seshan.

geofd Spatial prediction for function value data. Authors: Ramon Giraldo, Pedro Delicado, Jorge Mateu.

glmpathcr Fit a penalized continuation ratio model for predicting an ordinal response. Author: Kellie J. Archer.

glrt Generalized Logrank Tests for Interval-censored Failure Time Data. Authors: Qiang Zhao and Jianguo Sun. In view: *Survival*.

googleVis Using the Google Visualisation API with R. Authors: Markus Gesmann, Diego de Castillo.

gptk Gaussian Processes Tool-Kit. Authors: Alfredo A. Kalaitzis, Pei Gao, Antti Honkela, Neil D. Lawrence.

gridExtra functions in Grid graphics. Author: Baptiste Auguie.

grt General Recognition Theory. Authors: The original Matlab toolbox by Leola A. Alfonso-Reese. R port, with several significant modifications by Kazunaga Matsuki.

gsc Generalised Shape Constraints. Author: Charlotte Maia.

helpr Help for R. Authors: Hadley Wickham and Barret Schloerke.

hisemi Hierarchical Semiparametric Regression of Test Statistics. Authors: Long Qu, Dan Nettleton, Jack Dekkers.

hotspots Hot spots. Author: Anthony Darrouzet-Nardi.

huge High-dimensional Undirected Graph Estimation. Authors: Tuo Zhao, Han Liu, Kathryn Roeder, John Lafferty, Larry Wasserman.

hydroGOF Goodness-of-fit functions for comparison of simulated and observed hydrological time series. Author: Mauricio Zambrano Bigiarini.

hydroTSM Time series management, analysis and interpolation for hydrological modelling. Author: Mauricio Zambrano-Bigiarini.

icaOcularCorrection Independent Components Analysis (ICA) based eye-movement correction. Authors: Antoine Tremblay, IWK Health Center.

igraphtosonia Convert iGraph graps to SoNIA .son files. Author: Sean J Westwood.

imguR Share plots using the image hosting service imgur.com. Author: Aaron Statham.

indicspecies Functions to assess the strength and significance of relationship of species site group associations. Authors: Miquel De Caceres, Florian Jansen.

inference Functions to extract inferential values of a fitted model object. Author: Vinh Nguyen.

infochimps An R wrapper for the infochimps.com API services. Author: Drew Conway.

interactivity toggle R interactive mode. Authors: Jeffrey Horner, Jeroen Ooms.

ipdmeta Individual patient and Mixed-level Meta-analysis of Time-to-Event Outcomes. Author: S. Kovalchik.

isdals Provides data sets for "Introduction to Statistical Data Analysis for the Life Sciences". Authors: Claus Ekstrom and Helle Sorensen.

isva Independent Surrogate Variable Analysis. Author: Andrew E Teschendorff.

iv Information Visualisation with UML and Graphs. Author: Charlotte Maia.

labeling Axis Labeling. Author: Justin Talbot.

laeken Laeken indicators for measuring social cohesion. Authors: Andreas Alfons, Josef Holzer and Matthias Templ. In view: *OfficialStatistics*.

landsat Radiometric and topographic correction of satellite imagery. Author: Sarah Goslee.

lessR Less Code, More Results. Authors: David W. Gerbing, School of Business Administration, Portland State University.

list Multivariate Statistical Analysis for the Item Count Technique. Authors: Graeme Blair, Kosuke Imai.

log4r A simple logging system for R, based on log4j. Author: John Myles White.

longpower Sample size calculations for longitudinal data. Authors: Michael C. Donohue, Anthony C. Gamst, Steven D. Edland.

lubridate Make dealing with dates a little easier. Authors: Garrett Grolemund, Hadley Wickham.

magma Matrix Algebra on GPU and Multicore Architectures. Author: Brian J Smith. In view: *HighPerformanceComputing*.

makesweave Literate Programming with Make and Sweave. Author: Charlotte Maia.

maxLinear Conditional Samplings for Max-Linear Models. Author: Yizao Wang.

mcmcplots Create Plots from MCMC Output. Author: S. McKay Curtis.

mcprofile Multiple Contrast Profiles. Author: Daniel Gerhard.

mem Moving Epidemics Method R Package. Author: Jose E. Lozano Alonso.

memoise Memoise functions. Author: Hadley Wickham.

metatest Fit and test metaregression models. Author: Hilde M. Huizenga & Ingmar Visser.

mfr Minimal Free Resolutions of Graph Edge Ideals. Author: David J. Marchette.

mhurdle Estimation of models with limited dependent variables. Authors: Fabrizio Carlevaro, Yves Croissant, Stephane Hoareau.

