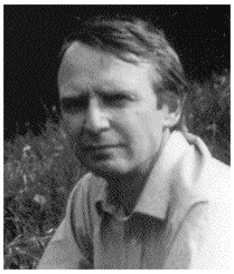
`rm.tools`: some S–PLUS tools for the exploratory and parametric analysis of repeated measures data

Adelchi Azzalini
(adelchi@pearson.stat.unipd.it)

Monica Chiogna
(monica@pearson.stat.unipd.it)



Dipartimento di Scienze Statistiche
Università di Padova
Italia

Rapporto tecnico n. 4.1995
Dicembre 1995

Summary

Software tools are presented for the exploratory and confirmatory analysis of repeated measures data. They are written in S–PLUS language, except for some FORTRAN routines which are interfaced through S–PLUS. Three main procedures are described, called `rm.plot()`, `rm.mean()` and `rm.fit()`. These are intended for data plotting, preliminary analysis and likelihood analysis, respectively. Some additional auxiliary procedures are included. All the procedures allow for the presence of missing or unbalanced data, time dependent covariates and for the presence of serial correlation. The procedure for fitting parametric models allows three types of response variables: normal, binary and Poisson.

Keywords: repeated measures, longitudinal data, smoothing, serial dependence, graphical methods, random effects models, unbalanced data, missing data, logistic regression, Poisson regression.

Contents

1 Introduction

1.1 Repeated measures data

Repeated measures (RM) data are obtained by observing a group of experimental units (called *subjects* or *individuals* in this context) over a number of occasions; the term longitudinal data is essentially equivalent. For a brief introduction to RM, see Ware (1985); more extended treatments are given by Crowder & Taylor (1990), Lindsey (1993), and Diggle, Liang & Zeger (1994).

In their simplest form, RM data appear as matrix of response values having as many rows as the number of subjects and as many columns as the number of replicated observations for each subject. The successive observations on the same subject, i.e. a row of the response matrix mentioned above, are sometimes called a *profile*. In more complicated situations, the subjects do not share the same number of observations and the time spacings may differ.

Although RM has been a long standing issue of statistical methodology, its importance and popularity have developed enormously in the '80s. At the moment, the statistical theory for the analysis of linear parametric models under normality assumption of the data can be considered fairly stabilized; current research concentrates on non–linear models and discrete data.

In spite of a remarkable research activity, there are not many computer programs and packages providing facilities for some of the most recent developments of the methodology.

The present report describes the features and the use of some procedures written in S–PLUS (version 3.2) language providing tools for graphical, non–parametric and parametric analysis of RM data. Some auxiliary FORTRAN routines are accessed through S–PLUS. The S language is described by Becker, Chambers & Wilks (1988); for its extension S–PLUS, see Statistical Sciences (1993).

The first version of this project was developed in 1993. The present version in an enhancement in three main directions:

(a) in the normal data case, a more general form of covariance structure has been introduced;

(b) discrete response variables (either binary or Poisson) are allowed; when a single profile is considered, this covers the case of logistic regression for autocorrelated observations;

(c) plotting routines have been substantially modified and improved.

Moreover, a great deal of the remaining code has been re-written.

1.2 A brief overview

Three main functions are provided, two intended for exploratory and non–parametric analysis, one for parametric model fitting under normality assumptions. A quick description is as follows.

`rm.plot()` plots individual profiles versus time, allowing the selection, highlighting and deletion of profiles according as whether the associated (subject–specific) covariates take certain values;

`rm.mean()` plots and returns average profiles of groups of subjects versus time, either by computing plain averages at each time point or by performing non–parametric smoothing, possibly with pointwise 95% confidence bands;

`rm.fit()` is the main module, and its role is to fit parametric models via likelihood methods, assuming that the response belongs to one of three possible types of distributions: normal, binary, or Poisson; in all cases, serial dependence is allowed for, according to some stochastic model suitable for the chosen type of distribution.

All these functions allow unbalanced and missing data, and unequal time spacings across the profiles. The same data structure is shared by all functions. This structure is of `list` type, as specified later; so it is possible to define this `list` only once, and then perform the entire data analysis working with this object.

The practical usage of the three modules is best introduced by practical examples, via a commented transcript of a session, in § 3. Before that, some background information on parametric models for likelihood analysis, to be fitted by `rm.fit()`, is provided in § 2. Some advanced features will be described in § 4. Section 5 provides additional examples. Table 3, at page 37, provides a quick list of the main options of the functions and their meaning. A more formal and exhaustive description of the syntax of the same modules is provided at the end of the report, including also a description of some subsidiary modules, not mentioned so far.

2 Parametric models

This section is organized in three subsections, each one providing background information for the corresponding types of distribution allowed for by `rm.fit()`. It is worth noticing that, at variance from many other existing proposals, only likelihood based methods are used here, and a full stochastic model for the data is introduced, not just a partial specification of it.

Under the standard assumption of independence between individuals, the discussion can concentrate on the modeling of a single profile as the overall log–likelihood can then be obtained by summing the contributions of each of the n subjects..

For a generic individual, denote by $y = (y_1, y_2, \ldots, y_T)'$ the vector of the response values obtained at time points $t_1, t_2, \ldots, t_T$, and by x_j a vector containing a set of covariate values associated to y_j (for $j = 1, \ldots, T$). An important distinction is between *subject–specific covariates* which are constant over time but vary across individuals and *time–varying covariates* which are constant across individuals but vary with time (time itself is one such covariate); sometimes covariates vary both within and between individuals.

Broadly speaking, the aim of the analysis is to assess the effect of the covariates values on the behaviour of the response variable, allowing for the presence of correlations among the y_j's, since they are measured on the same individual.

2.1 Model for normal data

To ease exposition, we start from a fairly simple setting, and then we develop it to deal with more complicated cases.

In a large number of situations, it is reasonable to interpret the observed value y_j at time t_j as depending on three components, namely

- the effect of a number of covariates, contained in the vector x_j;
- an individual random effect, to account for the variability between individuals, even if they share the covariate values x_j;
- an additional random component which accounts for the serial dependence between successive observations on the same individual.

There are clearly many possible specifications of the above structure, of which the simplest seems to be:

- to assume linearity of the covariates effects, namely to set them to $x_j'\beta$ for some vector β (constant across individuals, to avoid unidentifiability problems),
- to assume additivity of the three components above,
- to describe serial dependence by a correlation structure of very simple form, such as the one associated with first order autoregression.

Our parametric model for observation y_j then becomes

$$y_j = x_j'\beta + d + \eta_j \qquad (j = 1, \ldots, T) \tag{1}$$

with d being a random component specific to the individual and η_j the random component describing the serial dependence. Therefore, d contributes to $\mathrm{var}(y)$ via a matrix with all elements equal.

For the η process assume that

$$\mathrm{corr}\,(\eta_i, \eta_j) = \rho^{|t_i - t_j|} = (\Omega)_{ij} \tag{2}$$

for some $-1 < \rho < 1$. This expression gives the correlation matrix of stationary first-order autoregressive process, AR(1) for brevity, when all time spacings are equal. In the general case of unequal time spacings and $\rho > 0$, (2) gives the correlation matrix of a continuous time Ornstein–Uhlenbeck process observed at a discrete set of time points; however, for simplicity, we shall use the phrase 'AR(1)' in both cases.

Adding the assumption of normality and independence of d and η, we can write for the whole profile y of a subject

$$y \sim N_T(X\beta, \sigma^2 R) \tag{3}$$

where X is a matrix with j-th row x_j' and

$$R = \Omega + \delta J \tag{4}$$

with $\sigma^2 > 0$, $\delta \geq 0$, and J a $T \times T$ matrix of all ones. Besides having conceptual simplicity, matrix form (4) has the advantage of allowing explicit matrix inversion and computation of the determinant, using results given, e.g., by Glasbey (1989) and by standard matrix theory textbooks.

Depending on the form of the X matrix, various types of covariates are possible. For instance, if X has a constant column, this is effectively a subject–specific covariate, constant across time. However, building of the X matrix is taken care of by `rm.fit()`, as described later.

Except for a minor change of the parameterization, the covariance structure (4) is similar to that considered by Azzalini (1984) with the extra feature of possible unequal time spacings, and it includes a broad subset of the covariance structures discussed by Diggle (1988).

On setting $\delta = 0$, the subject random effect component d is removed from (1). On setting $\rho = 0$, serial dependence is eliminated, leading to a matrix R with compound symmetry (CS) structure, which in the past used to be the standard method for handling repeated measures data as described, for instance, by Winer (1971).

We now move on to consider more complicated covariance structures. So far, the subject random effect has been supposed to produce a perturbation of the intercept only, by an amount d, but in some cases its effect may be more complex.

A typical example of this situation arises in the context of growth curves analysis, as in the following prototype case. Suppose that the profile of an individual is described by a form like

$$y_j = b_0 + b_1 t_j + \eta_j \qquad (j = 1, \ldots, T) \tag{5}$$

where b_0 and b_1 are parameters. If these parameters are constant across individuals (more precisely, across individuals with the same subject-specific covariates), we are back to the previous situation, with d removed. Suppose instead that b_0 and b_1 vary across individuals; in particular, suppose that the specific values for any given individual are generated as

$$b_0 \sim N(\beta_0, \sigma^2\delta_0), \qquad b_1 \sim N(\beta_1, \sigma^2\delta_1)$$

where δ_0, δ_1 are positive constants. Under the new assumptions, (4) is modified to

$$R = \Omega + X\,\mathrm{diag}(\delta_0, \delta_1)X'$$

where X is the design matrix associated to (5).

Notice the following points:

- the argument is not restricted to the simple linear regression (5), and it extends immediately to multiple regression;
- the random effects do not need to affect all components of the linear models; for instance, if $\delta_1 = 0$ and $\delta_0 = \delta$, we are back to (4), replacing X by 1_T, the T-vector of all 1's.