mixedQF Estimator with Qudratics Forms in Mixeds Models. Authors: Jean-Benoist Leger, ENS Cachan & Jean-Benoist Leger, INRA.

mixsep Forensic Genetics DNA Mixture Separation. Author: Torben Tvedebrink.

mixsmsn Fitting finite mixture of scale mixture of skew-normal distributions. Authors: Marcos Prates, Victor Lachos and Celso Cabral.

mkssd Efficient multi-level k-circulant supersaturated designs. Author: B N Mandal.

modelcf Modeling physical computer codes with functional outputs using clustering and dimensionality reduction. Author: Benjamin Auder.

modelfree Model-free estimation of a psychometric function. Authors: Ivan Marin-Franch, Kamila Zychaluk, and David H. Foster.

mondate Keep track of dates in terms of months. Author: Dan Murphy. In view: *TimeSeries*.

mpmcorrelogram Multivariate Partial Mantel Correlogram. Author: Marcelino de la Cruz.

mpt Multinomial Processing Tree (MPT) Models. Author: Florian Wickelmaier.

mr Marginal regresson models for dependent data. Authors: Guido Masarotto and Cristiano Varin.

mritc MRI tissue classification. Authors: Dai Feng and Luke Tierney. In view: *MedicalImaging*.

mtsdi Multivariate time series data imputation. Authors: Washington Junger and Antonio Ponce de Leon.

multitaper Multitaper Spectral Analysis. Author: Karim Rahim.

mutoss Unified multiple testing procedures. Authors: MuToss Coding Team (Berlin 2010), Gilles Blanchard, Thorsten Dickhaus, Niklas Hack, Frank Konietschke, Kornelius Rohmeyer, Jonathan Rosenblatt, Marsel Scheer, Wiebke Werft.

mutossGUI A graphical user interface for the MuToss Project. Authors: MuToss Coding Team (Berlin 2010), Gilles Blanchard, Thorsten Dickhaus, Niklas Hack, Frank Konietschke, Kornelius Rohmeyer, Jonathan Rosenblatt, Marsel Scheer, Wiebke Werft.

nacopula Nested Archimedean Copulas. Authors: Marius Hofert and Martin Maechler. In view: *Distributions*.

neuRosim Functions to generate fMRI data including activated data, noise data and resting state data. Authors: Marijke Welvaert with contributions from Joke Durnez, Beatrijs Moerkerke, Yves Rosseel and Geert Verdoolaege.

nnc Nearest Neighbor Autocovariates. Authors: A. I. McLeod and M. S. Islam.

nncRda Improved RDA Using nnc. Authors: M. S. Islam and A. I. McLeod.

normwhn.test Normality and White Noise Testing. Author: Peter Wickham.

npst A generalization of the nonparametric seasonality tests of Hewitt et al. (1971) and Rogerson (1996). Author: Roland Rau.

openair Tools for the analysis of air pollution data. Authors: David Carslaw and Karl Ropkins.

operator.tools Utilities for working with R's operators. Author: Christopher Brown.

optimx A replacement and extension of the optim() function. Authors: John C Nash and Ravi Varadhan. In view: *Optimization*.

orQA Order Restricted Assessment Of Microarray Titration Experiments. Author: Florian Klinglmueller.

orderbook Orderbook visualization/Charting software. Authors: Andrew Liu, Khanh Nguyen, Dave Kane. In view: *Finance*.

parfossil Parallelized functions for palaeoecological and palaeogeographical analysis. Author: Matthew Vavrek.

partitionMap Partition Maps. Author: Nicolai Meinshausen.

pathClass Classification using biological pathways as prior knowledge. Author: Marc Johannes.

pathmox Segmentation Trees in Partial Least Squares Path Modeling. Authors: Gaston Sanchez, and Tomas Aluja.

pbapply Adding Progress Bar to '*apply' Functions. Author: Peter Solymos.

pequod Moderated regression package. Author: Alberto Mirisola & Luciano Seta.

pesticides Analysis of single serving and composite pesticide residue measurements. Author: David M Diez.

pfda Paired Functional Data Analysis. Author: Andrew Redd.

pglm panel generalized linear model. Author: Yves Croissant.

pheatmap Pretty Heatmaps. Author: Raivo Kolde.

phitest Nonparametric goodness-of-fit methods based on phi-divergences. Author: Leah R. Jager.

phybase Basic functions for phylogenetic analysis. Author: Liang Liu. In view: *Phylogenetics*.