These remarks motivate the introduction of the more general correlation structure

$$R = \Omega + F'\Delta F \tag{6}$$

where F is a $q \times T$ matrix of constants (not necessarily a subset of X) and

$$\Delta = \mathrm{diag}(\delta_1, \ldots, \delta_q)$$

Table 1: Covariance structures allowed for normal data

Case	ρ	F'	syntax used by `rm.fit()`	
			`cov.struct=`	`random.effects=`
Independence	0		`"I"`	
Compound symmetry (CS)	0	1_T	`"I+random"`	
AR(1)	free		`"AR(1)"`	
AR(1) combined with CS	free	1_T	`"AR(1)+random"`	
Simple linear growth curves with conditional independence	0	$(1_T, x)$	`"I+random"`	`y~x`
Linear growth curves with AR(1) serial dependence	free	$(1_T, x_1, \ldots, x_q)$	`"AR(1)+random"`	`y~x1+...+xq`

where $\delta_1, \ldots, \delta_q$ are nonnegative parameters. For an extended discussion of these issues, see Laird & Ware (1982).

To conclude, the general form of the stochastic model for an individual profile is given by (3) with R as in (6). By suitable restrictions on the form of R or by appropriate specification of F, various special cases and variants can be obtained, as sketched in Table 1. The table includes the description of the corresponding syntax adopted by `rm.fit()`, which will be properly explained in § 3.4, § 4.1 and the documentation at the end of this report.

2.2 Model for binary data

Differently from the Gaussian case, the state of the methodology for RM data is not yet consolidated when the response variable is discrete, and various proposals are still under discussion in the literature.

For the case of binary data, we consider the model proposed by Azzalini (1994) which is effectively a logistic regression model with the addition of serial dependence of Markovian form. Specifically, assume that the generic profile $y = (y_1, \ldots, y_T)'$ is a vector of 0 and 1 elements such that

$$E(y_j) = \theta_j$$

where

$$\operatorname{logit}(\theta_j) = x_j'\beta \qquad (j = 1, \ldots, T)$$

and β is a vector of parameters of interest. The logit link function could 'obviously' be replaced by any other legitimate link, but we shall restrict ourselves to the logit link, for simplicity and because other links are rarely used in practice.

A simple form of serial dependence is allowed for by assuming that the sequence $y_1, \ldots, y_T$ is generated by an (inhomogeneous) Markov chain such that the log-odds ratio between adjacent observations, namely

$$\lambda = \log \frac{\Pr(y_{j-1} = 1,\, y_j = 1)\ \Pr(y_{j-1} = 0,\, y_j = 0)}{\Pr(y_{j-1} = 0,\, y_j = 1)\ \Pr(y_{j-1} = 1,\, y_j = 0)},$$

is constant, for $j = 2, \ldots, T$. When $\lambda = 0$, data are independent, and we are back to regular logistic regression. For a generic λ, there is positive or negative association between adjacent data, depending on the sign of λ.

For the expression of the corresponding likelihood function and for further details, we refer the reader to the above mentioned paper. Notice however the following points:

- the approach holds also for a single profile, a situation of interest on its own;
- no individual random effect is included in this formulation, as yet;
- randomly missing data can be catered for, although the present implementation introduces restrictions on their pattern, as described later.

2.3 Model for Poisson data

The approach adopted for Poisson response variables is the one developed independently by KcKenzie (1985) and by Azzalini (1995), and it is similar in spirit to the one followed for binary data, although the technical development is somewhat different.

Suppose that the components of $y = (y_1, \ldots, y_T)'$ are random variables, each having a Poisson distribution. The mean value of the y_j's is modeled as it is usual for independent variables, namely

$$E(y_j) = \theta_j$$

where

$$g(\theta_j) = x_j'\beta \qquad (j = 1, \ldots, T).$$

Here, $g(.)$ indicates a link function; the identity and the logarithmic link are the most relevant ones for Poisson data. Serial dependence is again assumed to be of Markovian type, more specifically in the form of 'discrete autoregression'

$$y_j = \rho \circ y_{j-1} + \varepsilon_j \qquad (j = 2, \ldots, T)$$

where $\rho \circ y_{j-1}$ is suitably defined so that the outcome is a nonnegative integer valued variable, for all $0 \leq \rho < 1$, and ε_j is a Poisson random disturbance. When $\rho = 0$, the model reduces to a common log-linear model for independent Poisson variables, while increasing ρ implies higher autocorrelation between observations.

One peculiar feature of this model is that it implies that

$$\nu_j = \theta_j - \rho\,\theta_{j-1}$$

is positive ($j = 2, \ldots, T$). This means that certain choices of (β, ρ) are not compatible with the model, for instance when they corresponds to decreasing θ_j's. In addition, negative autocorrelations are not allowed. This situation makes this model appropriate only for data of a suitable nature, and it is not intended as a general-purpose model for Poisson data.

Table 2: Data of Andersen, Jensen & Schou (1981).

1	-1	93	109	114	121	101	109	112	107	97	117	89	132	121	124
2	-1	116	116	111	135	107	115	114	106	92	98	116	105	135	93
3	1	125	166	180	137	142	114	119	121	95	105	152	154	102	110
4	1	144	157	161	173	158	138	148	147	133	124	122	133	122	130
5	1	105	134	128	119	136	126	125	125	103	91	98	112	133	124
6	-1	109	121	100	83	87	110	109	100	93	80	98	100	104	97
7	-1	89	109	107	95	101	96	88	83	85	91	95	109	116	86
8	-1	116	138	138	128	102	116	122	100	123	107	117	120	119	99
9	1	151	165	156	149	136	142	141	128	130	126	154	148	138	127
10	1	137	155	145	139	150	141	125	109	118	109	112	102	107	107

3 A sample session

This section gives an introductory example of the use of the repeated measurements functions `rm.plot()`, `rm.mean()`, `rm.fit()`. To this end, the dataset of Andersen, Jensen and Schou (1981) will be used. This is a "traditional" dataset used by a number of people for illustrating RM methods, and is reproduced in Table 1.

Here, rows represent individuals, the first column numbers the individuals, the second column contains the observed values for a dichotomous covariate x, while the remaining fourteen columns contain the fourteen repeated observations of the response variable on the individuals, namely plasma citrate concentration.

3.1 Creating the data list

Data lists can be created in a number of ways: by binding together objects of various kind; by replacements or additions to existing data lists or by other user–defined specialized functions.

In this example, we start by reading the data from an external table–like file named "citrate.dat", containing the data shown in Table 1. This may be done using standard S–PLUS input functions. Usually, the best way of reading mixed data in a table format from an external file is by using the function `read.table()`:

```
> citrate <- read.table("citrate.dat")
```

The output is contained in a data frame with as many rows as the file has lines and as many columns as the file has fields, unless optional parameters are specified in the call. We check the content of the data frame by typing:

```
> citrate
   V1 V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16
 1  1 -1  93 109 114 121 101 109 112 107  97 117  89 132 121 124
```

```
2   2 -1 116 116 111 135 107 115 114 106  92  98 116 105 135  93
3   3  1 125 166 180 137 142 114 119 121  95 105 152 154 102 110
4   4  1 144 157 161 173 158 138 148 147 133 124 122 133 122 130
5   5  1 105 134 128 119 136 126 125 125 103  91  98 112 133 124
6   6 -1 109 121 100  83  87 110 109 100  93  80  98 100 104  97
7   7 -1  89 109 107  95 101  96  88  83  85  91  95 109 116  86
8   8 -1 116 138 138 128 102 116 122 100 123 107 117 120 119  99
9   9  1 151 165 156 149 136 142 141 128 130 126 154 148 138 127
10 10  1 137 155 145 139 150 141 125 109 118 109 112 102 107 107
```

To build the data list we start extracting the matrix of the response values, whose rows must contain the individual replicated observations. Since variables in a data frame can be of any type (numerical, categorical, mixed), the data frame as an object and objects extracted from it do not have any type. Therefore, whenever an object is extracted from a data frame, its type must be declared. It is normally easier to make such declarations when extracting the objects, although type declarations and other type–conversions can be applied at any later point. The matrix can be obtained by:

```
> response.matrix <-as.matrix(citrate[ ,3:16])
```

If the original table used columns instead of rows to contain individual replications, the obtained matrix should be transposed.

In the same way we extract from the data frame all the covariates which are going to be used in the following analyses. Assuming that the codes -1 and 1 in x represent respectively "yes" and "not" values for the variable x, a factor declaration and extraction can be performed in the following way:

```
> x.vector <- factor(citrate[ ,2], levels = c(-1,1), labels = c("yes", "not"))

> x.vector
 [1] yes yes not  not  not  yes yes yes not  not
```

The data list must also contain the time component, i.e. the component representing the times of the observations. Since all individuals in the example share the same time of observation, the time component can be represented by a vector. Assuming that the individuals have been observed at regular intervals of time starting from time one, we can create a time vector by:

```
> time.vector <- seq(1:14)

> time.vector
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14
```

When all the components have been prepared, they can be bound in a list. Each component in the list must carry a name, which is used by the repeated measurements functions for extracting, inserting and selecting variables and/or individuals and for creating fitted models. Since names for elements of a list are more easily assigned when creating the list than in later stages, a convenient way of creating the final data list is:

```
> citrate <- list(y = response.matrix, x = x.vector,
                               time = time.vector)
```

```
> citrate
$y:
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
 [1,]  93.2  109  114  121  101  109  112  107   97   117    89
 [2,] 116.0  116  111  135  107  115  114  106   92    98   116
 [3,] 125.0  166  180  137  142  114  119  121   95   105   152
 [4,] 144.0  157  161  173  158  138  148  147  133   124   122
 [5,] 105.0  134  128  119  136  126  125  125  103    91    98
 [6,] 109.0  121  100   83   87  110  109  100   93    80    98
 [7,]  89.0  109  107   95  101   96   88   83   85    91    95
 [8,] 116.0  138  138  128  102  116  122  100  123   107   117
 [9,] 151.0  165  156  149  136  142  141  128  130   126   154
[10,] 137.0  155  145  139  150  141  125  109  118   109   112
     [,12][,13] [,14]
 [1,]  132  121   124
 [2,]  105  135    93
 [3,]  154  102   110
 [4,]  133  122   130
 [5,]  112  133   124
 [6,]  100  104    97
 [7,]  109  116    86
 [8,]  120  119    99
 [9,]  148  138   127
[10,]  102  107   107

$x:
 [1] "yes" "yes" "not" "not" "not" "yes" "yes" "yes" "not" "not"
$x: Levels:
[1] "yes" "not"

$time:
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14
```

Note that it is not necessary to name the response matrix `response`, as we have used in the example.

When the response is discrete, the list can also be used to represent a table of profiles. In this case, rows of the response matrix represent the observed patterns of profiles and a vector must be inserted in the list which contains the counts, i.e. the observed replications of each row of the response matrix. By default, the programs expect such vector to be named "`counts`". If each profile has been observed only once, the construction of the vector `counts` is not required.

The list can now be passed to the repeated measurements functions.

3.2 Exploring individual profiles with `rm.plot()`

Individual profiles can be explored using `rm.plot()`, a graphical function designed to allow an overall look at the data. Before a call to `rm.plot()` is made, a graphics device must be initialized. For instance, one could use:

```
> motif()
```

but similar commands, such as `openlook()`, are possible.

The call to `rm.plot()` requires the selection of a data list and the definition of a formula used to select and possibly transform variables to be plotted from the data list. The simplest call to the function is given by :

```
> rm.plot(y ~ time, data = citrate)
```

which produces Figure 1.

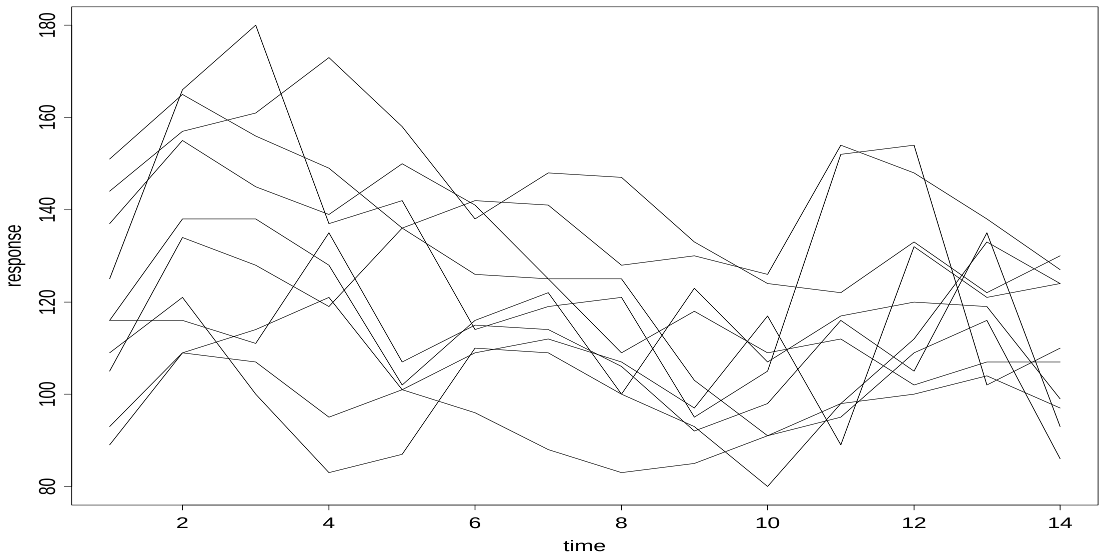


Figure 1: Standard layout of `rm.plot()`

Labels of axes in the plot can be modified easily by the use of the optional arguments `timelab` and `ylab`:

```
> rm.plot(y ~ time, data = citrate, timelab
        = "time of observation")
```

Standard transformations can be applied to the variables in the formula, like for example:

```
> rm.plot(log(y) ~ time, data = citrate)
```

Other optional arguments can be used in the call to `rm.plot()`. Suppose that a narrower range of time (or response) values is to be studied; a zoomed plot of the profiles limited to the restricted range of interest can be obtained by setting the argument `timelim` (or `ylim`) in the call to `rm.plot()`:

```
> rm.plot(y ~ time,  citrate, timelab
        = "time of observation", timelim = c(4,9))
```

If only a subset of individuals needs to be taken into account, the selection can be obtained by setting the argument `subset`, which defines a logical statement to be used to identify the individuals of interest:

```
> rm.plot(y ~ time,  citrate, subset = (x == "yes"))
```

`rm.plot()` also allows to analyze the response profiles in subgroups of individuals identified through the values of some covariates. The method can be invoked by specifying the covariates with respect to which the behaviour of the individual profiles are to be analyzed. The specification is made through a conditioning formula. This is a special kind of formula which in the standard S–PLUS model fitting functions uses the operator "|" to separate the predictor(s) from the conditioning variables. Analogously, `rm.plot()` uses a conditioning formula to identify, through the conditioning variables, subgroups of individual profiles of interest. Suppose we want to separately visualize the profiles of individuals belonging to the two categories of the covariate `x`. A call to the function, such as:

```
> rm.plot(y ~ time | x, data=citrate)
```

produces on the graphical window the plot shown in Figure 2.

The graphical layout presents a matrix of dependence panels, each one of which plots the profiles for the subgroup of individuals identified by the condition shown at the top of the panel. In the following, this graphical layout will be referred to as "panel plot". If the conditioning variable shows too many levels, so that the panel plot becomes unreadable, only the first dependence panel is shown on the graphical window.

3.2.1 Interactive mode

In exploratory analyses, when not much is known about the effects of covariates on the response variable, it is sometimes preferable to be able to highlight the relative "position" of single profiles or group of profiles on the whole set of profiles. To this end, `rm.plot()` also provides an interactive mode to analyze groups of profiles based on the highlighting

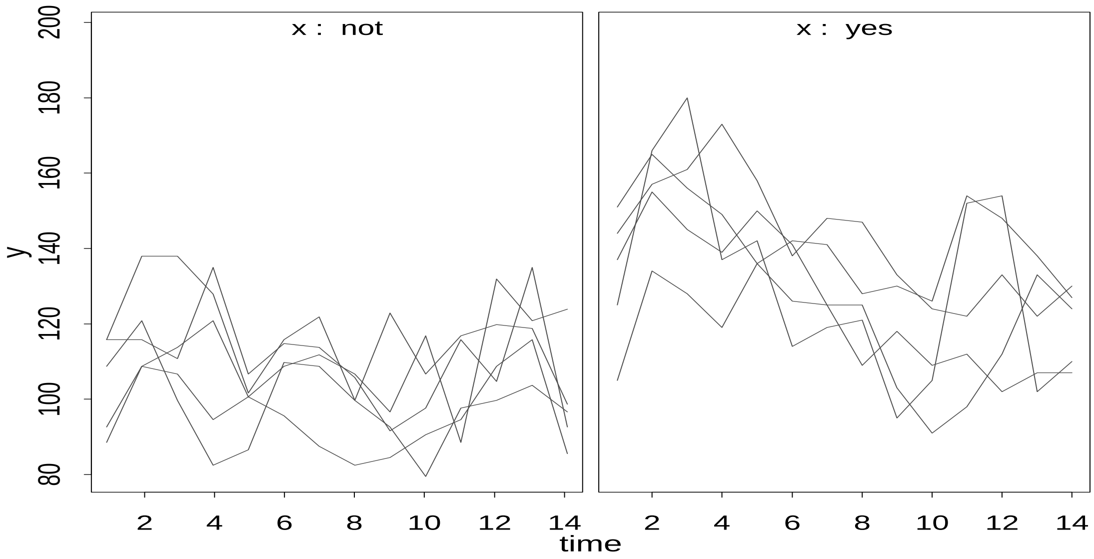


Figure 2: Standard panel plot layout for `rm.plot()`

(downlighting) of selected groups of individuals. To be visible, the interactive mode requires a color monitor and the current graphics window set in color mode. The interactive mode can be invoked by setting the flag `panel` equal to `FALSE` in the call to the function.

A call such as:

```
> rm.plot(y ~ time | x, data=citrate,panel=F)
```

produces on the plotting window a plot identical to that of Figure 1 and opens a dialog box containing the selecting levels for the covariate `x`. On the plot, profiles corresponding to the active selection for the covariate `x` are highlighted.

By subsequently selecting levels of the conditioning covariate, the profiles of the individuals satisfying the selection (if any) are highlighted in the plotting area. An example is given in Figure 3. See the documentation of `rm.plot()` for details on how to use the two interactive options `show unselected` and `hide unselected`.

If factors are used in the conditioning, their declared levels are used to build the possible selecting levels. If continuous variables are used, their quartiles are automatically used to split the variables into four grouping intervals, unless a different split is specified through the argument `given.values` (see also the S–PLUSdocumentation of the function `co.intervals()` which computes conditioning intervals for continuous variables).

The interactive mode can be exited by clicking the "quit" button of the dialog box, which causes the function `rm.plot()` to terminate and the S–PLUS session to continue.

Interaction is also useful to visualize and study single profiles. This can be done by introducing in the data list a factor containing an identifier of the observations and by conditioning the formula to the identifier, as shown below:

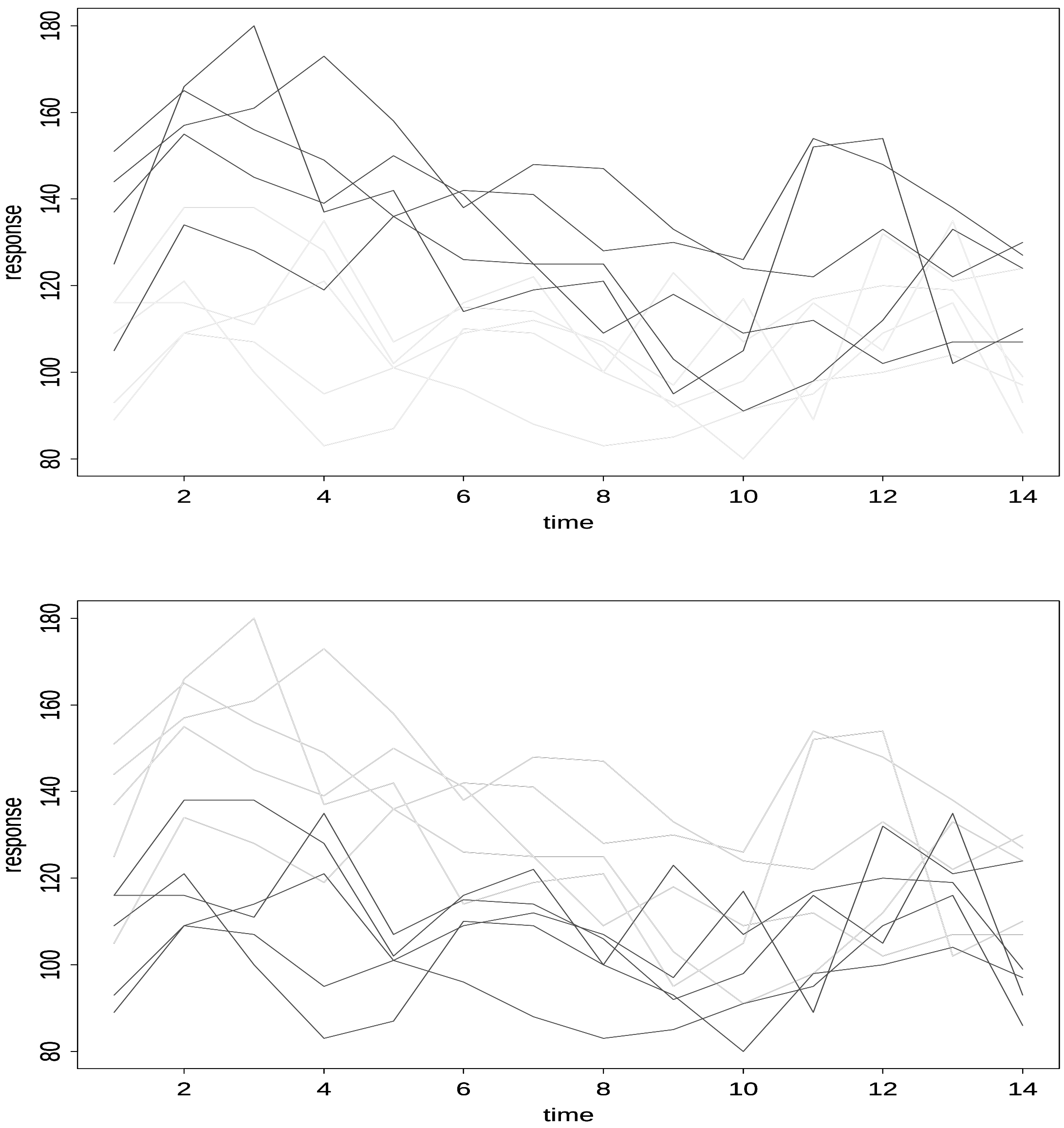


Figure 3: Subsequent highlighting of "yes" profiles and "not" profile in `hide unselected` mode

```
> id <- factor(1:10) # 10 profiles to be scanned through
> citrate$id <- id
> rm.plot(y~time|id, citrate,panel=F)
```

The dialog box now contains the labels of each profile, which can now be detected.

The interactive mode allows to extend the dependence analysis to two conditioning variables. Suppose to have another covariate `z` available in the "citrate" example. The conditioning formula could now be: `y ~time | x + z`. This would permit to study the response profiles in the groups identified through the combination of levels of both the covariates `x` and `z`. If `panel = T` (this is the default option), a panel plot is created showing the profiles for the groups identified through the levels of the first covariate. A dialog box allowing to select the levels of the second conditioning variable is also opened. On each dependence panel, profiles belonging to the intersection group are highlighted. If `panel` is set equal to `FALSE`, a single plot is created. The dialog box now allows to select the levels of both the conditioning variables and profiles corresponding to the intersection group are highlighted.

3.3 Analyzing mean profiles with `rm.mean()`

We have seen in § 3.2 how to explore individual profiles through the function `rm.plot()`. A second goal in the study could be to analyze the shape of mean profiles in the whole sample of individuals or in subgroups of individuals. This can be done by using `rm.mean()`, a function designed to estimate and plot mean profiles computed either by plain averages or by smoothing methods.

`rm.mean()` shows a lot of similarities with `rm.plot()`. In fact, `rm.mean()` requires the definition of a formula and a data list, as `rm.plot()` does, and all the optional parameters of `rm.plot()` can be seen to produce exactly the same effects with `rm.mean()`.

The call:

```
> rm.mean(y ~ time|x, citrate)
```

causes a panel plot to be created. On each dependence panel, the general mean profile and the mean profile for the subgroups identified through the levels of the covariate `x` are shown.

If the flag `se` is set to `TRUE` in the call to the function, the profile plot includes upper and lower pointwise confidence bands. By default, a 95% confidence band is computed; the band width can be modified by setting the parameter `coverage`, which permits specification of the confidence level of the limits as a fraction. An example of the panel plot layout for `rm.mean()` is shown in Figure 4. Note that the general mean profile is retained in each dependence panel.

Analogously to `rm.plot()`, `rm.mean()` can also be used interactively. The call:

```
> rm.mean(y ~ time|x, citrate,panel = F)
```

causes the general mean profile to be plotted on the plotting window of the `rm.mean()` layout. The usual dialog box containing the conditioning levels for the explanatory variable(s) is also presented on the screen. Profiles of the subgroups of individuals of interest can be identified through the levels menu and can be interactively plotted. During interaction, the mean profile

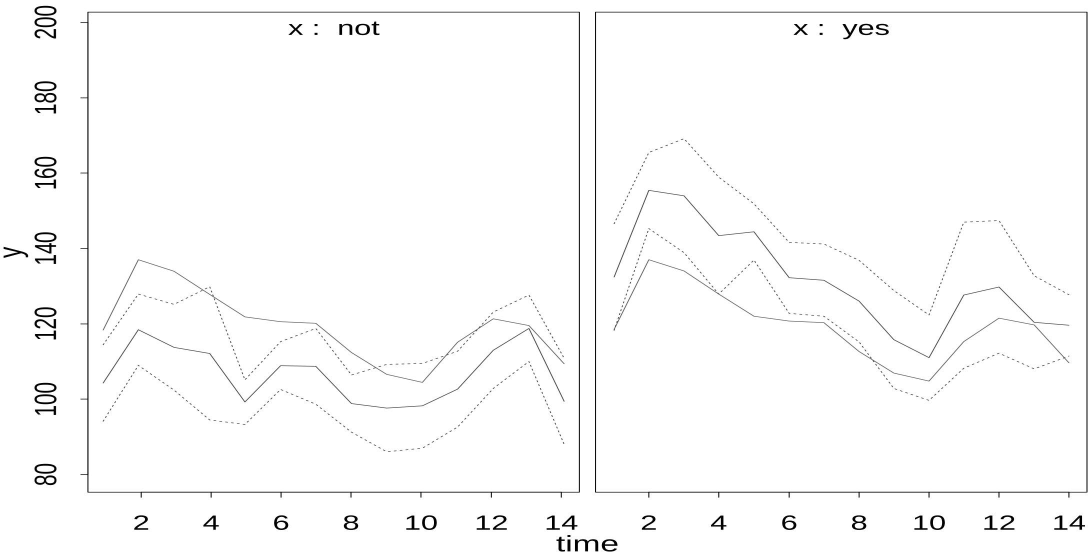


Figure 4: Graphical layout for the call `rm.mean(y~time|x, citrate, se=T)`

is retained in the plotting window. Two interactive options are available: `Single scan` and `Incremental scan`; see the documentation of `rm.mean()` for details on their effect on the interaction.

Although nothing is printed on the screen when `rm.mean` is exited, the function also provides numerical output. The output, in the form of matrices in a list, consists of the call to the function, of a time component, showing the time of estimation for each final profile value, a fit component, showing the estimated values for each final profile and, if `se = TRUE`, an upper component and a lower component, containing, respectively, the values of the pointwise upper and lower confidence limits. To be available and accessible, the output must be assigned to an S-PLUS object:

```
> citr.mean <-  rm.mean(y ~ time|x, citrate, se = TRUE)
```

The numerical output provided by the function is given by:

```
> citr.mean
$time:
          [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
grandmean    1    2    3    4    5    6    7    8    9    10    11
      yes    1    2    3    4    5    6    7    8    9    10    11
      not    1    2    3    4    5    6    7    8    9    10    11
.........

$fit:
```

```
            [,1]  [,2] [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
grandmean 118.5 137.0  134 127.9 122.0 120.7 120.3 112.6 106.9 104.8
      yes 104.6 118.6  114 112.4  99.6 109.2 109.0  99.2  98.0  98.6
      not 132.4 155.4  154 143.4 144.4 132.2 131.6 126.0 115.8 111.0
.........

$upper:
              [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
grandmean 131.6597 151.3357 150.7544 144.1766 137.7818 130.4846 130.8916
      yes 116.0647 129.3070 126.8841 132.3760 106.2813 116.3386 120.3490
      not 148.5691 166.9343 171.2395 161.0816 152.9229 142.9406 142.5836
.........

$lower:
               [,1]     [,2]     [,3]      [,4]      [,5]     [,6]
grandmean 105.34030 122.6643 117.2456 111.62336 106.21815 110.9154
      yes  93.13527 107.8930 101.1159  92.42401  92.91868 102.0614
      not 116.23089 143.8657 136.7605 125.71837 135.87709 121.4594
.........
```

The rows of the matrices labeled by `grandmean` contain the values for the general mean profile; each subsequent labeled row contains the values for the corresponding profiles identified through the conditioning; see the documentation of the function for details of the profiles identified in interactive mode.

We can easily extract the output matrices from `citr.mean` by the use of the extractor functions:

```
> citr.mean$fit
            [,1]  [,2] [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
grandmean 118.5 137.0  134 127.9 122.0 120.7 120.3 112.6 106.9 104.8
      yes 104.6 118.6  114 112.4  99.6 109.2 109.0  99.2  98.0  98.6
      not 132.4 155.4  154 143.4 144.4 132.2 131.6 126.0 115.8 111.0
.........
```

3.3.1 Smoothing mean profiles

Mean profiles can also be smoothed. At the present stage, two S–PLUS smoothing functions can be used within `rm.mean()`: `ksmooth()` and `rm.loess()`, which interfaces the standard S–PLUS `loess()` with `rm.mean()`. The declaration of the smoother to be used can be made in the call to `rm.mean()` by setting the input parameter `smoothing` equal to either of the two options `"ksmooth"` or `"loess"`. Suppose that the S–PLUS function `ksmooth()` performing smoothing using kernel estimates wants to be used. The call to `rm.mean()` can be:

```
> citrate.ksmooth <- rm.mean(y ~ time|x, citrate,
        smoothing = "ksmooth")
```

If some optional arguments for the smoother need to be specified or changed, the input parameter `control` can be used in the call to `rm.mean()`, as follows:

```
> citrate.ksmooth <- rm.mean(y ~ time|x, citrate,
        smoothing = "ksmooth", control = ksmooth.ctrl(n.points=20,
        kernel="normal",bandwidth = 2))
```

The function `ksmooth.ctrl()` modifies the default setting of the optional arguments of `ksmooth()`. Its input arguments can therefore be any of the optional arguments of the smoother which is used.