phylosim R packge for simulating biological sequence evolution. Authors: Botond Sipos, Gregory Jordan.

phylotools Phylogenetic tools for ecologists. Authors: Jinlong Zhang, Xiangcheng Mi, Nancai Pei.

pi0 Estimating the proportion of true null hypotheses for FDR. Authors: Long Qu, Dan Nettleton, Jack Dekkers.

plotGoogleMaps Plot HTML output with Google Maps API and your own data. Author: Milan Kilibarda.

plsRglm Partial least squares Regression for generalized linear models. Authors: Frederic Bertrand, Nicolas Meyer, Myriam Maumy-Bertrand. In views: *ChemPhys*, *Psychometrics*.

pmr Probability Models for Ranking Data. Authors: Paul H. Lee and Philip L. H. Yu.

polysat Tools for Polyploid Microsatellite Analysis. Author: Lindsay V. Clark.

portes Portmanteau Tests for ARMA, VARMA, ARCH, and FGN Models. Author: Esam Mahdi & A. Ian McLeod.

ppstat Point Process Statistics. Author: Niels Richard Hansen.

processdata Process Data. Author: Niels Richard Hansen.

pso Particle Swarm Optimization. Author: Claus Bendtsen.

ptinpoly Point-In-Polyhedron Test (3D). Authors: Jose M. Maisog, Yuan Wang, George Luta, Jianfei Liu.

pvclass P-values for Classification. Authors: Niki Zumbrunnen, Lutz Duembgen.

qualityTools Statistical Methods for Quality Science. Author: Thomas Roth.

rAverage Parameter estimation for the Averaging model of Information Integration Theory. Authors: Giulio Vidotto, Stefano Noventa, Davide Massidda, Marco Vicentini.

rDNA R Bindings for the Discourse Network Analyzer. Author: Philip Leifeld.

rJython R interface to Python via Jython. Authors: G. Grothendieck and Carlos J. Gil Bellosta (authors of Jython itself are Jim Hugunin, Barry Warsaw, Samuele Pedroni, Brian Zimmer, Frank Wierzbicki and others; Bob Ippolito is the author of the simplejson Python module).

rTOFsPRO Time-of-flight mass spectra signal processing. Authors: Dariya Malyarenko, Maureen Tracy and William Cooke.

refund Regression with Functional Data. Authors: Philip Reiss and Lei Huang.

relevent Relational Event Models. Author: Carter T. Butts.

reportr A general message and error reporting system. Author: Jon Clayden.

reshape2 Flexibly reshape data: a reboot of the **reshape** package. Author: Hadley Wickham.

rmac Calculate RMAC or FMAC agreement coefficients. Author: Jennifer Kirk.

rphast R interface to PHAST software for comparative genomics. Authors: Melissa Hubisz, Katherine Pollard, and Adam Siepel.

rpud R functions for computation on GPU. Author: Chi Yau. In view: *HighPerformanceComputing*.

rrcovNA Scalable Robust Estimators with High Breakdown Point for Incomplete Data. Author: Valentin Todorov. In view: *OfficialStatistics*.

rseedcalc Estimation of Proportion of GM Stacked Seeds in Seed Lots. Authors: Kevin Wright, Jean-Louis Laffont.

rvgtest Tools for Analyzing Non-Uniform Pseudo-Random Variate Generators. Authors: Josef Leydold and Sougata Chaudhuri.

satin Functions for reading and displaying satellite data for oceanographic applications with R. Authors: Héctor Villalobos and Eduardo González-Rodríguez.

schwartz97 A package on the Schwartz two-factor commodity model. Authors: Philipp Erb, David Luethi, Juri Hinz, Simon Otziger. In view: *Finance*.

sdcMicroGUI Graphical user interface for package **sdcMicro**. Author: Matthias Templ.

selectMeta Estimation weight functions in meta analysis. Author: Kaspar Rufibach.

seqCBS CN Profiling using Sequencing and CBS. Authors: Jeremy J. Shen, Nancy R. Zhang.

seqRFLP Simulation and visualization of restriction enzyme cutting pattern from DNA sequences. Authors: Qiong Ding, Jinlong Zhang.

sharx Models and Data Sets for the Study of Species-Area Relationships. Author: Peter Solymos.

sigclust Statistical Significance of Clustering. Authors: Hanwen Huang, Yufeng Liu & J. S. Marron.

simexaft Authors: Juan Xiong, Wenqing He, Grace Y. Yi. In view: *Survival*.

sinartra A simple web app framework for R, based on sinatra. Author: Hadley Wickham.

smirnov Provides two taxonomic coefficients from E. S. Smirnov "Taxonomic analysis" (1969) book. Author: Alexey Shipunov (with help of Eugenij Altshuler).

smoothmest Smoothed M-estimators for 1-dimensional location. Author: Christian Hennig.

soil.spec Soil spectral data exploration and regression functions. Author: Thomas Terhoeven-Urselmans.

soiltexture Functions for soil texture plot, classification and transformation. Authors: Julien MOEYS, contributions from Wei Shangguan.

someMTP Functions for Multiplicty Correction and Multiple Testing. Author: Livio Finos.

spacetime classes and methods for spatio-temporal data. Author: Edzer Pebesma.

splinesurv Nonparametric bayesian survival analysis. Authors: Emmanuel Sharef, Robert L. Strawderman, and David Ruppert. In view: *Survival*.

sprint Simple Parallel R INTerface. Author: University of Edinburgh SPRINT Team.

survJamda Survival prediction by joint analysis of microarray gene expression data. Author: Haleh Yasrebi.

survJamda.data Author: Haleh Yasrebi.

synchronicity Boost mutex functionality for R. Author: Michael J. Kane.

tableplot Represents tables as semi-graphic displays. Authors: Ernest Kwan and Michael Friendly.

tfer Forensic Glass Transfer Probabilities. Authors: James Curran and TingYu Huang.

thgenetics Genetic Rare Variants Tests. Author: Thomas Hoffmann.

tpsDesign Design and analysis of two-phase sampling and case-control studies. Authors: Takumi Saegusa, Sebastien Haneuse, Nilanjan Chaterjee, Norman Breslow.

treemap Treemap visualization. Author: Martijn Tennekes.

triads Triad census for networks. Authors: Solomon Messing, Sean J Westwood, Mike Nowak and Dan McFarland.

validator External and Internal Validation Indices. Author: Marcus Scherl.

violinmplot Combination of violin plot with mean and standard deviation. Author: Raphael W. Majeed.

wSVM Weighted SVM with boosting algorithm for improving accuracy. Authors: SungHwan Kim and Soo-Heang Eo.

weirs A Hydraulics Package to Compute Open-Channel Flow over Weirs. Author: William Asquith.

wild1 R Tools for Wildlife Research and Management. Authors: Glen A. Sargeant, USGS Northern Prairie Wildlife Research Center.

wvioplot Weighted violin plot. Author: Solomon Messing.

xpose4 Xpose 4. Authors: E. Niclas Jonsson, Andrew Hooker and Mats Karlsson.

xpose4classic Xpose 4 Classic Menu System. Authors: E. Niclas Jonsson, Andrew Hooker and Mats Karlsson.

xpose4data Xpose 4 Data Functions Package. Authors: E. Niclas Jonsson, Andrew Hooker and Mats Karlsson.

xpose4generic Xpose 4 Generic Functions Package. Authors: E. Niclas Jonsson, Andrew Hooker and Mats Karlsson.

xpose4specific Xpose 4 Specific Functions Package. Authors: E. Niclas Jonsson, Andrew Hooker and Mats Karlsson.

Other changes

- The following packages were moved to the Archive: **AIS**, **BayesPanel**, **BootCL**, **DEA**, **FKBL**, **GRASS**, **MSVAR**, **NADA**, **PhySim**, **RGtk2DfEdit**, **Rfwdmv**, **Rlsf**, **SNPMaP**, **SharedHT2**, **TwslmSpikeWeight**, **VDCutil**, **VaR**, **XReg**, **aaMI**, **asuR**, **bayesCGH**, **bicreduc**, **bim**, **birch**, **celsius**, **csampling**, **cts**, **demogR**, **dprep**, **glmc**, **grasp**, **grnnR**, **ifa**, **ig**, **inetwork**, **ipptoolbox**, **knnTree**, **lnMLE**, **locfdr**, **logilasso**, **lvplot**, **marg**, **mlCopulaSelection**, **mmlcr**, **netmodels**, **normwn.test**, **npde**, **nytR**, **ofw**, **oosp**, **pga**, **pinktoe**, **ppc**,

qgen, **risksetROC**, **rjacobi**, **rv**, **sabreR**, **singlecase**, **smd.and.more**, **stochasticGEM**, **titecrm**, and **udunits**.

- The following packages were resurrected from the Archive: **MIfuns**, **agsemisc**, **assist**, **gllm**, **mlmmm**, **mvgraph**, **nlts**, **rgr**, and **supclust**.
- The following packages were removed from CRAN: **seaflow** and **simpleR**.