The numerical output in this case is given by:

```
> citrate.ksmooth
$time:
          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
grandmean    1 1.684211 2.368421 3.052632 3.736842 4.421053 5.105263
      yes    1 1.684211 2.368421 3.052632 3.736842 4.421053 5.105263
      not    1 1.684211 2.368421 3.052632 3.736842 4.421053 5.105263
.........

$fit:
              [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
grandmean 124.0016 129.8270 133.6017 132.7490 129.4317 125.6954 122.7862
      yes 108.7206 112.9036 115.1787 114.1658 111.5048 107.2934 104.6948
      not 139.2827 146.7505 152.0247 151.3322 147.3586 144.0975 140.8777
.........
```

and includes all the 20 points used by the smoother.

If the smoother also provides upper and lower confidence bands for the estimates, they can be included in the output by setting the `rm.mean()` input flag `se = TRUE`. If not, any setting of `se` or of `coverage` in the call to `rm.mean()` is ignored by the function.

Analogously:

```
> citrate.lsmooth <- rm.mean(y~time|x, citrate, se=T,
        smoothing = "loess", control=loess.ctrl(span=2/3,
        evaluation=20), coverage=0.90)
```

A plot of the smoothed mean profiles and their 90% confidence bands are shown in Figure 5

3.3.2 Batch use of `rm.mean()`

`rm.mean()` can be also used in a "batch" mode, to compute mean profiles of interest without passing through graphical displays. By setting the input argument `batch`, subgroups of individuals on which to compute the mean profiles can be selected:

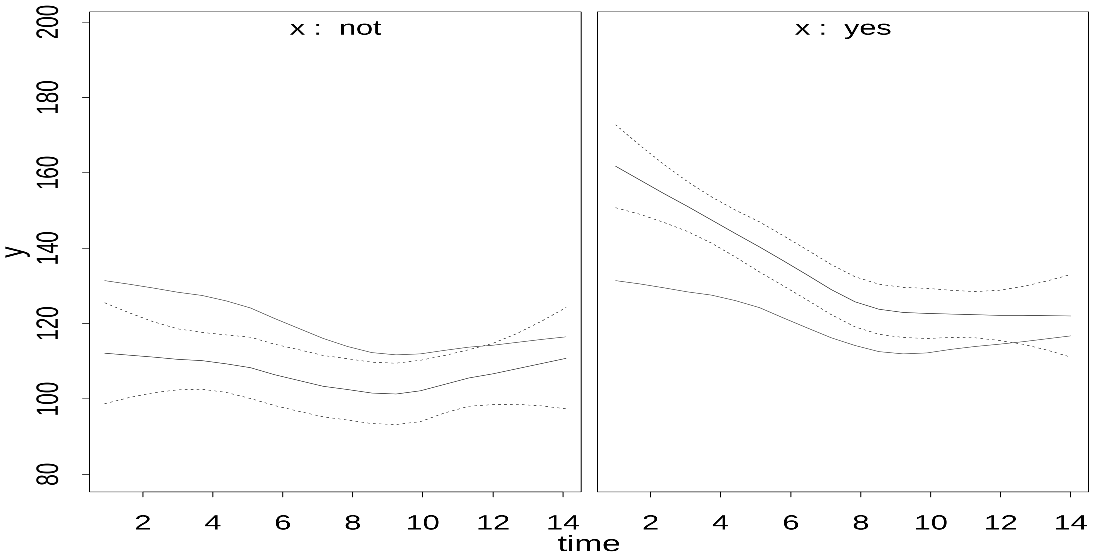


Figure 5: Graphical layout for the call `rm.mean(y ~time|x, citrate, se=T, smoothing = "loess", control=loess.ctrl(span=2/3, evaluation=20), coverage=0.90)`

```
> mean.p <- rm.mean(y~time, citrate, batch=list(expression(x=="yes"),
expression(x=="not")),se=T,smoothing="loess")
```

By default, results coming from the batch use of the function always contain the grand mean profile.

```
> mean.p
$time:
           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
 grandmean    1 1.265306 1.530612 1.795918 2.061224 2.326531 2.591837
       yes    1 1.265306 1.530612 1.795918 2.061224 2.326531 2.591837
.................
```

3.4 Fitting simple parametric models with `rm.fit()`

To compute a maximum likelihood fit of the model based on the assumption of normality of the observations, we shall use a statement of the form

```
> rm.fit(formula, data, ...)
```

where `formula` is the structural formula that specifies the model and `data` is the data list from which the model is to be fitted. The syntax of `formula` is exactly the same used to specify a linear model in `lm()`, described by Chambers & Hastie (1992), with the extra flexibility of allowing a form like:

```
y ~ v1 + v2
```

where `v1` is, say, subject specific and `v2` is time specific, i.e. covariates can be of different type and have different structure. Notice that `rm.fit()` uses the formula argument as any other modeling function does, i.e. as a symbolic expression. Therefore individual terms in `formula` can be any S-PLUS expression that, when evaluated, can be interpreted as a variable.

Three possible parametric families are considered in `rm.fit()`: Gaussian, binary and Poisson. The family can be selected by setting the parameter `family`. If `family` is not specified, the Gaussian family is assumed by default. In the remaining of this paragraph, we will concentrate on the normal case. Some other examples with normal data and with discrete responses are contained in § 4.

To perform the fit, `rm.fit()` requires the knowledge of the time component of the data list. By default, the program expects an object named `time` to be present in `data`; if the time component has been assigned a different name, the input parameter `time` should be set equal to the character value matching the name of the time component in the list.

In normal repeated measurements problems, a full specification of the model requires, besides the definition of the structural model, the identification of the covariance structure defining the dependence relationship model between replications. Such a definition can be made via the input arguments `cov.struct` and `random.effects`. The joint use of these two input parameters permits the selection of various covariance models, as described in the following and in § 4.

Four built-in structures for the covariance matrix are available by specifying `cov.struct`:

- `cov.struct = "I"`: independence;
- `cov.struct = "AR(1)"`: first order autoregressive structure;
- `cov.struct = "I+random"`: compound symmetry;
- `cov.struct = "AR(1)+random"`: first order autoregressive+compound symmetry.

If `cov.struct` is left unspecified, the `"I"` structure is assumed by default.

Using the citrate example, some simple calls to `rm.fit()` are

```
> rm.fit(y ~ x, citrate)

> rm.fit(log(y) ~ x, citrate, cov.struct="I=random")

> rm.fit(log(y) ~ factor(time), citrate, cov.struct="AR(1)")
```

The output of the function is contained in an object of class `rm`, for which methods for generic functions such as `print()`, `summary()`, `residuals()` and `fitted()` have been implemented. Whenever an object of class `rm` is printed, a summary is printed of the main quantities of interest of the rm object, e.g. the call to the function, the values of the coefficients in the linear predictor, their standard errors, the ratio between the coefficients and their standard errors and the estimated maximum for the likelihood. A message on the convergence of the optimizer is also added. Names of the coefficients are automatically constructed from the data and the formula and correspond to the names of the single-degree-of-freedom effects identified from the formula. The function `summary()` adds to the summary produced by `print()` the estimated values for the likelihood parameters, which depend on the parametric family and the matrix of correlations between the estimated parameters. One can assign the result of `summary()`, rather than printing it, in which case it produces a `summary.rm` object containing all the information printed. The object contains the call to the function, a matrix containing the estimated coefficients, their standard error estimates and their "t values", ie, the ratios of the estimated coefficients to their standard errors. In the binary and Poisson case, this matrix includes also the estimated quantities fot the likelihood parameter; in the Gaussian case, the likelihood parameters are contained in separate components. Moreover, the summary object contains the matrix of correlations between estimated parameters, the number of profiles used in the fit, the estimated maximum for the likelihood and the message from the optimizer.

```
> citrate.fit <- rm.fit(log(y) ~ factor(time), citrate,
               cov.struct="AR(1)")

> citrate.fit
Call:
rm.fit(formula = log(y) ~ factor(time), data = citrate.list, cov.struct =
"AR(1)")

Coefficients:
                      Value  Std. Error     t value
   (Intercept)  4.7667949471 0.028751465 165.7931156
 factor(time)1  0.0732830947 0.019089669   3.8388876
 factor(time)2  0.0154006454 0.013368206   1.1520353
 factor(time)3 -0.0044830084 0.010567861  -0.4242115
 factor(time)4 -0.0120560327 0.008804127  -1.3693615
 factor(time)5 -0.0078589920 0.007562915  -1.0391485
 factor(time)6 -0.0063263308 0.006632053  -0.9539023
 factor(time)7 -0.0133167764 0.005904260  -2.2554521
 factor(time)8 -0.0161603281 0.005318133  -3.0387220
 factor(time)9 -0.0146834739 0.004835445  -3.0366333
factor(time)10 -0.0039594079 0.004430906  -0.8935889
factor(time)11  0.0015104300 0.004086993   0.3695700
factor(time)12  0.0005768072 0.003791126   0.1521467

Log likelihood:  303.6193

Message:  RELATIVE FUNCTION CONVERGENCE

> summary(citrate.fit)
```

```
Call:
rm.fit(formula = log(y) ~ factor(time), data = citrate.list, cov.struct =
"AR(1)")

Number of profiles used in the fit:  10

Coefficients:
               Label         Value  Std. Error  t value
   (Intercept)     1  4.7667949471 0.028751465  165.793
 factor(time)1     2  0.0732830947 0.019089669    3.839
 factor(time)2     3  0.0154006454 0.013368206    1.152
 factor(time)3     4 -0.0044830084 0.010567861   -0.424
 ............

Within-subject estimated variance:  0.0249

Autoregressive parameter: 0.7068

Correlation of Coefficients:
        1      2      3       4       5      6       7      8       9
 1  1.0000  0.113  0.128  0.1294  0.1238  0.114  0.1013  0.085  0.0656
 2  0.1125  1.000  0.336  0.1192  0.0223 -0.026 -0.0514 -0.064 -0.0701
 3  0.1281  0.336  1.000  0.4619  0.2169  0.089  0.0176 -0.023 -0.0467
 4  0.1294  0.119  0.462  1.0000  0.5321  0.284  0.1422  0.058  0.0069
 ............

Log likelihood:  303.6193

Message:  RELATIVE FUNCTION CONVERGENCE
```

Matrices of the fitted values and of the residuals can be extracted from `rm` objects with the use of the extractor functions:

```
> citrate.fit$fitted
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
 [1,] 4.760911 4.907477 4.880396 4.831663 4.784832 4.785902 4.780913
 [2,] 4.760911 4.907477 4.880396 4.831663 4.784832 4.785902 4.780913
 [3,] 4.760911 4.907477 4.880396 4.831663 4.784832 4.785902 4.780913
 ...........

> citrate.fit$residuals
              [,1]         [,2]        [,3]        [,4]       [,5]
 [1,] -0.228311735 -0.216129535 -0.14419781 -0.03587239 -0.1697113
 [2,] -0.007321037 -0.153887226 -0.17086606  0.07361184 -0.1120030
```

```
[3,] 0.067402510 0.204510371 0.31256059 0.08831799 0.1709953
...........
```

If these values want to be used as new input values for one of the repeated measurements functions, like for example `rm.plot()`, the functions `fitted()` and `residuals()` provide a way of creating a data list suitable to the functions which includes the desired matrix of values (fitted or residual) as long as all the other variables appearing in the original data list. Although only the relevant values are printed when the functions are called, the resulting output objects can be seen to contain all the variables included in the starting data list except the response variable, which in turn is substituted by the new values:

```
> citrate.residuals <- residuals(citrate.fit)
             [,1]         [,2]        [,3]        [,4]       [,5]
 [1,] -0.228311735 -0.216129535 -0.14419781 -0.03587239 -0.1697113
 [2,] -0.007321037 -0.153887226 -0.17086606  0.07361184 -0.1120030
 ..........

> citrate.residuals
$x:
 [1] "yes" "yes" "not" "not" "not" "yes" "yes" "yes" "not" "not"
$x: Levels:
[1] "yes" "not"

$time:
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14

$residuals:
             [,1]         [,2]        [,3]        [,4]       [,5]
 [1,] -0.228311735 -0.216129535 -0.14419781 -0.03587239 -0.1697113
 [2,] -0.007321037 -0.153887226 -0.17086606  0.07361184 -0.1120030
 ...........
```

3.4.1 Analysis of Deviance Tables

After fitting a model like the above one, with output called `citrate.fit`, say, an analysis of variance table can be produced using `anova()`. When applied to the object produced by `rm.fit()` for Gaussian data, this command performs a suitably modified ANOVA where the usual sums of squares are replaced by analogous quantities which accounts for the presence of correlation among the data.

The reason for using such modified 'sum of squares' is to achieve a χ^2 with appropriate degrees of freedom even in the presence of autocorrelation among the data. The χ^2 distribution would be exact if ρ were know exactly; when an estimate is used in its place, some degree of approximation is involved, but it is asymptotically negligible. When $\rho = 0$, this ANOVA coincides with the traditional one.

The theoretical background and details for this modified ANOVA table are provided by Azzalini (1984). The present calculations differ from those in the above reference for the inessential multiplication of all 'sum of squares' by a factor $1/(1-\rho^2)$, due to a minor difference in the parameterization; however, when F ratios are computed, the multiplicative constant disappears.

For the present example, the command

```
> anova(citrate.fit)
```

gives:

```
> anova(citrate.fit)
Analysis of Deviance Table

Response: y

          Terms  Df Sum.of.Sq  Mean.Sq
1   (Intercept)   1  731.3357 731.3357
2  factor(time)  13    0.9173   0.0706
3      Residual 126    3.4800   0.0276
4         Total 140  735.7330   5.2552
```

4 Additional options of `rm.fit()`

4.1 Fitting more complex normal models

A random-effects structure can be added to the last two options (`I+random` and `AR(1)+random`) of `cov.struct` by specifying the optional argument `random.effects`. The argument of the latter expects a formula, which specifies the model for the random components on each individual observation:

```
> rm.fit(log(y) ~ factor(time), citrate,
         cov.struct="AR(1)+random", random.effects=log(y)~x)
```

This will create the matrices F's of known coefficients loading the random effects on the linear predictor in the model for each subject. As `random.effects` can contain time-dependent covariates, the resulting F's are allowed to vary across subjects. An illustrative example of this type will be presented in § 5.1.

Excluding the independence case, which reduces the problem to a standard linear model fit, in any other case starting values for the optimization task can be defined through the argument `start`, which is passed by `rm.fit()` to the optimizer. The values in `start` refer to the correlation parameters and not to the parameters of the model predictor. Therefore, the structure of the vector `start` should be:

- a vector of length 1 when `cov.struct = "AR(1)"` or `cov.struct = "I+random"`, representing the initial value for either ρ or δ, respectively;

- a vector of length 2 when `cov.struct = "AR(1)+random"`: the first component should represent the starting value for ρ and the second component that of δ;
- a vector of length k when `cov.struct = "I+random"` and `random.effects = ...`; here, k represents the number of unknown variances for the random effects and `start` their initialization;
- a vector of length $k+1$ when `cov.struct = "AR(1)+random"` and `random.effects = ...`; here, the first component should represent the starting value for ρ and the following components the starting values for the variances of the random effects.

Missing values and unbalanced data are automatically accounted for. `rm.fit()` tries to use all the significant information from each observation by dropping any missing values and by creating for the individuals used in the fit the appropriate model matrices. Profiles containing any missing values in a subject specific covariate are ignored. missing values on time costant predictors are not considered in the fit. Any different missing value action must be applied to the data list before `rm.fit()` is called.

4.2 Fitting binary data

To perform a fit on binary data, the input parameter `family` must be set equal to `"binary"` in the call to the function.

The next example refers to data reported by Fitzmaurice, Laird and Lipsitz (1994), on the "Muscatine coronary risk factor study". The data set contains record on 1014 children who were 7–9 years old in 1977 and were examined in 1977, 1979 and 1981. Height and weight were measured in each survey year, and those with relative weight greater that 110% of the median weight in their respective stratum were classified as obese. The binary response is whether the child is obese (1) or not (0).

Data are read in and inserted in a list by the following statements.

```
> musc <- read.table("muscatine.dat")
> freq <-  musc[,1]
> sex <- musc[,2]
> obese <- musc[,3:5]
> names(obese) <- c("1977","1979","1981")
> musc <- list(obese=obese, time=1:3, counts=freq, sex=factor(sex))
```

Notice that, in this case, each distinct response profile is replicated several times, in general. A corresponding vector, called `counts`, is created accordingly.

A simple logistic model is fitted by a statement like

```
> fit.bin <- rm.fit(obese~time+(time^2)+sex, musc, family="binary")
```

which fits a model quadratic in time; here the new feature is the use of the parameter `family="binary"`. This parameter causes `rm.fit()` to fit a logistic models of the type described in § 2.2.

The presence of the vector counts is detected, and this vector is used for weighting the response profiles. If the vector of counts had a different name, freq say, then the parameter counts="freq" must be inserted in the calling statement. In other words, the treatment of counts is similar to the use of time in this respect.

If we wish to decompose the time effect in orthogonal components, the following are the appropriate statements

```
> musc$Time1 <- c(-1,0,1)
> musc$Time2 <- c(1,-2,1)
> fit.bin2 <- rm.fit(obese~Time1+Time2+sex, musc, family="binary")
```

To initialize the optimizer, a single value is required for start, representing the initial guess for the log-odds ratio, as in

```
> fit.bin3 <- rm.fit(obese~Time1+Time2, musc, family="binary",
              start=3, control=NULL)
```

Missing values are allowed on the response, provided they are missing at random. Although the theoretical approach has no restriction on the their patterns, the present implementation requires that only one missing value should appear between two significant observations. In practice, the program performs the fit even when the condition is not fulfilled, but a warning message is printed to advice on the possible inaccuracy of the results. NA's on covariates are not allowed. At the present stage, no random effects are allowed for binary data.

4.3 Fitting Poisson data

To perform a fit on Poisson data, the input parameter family must be set equal to "poisson" in the call to the function:

```
> rm.fit(response~lin.predictor, epi.list, family="poisson")
```

This fits a model of the type described in § 2.3. We remind the reader of the rather specialized nature of this model, which implies a suitable type of behaviour on the observed data. Therefore, it cannot be regarded as a "general purpose" scheme, like the models described in the previous sections.

If the data list contains a table and the counts' vector is not named counts, then the input parameter counts should be set equal to the string representing the name of the counts' vector in the call to rm.fit().

By default, the program utilizes the canonical link for Poisson data, i.e. the logarithmic transformation. Alternatively, the identity link can be selected by setting the input argument link to be equal to "identity" in the call to the function, like

```
> rm.fit(response~lin.predictor, epi.list, family="poisson", link="identity")
```

At the present stage, no random effects are allowed for Poisson data. To initialize the optimizer, a single value is required for `start`, representing the initial guess for the autocorrelation parameter ρ.

Missing values and unbalanced data are automatically accounted for. `rm.fit()` tries to use all the significant information from each observation by dropping any missing values and by creating for the individuals used in the fit the appropriate model matrices. Observations showing A different missing value action must be applied to the data list before `rm.fit()` is called.

4.4 Advanced use of `rm.fit()`

The design matrices, which might vary from individual to individual, are created by `rm.fit()` with the use of the function `model.matrix()`; see the documentation of this function for details. Contrasts for the factors appearing in the formula are automatically created by `model.matrix()` using the Helmert parametrization. The optional argument `contrasts` can be used to modify this default. Supposing that in the " citrate" example of § 3.4 the standard contrasts based on the constraint that the sum of each column of the model matrix must be zero want to be used for `factor(time)`. Since `factor(time)` becomes a categorical variable with 14 possible levels, the required parametrization can be obtained by the call:

```
> rm.fit(y~factor(time), citrate,
              contrast= list(time = contr.sum(14)))
```

See the documentation of `rm.fit()` or of any other S-PLUS modeling function for details on the use of the contrast argument.