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

Achim Zeileis
Universität Innsbruck, Austria
Achim.Zeileis@R-project.org

News from the Bioconductor Project

by the Bioconductor Team
Program in Computational Biology
Fred Hutchinson Cancer Research Center

We are pleased to announce Bioconductor 2.7, released on October 18, 2010. Bioconductor 2.7 is compatible with R 2.12.0, and consists of 419 packages. There are 34 new packages, and enhancements to many others. Explore Bioconductor at http://bioconductor.org, and install packages with

```
> source("http://bioconductor.org/biocLite.R")
> biocLite() # install standard packages...
> biocLite("IRanges") # ...or IRanges
```

New and revised packages

This release includes new packages for diverse areas of high-throughput analysis. Highlights include:

Next-generation sequencing packages for ChIP (**iSeq**, **RMAPPER**), methylated DNA immunoprecipitation (**MEDIPS**), and RNA-seq (**rnaSeqMap**) work flows, 454 sequencing (**R453Plus1Toolbox**) and management of microbial sequences (**OTUbase**).

Microarray analysis of domain-specific applications (array CGH, **ADaCGH2**; tiling arrays, **les**; miRNA, **LVSmiRNA**; and bead arrays, **MBCB**); specialized statistical methods (**fabia**, **farms**, **RDRToolbox**), and graphical tools (**IsoGeneGUI**).

Gene set, network, and graph oriented approaches and tools include **gage**, **HTSanalyzeR**, **PatientGeneSets**, **BioNet**, **netresponse**, **attract**, **CoGAPS**, **ontoCAT**, **DEgraph**, **NTW**, and **RCytoscape**.

Advanced statistical and modeling implementations relevant to high-throughtput genetic analysis include **BHC** (Bayesian Hierarchical Clustering), **CGEN** (case-control studies in genetic epidemiology), and **SQUADD**.

Image, cell-based, and other assay packages, include **imageHTS**, **CRImage**, **coRNAi**, **GeneGA**, **NuPoP**.

Our large collection of microarray- and organism-specific annotation packages have been updated to include information current at the time of the Bioconductor release. These annotation packages contain biological information about microarray probes and the genes they are meant to interrogate, or contain gene-based annotations of whole genomes. They are particularly valuable in providing stable annotations for repeatable research.

Several developments in packages maintained by the Bioconductor core team are noteworthy. The *graphBAM* class in the **graph** package is available to manipulate very large graphs. The **GenomicRanges**, **GenomicFeatures**, and **Biostrings** packages have enhanced classes such as *TranscriptDb* for representing genome-scale 'track' annotations from common data resources, *MultipleAlignment* for manipulating reference (and other moderate-length) sequences in a microbiome project, and *SummarizedExperiment* to collate range-based count data across samples in sequence experiments. The **chipseq** package has enhanced functionality for peak calling, and has been updated to use current data structures.

Further information on new and existing packages can be found on the Bioconductor web site, which contains 'views' that identify coherent groups of packages. The views link to on-line package descriptions, vignettes, reference manuals, and use statistics.

Other activities

The Bioconductor community met on July 28-30 at our annual conference in Seattle for a combination of scientific talks and hands-on tutorials, and on November 17-18 in Heidelberg, Germany for a meeting highlight contributions from the European developer community. The active Bioconductor mailing lists (http://bioconductor.org/docs/mailList.html) connect users with each other, to domain experts, and to maintainers eager to ensure that their packages satisfy the needs of leading edge approaches. Bioconductor package maintainers and the Bioconductor team invest considerable effort in producing high-quality software. The Bioconductor team continues to ensure quality software through technical and scientific reviews of new packages, and daily builds of released packages on Linux, Windows, and Macintosh platforms.

Looking forward

Contributions from the Bioconductor community play an important role in shaping each release. We anticipate continued efforts to provide statistically informed analysis of next generation sequence data, especially in the down-stream analysis of comprehensive, designed sequencing experiments and integrative analyses. The next release cycle promises to be one of active scientific growth and exploration.

R Foundation News

by Kurt Hornik

Donations and new members

Donations

Agrocampus Ouest, France
Helsana Versicherungen AG, Switzerland
Kristian Kieselman, Sweden

New benefactors

eoda, Germany
Giannasca Corrado & Montagna Maria, Italy

New supporting institutions

Marine Scotland Science, UK

New supporting members

Ian M. Cook, USA
Chris Evans, UK
Martin Fewsow, Germany
Laurence Frank, Netherlands
Susan Gruber, USA
Robert A. Muenchen, USA
Geoff Potvin, USA
Ivo Welch, USA
Alex Zolot, USA

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