Excluding the Gaussian independence case, which reduces to a standard linear model fit, in any other case a non linear maximization of the likelihood is performed by `rm.fit()` using the S-PLUS optimizer `nlminb()`. Unless the argument `trace` is set to `FALSE`, the intermediate values of the likelihood parameters and of the likelihood itself are printed on the screen to make possible to follow the optimization process. Control parameters of the optimizer can be passed to the function `nlminb()` through the `rm.fit()` input argument `control`; see the documentation `nlminb.control()` for details. By default, the parameters `x.tol` and the `rel.tol` of the optimizer have been fixed respectively to 0.001 and 0.001; this choice can be modified by assigning the `control` argument to a list whose elements are the desired values for all the control parameters of interest:

```
> rm.fit(..., control=nlminb.control(abs.tol=10^(-6)))
```

Any setting of the parameter `control` cancels both the default values.

Warning. Since `nlminb()` does not have an option for stopping on small absolute variations near the maximum (which is what is really needed for maximum likelihood estimation), it is difficult to choose default values of the available options which are appropriate for all situations. In some cases, it may be necessary to suitably modify the default values for the given problem.

The default fit includes all the individuals; restricted fits on subgroups of individuals can be obtained including the argument `subset` in the call to the function. `subset` is used by `rm.fit()` with the same syntax and effects that were seen to be valid for the `rm.plot()` and `rm.mean()`.

If a `subset` condition has been used in the call to `rm.fit()`, the restricted set of values used in the fit is included in the list components created by `fitted()` and `residuals()`.

```
> rm.fit(y ~ factor(time), citrate, subset = (x == "yes"))
```

5 Additional examples

5.1 Pothoff & Roy data

Pothoff & Roy (1964) present data on measurements of the distance from the pituitary to the pterygomaxillary fissure at the age of 8, 10,12, 14 years for 16 boys and 11 girls. For use with `rm.tools`, a list is created with

```
> PR.list <- list(y=matrix(scan("PR.dat"),byrow=T,ncol=4),
             age=c(8,10,12,14),time=1:4,
             sex=factor(c(rep("M",16),rep("F",11))))
```

and plotted with

```
> rm.plot(y~age|sex,PR.list)
```

which produces Figure 6.

Notice that two nearly equivalent vectors are introduced to describe time, namely `age` and `time`. While the first one is the natural time scale, the second one is required to inform `rm.fit()` that the span of time between, say, 8 and 10 years is to be considered as one time unit, and not two. This aspect may become relevant if an autoregressive dependence structure is to be fitted, as this vector is used by `rm.fit()` to construct the correlations matrix Ω.

From inspection of Figure 6, it seems sensible to regard `age` as a continuous variable with linear effect. Therefore, we fit the following two models

```
> PR.fit1 <- rm.fit(formula = y ~ age + sex, data = PR.list,
             cov.struct = "I+random")
> PR.fit2 <- rm.fit(formula = y ~ age + sex, data = PR.list,
             cov.struct = "AR(1)+random")
```

A significance test for the presence of the autoregressive component is given by the likelihood ratio test statistic $2(\ell_2 - \ell_1)$, obtaining

```
> 2*(PR.fit2$log.lik-PR.fit1$log.lik)
[1] 0.07476632
```

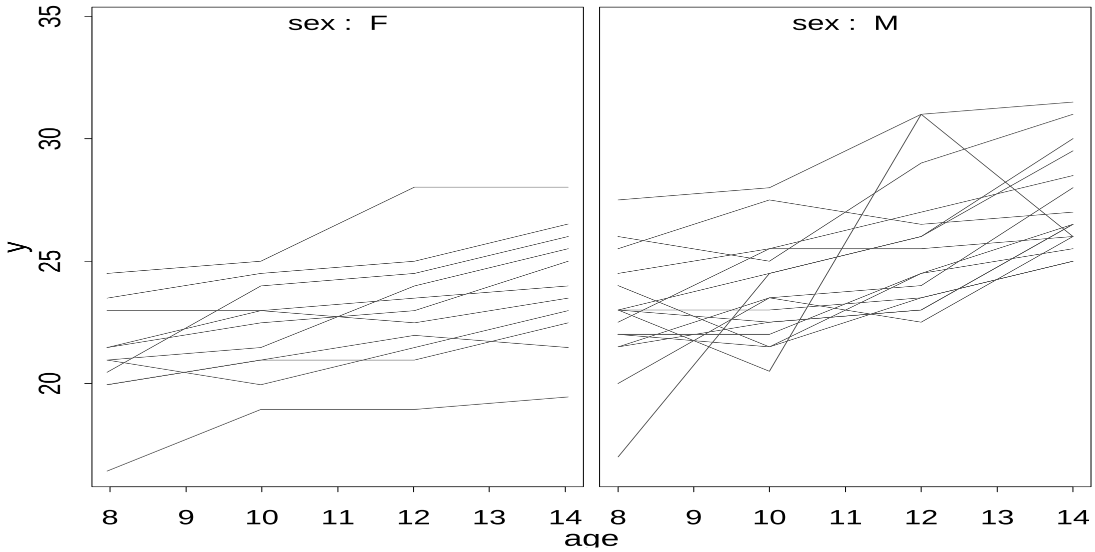


Figure 6: Plot of the data of Pothoff & Roy (1964)

which is so small that a comparison with percentage points of the χ^2_1 distribution is not required, leading to definitive omission of the AR(1) component.

There are two possible directions for extending the second model introducing: (a) interaction between age and sex, (b) a random effect on the individual slope (which is often referred to as a growth curve model). We consider these additions in steps, obtaining:

```
> PR.fit3 <- rm.fit(formula = y ~ age * sex, data = PR.list,
             cov.struct = "I+random")
```

Similarly to the previous case, a formal test statistic for the significance of the interaction is computed as

```
> p <-  2*(PR.fit3$log.lik-PR.fit1$log.lik)
> p
[1]  5.690903
> 1-pchisq(p,1)
[1] 0.01705308
```

which leads to maintaining the interaction term.

The random effect effect component on the individual slope is added with

```
> PR.fit4_rm.fit(y~age*sex,PR.list,cov.struct="I+random",
                 random.effects=y~age)
```

and the likelihood ratio test for the significance of the random effect of the linear component is

```
> p <- 2*(PR.fit4$log.lik-PR.fit3$log.lik)
> p
[1] 0.6568439
> 1-pchisq(0.6568439,1)
[1] 0.4176764
```

which is in favour of model 3.

5.2 Locust data

MacDonald and Raubenheimer (1995) analyze the effect of hunger on locomotory behaviour of 24 locusts (*Locusta migratoria*). The experiment, carried out in the Zoology department of the university of Cape Town, is described by the authors as follows.

> The locusts were placed individually in glass observation chambers. The even-numbered subjects were deprived of food for $5\frac{1}{2}$ hours before observation commenced, during which period the odd-numbered subjects were allowed to feed ad libitum. Food and water were not available during observation. Within each of the two groups, the subjects were alternatively male and female. During the observation period, data were entered directly into a computer, which had been programmed to emit a signal at thirty-second intervals and thereafter to accept an entry for the observed behaviour of each of the subjects: locomoting, quiescent or (rarely) grooming. The number of observations made on each subject was 161. For the purpose of this analysis the categories used were 'locomoting' (denoted by 1) and 'not locomoting' (0), the latter category including both quiescent and grooming behaviour.

Data are read in and arranged in suitable form by the following instructions:

```
> locust <- matrix(scan("locust.dat"),byrow=F,nrow=24)
> feed <- rep(c("yes","no"),12)
> sex <- rep(c("M","M","F","F"),6)
> id <- 1:24
> locust.list <- list(move=locust,sex=factor(sex),feed=factor(feed),
              id=factor(id),time=1:161)
```

An overall look at the data can be obtained through an individual scan of the profiles:

```
> rm.plot(move~time|id,locust.list)
```

Because of the number of subjects and of the binary nature of the response, the resulting panel plot is not particularly effective. Nevertheless, the plot seems to highlight some features related to the covariate `feed`: even-numbered, ie not-fed, locusts tend to move around much more frequently then fed locusts.

Following this suggestion, we decide to explore the mean profiles in the two groups identified by the variable `feed`. In order to be able to compare the two mean profiles more easily, we decide to utilize a single plot, instead of a panel plot, and to superimpose plots of the two mean profiles by using the `Incremental scan` option in interactive mode.

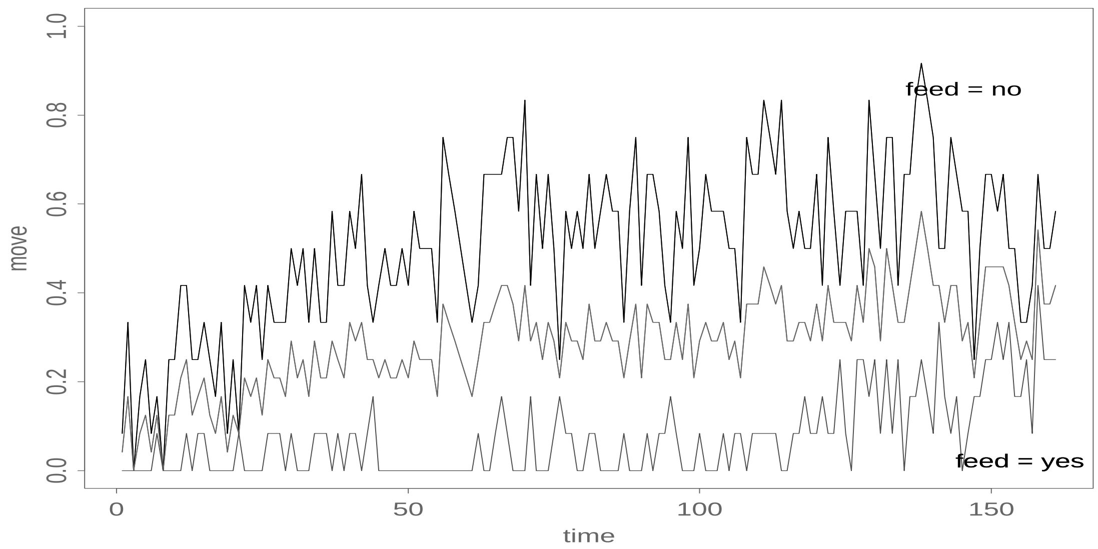


Figure 7: Mean profiles by `feed` for the locust data. Labels on the profiles have been added to the plot after exiting `rm.mean()` through instructions of the type: `text(locator(), "feed = no")`

```
> rm.mean(move~time|feed,locust.list, panel=F)
```

The resulting plot is shown in Figure 7. Clearly, the two mean profiles differ, although the plot is still quite rough. A considerable improvement can be achieved by smoothing the profiles. As we may need to use the nonparametric fit for further analysis, we save the output of `rm.mean()` by assigning it to an object.

```
> loc.mean <- rm.mean(move~time|feed,locust.list,smoothing="loess",
          control=loess.ctrl(evaluation=20),panel=F)
```

as shown in Figure 8.

Further analysis, not reported here, has been conducted including `sex` as an additional conditioning factor, leading to the conclusion that `sex` can be omitted from the subsequent analysis, at least at this level of analysis.

The shape of the smoothed plot in Figure 8 suggests a quadratic relationship between time and probability of locomotion, within each treatment group. Fitting of a parametric model of this kind is obtained by:

```
> loc.fit.no <- rm.fit(move~time+time^2,locust.list,family="binary",
          subset=(feed=="no"))
> loc.fit.yes <- rm.fit(move~time+time^2,locust.list,family="binary",
          subset=(feed=="yes"))
```

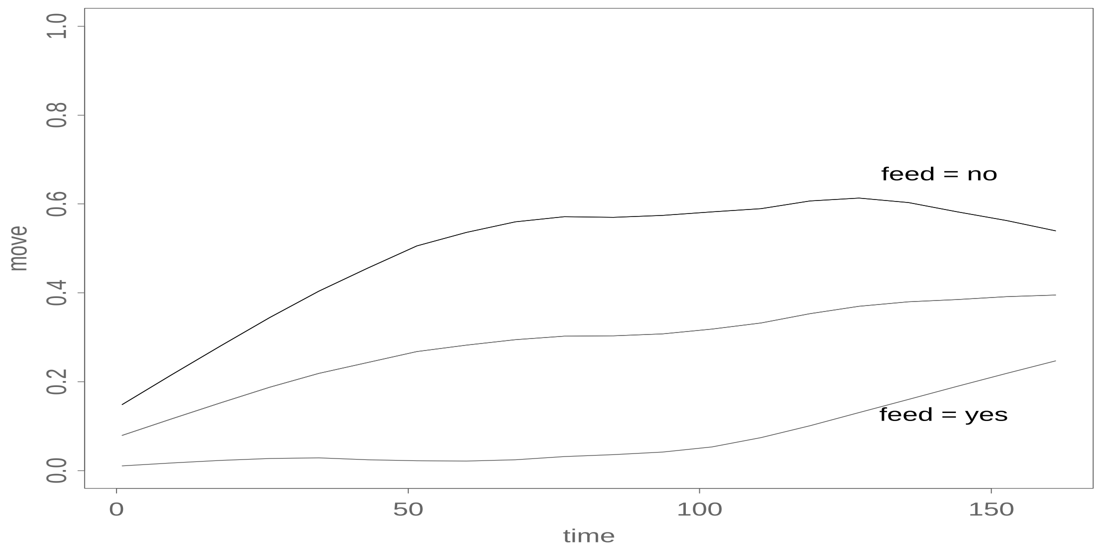


Figure 8: Smoothed mean profiles by `feed` for the locust data.

The fitted models can now be plotted and compared with the nonparametric estimates obtained through the smoothing of the mean profiles. To obtain a plot of the fitted models, the following commands can be used:

```
> support <- seq(1,161,length=20)
# abscissae used by the smoother in rm.mean()
> plot(support, loc.mean$fit[1,], ylim=c(0,1), type="l"
+  xlab="time", ylab="Prob(locomoting)")
# plot the general mean profile
> lines(support,loc.mean$fit[2,],type="l")
> lines(support,loc.mean$fit[3,],type="l")
# add the two group mean profiles
>  lines(locust.list$time,loc.fit.yes$fitted[1,],type="l",lty=3)
>  lines(locust.list$time,loc.fit.no$fitted[1,], type="l",lty=3)
# add the parametric fits
```

which produce Figure 9.

6 Software availability

Software is available from the archive server StatLib and from the `ftp` server of the Department of Statistical Sciences, University of Padua. For first kind of access, send to `statlib@lib.stat.cmu.edu` the e–mail message

```
send rm.tools2 from S
```

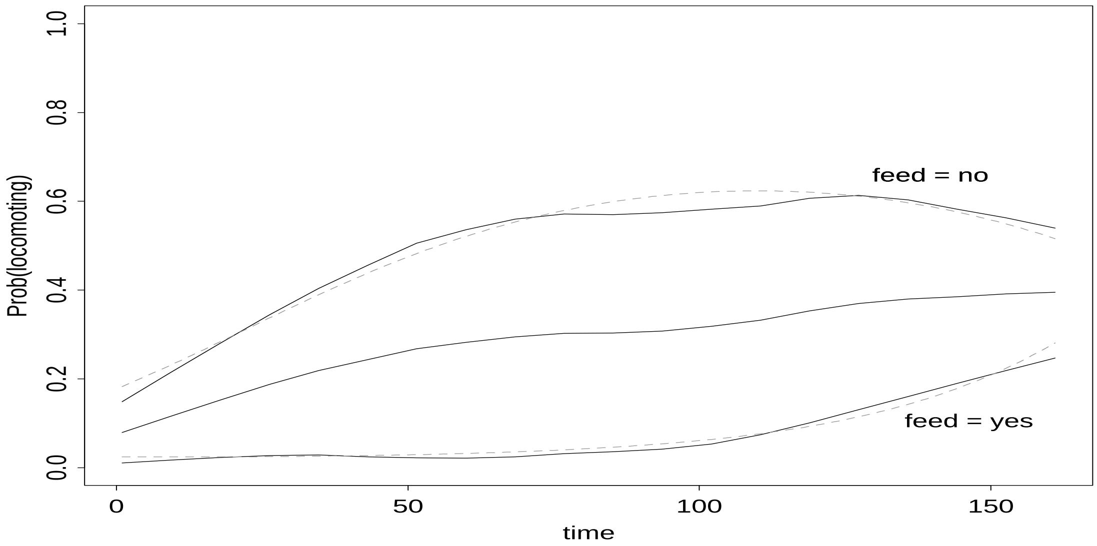


Figure 9: Comparison of the nonparametric and parametric fit for the locust data.

For the second server, connect as `anonymous` user to `ftp.stat.unipd.it` and get the content of `/users/azzalini/rmtools/`. In either cases, refer to the file `Install` for instructions on installation.

The programs have been developed and tested on a Sun workstation running SunOS 4.1.3, using S–plus version 3.2. Note that some Fortran 77 routines, compiled using Fortran SC 2.0, have been used in `rm.fit()` for some heavy computations in the binary and Poisson case. Therefore, a Fortran compiler is required to make `rm.fit()` work in these two cases. The other functions (`rm.plot()` and `rm.mean()`) and `rm.fit()` in the Gaussian case work without making use of `Fortran` supporting functions; therefore they are usable without a Fortran compiler.

If the installation had been successful, the module `rm.tools` is made available in a S–plus session through the command:

```
> library(section="rmtools", lib.loc="/home/xyz/mylibrary")
```

where `/home/xyz/mylibrary` is the directory chosen to contain the library "rmtools". A more detailed description is contained in the file `Install`. One may wish to insert the above given command in the `.First()` function.

The directory `/home/xyz/mylibrary` can be any directory with writing permission for the user. To make the module accessible to any S–plus user, choose the directory `$SHOME/library`. The Unix address of the `$SHOME/library` directory is obtained by typing `Splus HOME`. Ask your system manager if you do not have permission to write in this directory.

Feedback from users is welcome. Please send any comments to either authors; to ease communication, e–mail is the preferred communication medium.

Acknowledgments

We like to thank Ian MacDonald and to David Raubenheimer for providing us with the locust dataset. Financial support for this research has been provided by Consiglio Nazionale delle Ricerche, Italy, under grant no. 91.04081.CT10.

References

A.H.Andersen, E.B.Jensen, G.Schou (1981). Two–way analysis of variance with correlated errors. *Int.Statist.Rev.* **49**, 153-167.

A.Azzalini (1984). Estimation and hypothesis testing for collections of autoregressive time series. *Biometrika* **71**, 85-90; correction **74** (1987), 667.

A.Azzalini (1994). Logistic regression for autocorrelated data with application to repeated measures. *Biometrika* **81**, 767-775.

A.Azzalini (1994). Logistic regression and other discrete data models for serially correlated observations. *J. Ital. Stat. Soc.*, **3**, 169-179.

R.A.Becker, J.M.Chambers & A.R.Wilks (1988). *The new* S *language.* Pacific Grove: Wadsworth & Brooks.

J.M.Chambers & T.J.Hastie (1992). *Statistical models in* S. Pacific Grove: Wadsworth & Brooks.

M.J.Crowder, and D.J.Hand (1990). *Analysis of Repeated Measures.* London: Chapman & Hall.

P.J.Diggle (1988). An approach to the analysis of repeated measurements. *Biometrics*, **44**, 959-971.

P.J.Diggle, K.Y.Liang & S.L.Zeger (1994). *Analysis of Longitudinal Data.* Oxford: Oxford University Press.

G.M.Fitzmaurice, N. M. Laird & S.R.Lipsitz (1994). Analysing incomplete longitudinal binary responses: a likelihood based approach. *Biometrics*, **50**, 602–612.

C.A.Glasbey (1979). Correlated residuals in non–linear regression applied to growth data. *Appl. Statistics* **28**, 251-9.

N.M.Laird and J.H.Ware (1982). Random–effect models for longitudinal data. *Biometrics* **38**, 963-974.

J.K.Lindsey (1993). *Models for Repeated Measurements.* Oxford: Oxford University Press.

I. MacDonald and D. Raubenheimer (1995). Hidden Markov models and animal behaviour. *Biometrical Journal*, **37**. 701–712.

E. Mc Kenzie (1985). Some simple models for discrete variate time series. *Water resources Bulletin* **21**, 645-650.

Statistical Sciences, Inc. (1993) *S–plus User's manual, version 3.2*. Seattle: Statistical Sciences, Inc.

J.H.Ware (1985). Linear models in the analysis of longitudinal studies. *The American Statistician* **39**, 95-101.

B.J.Winer (1971). *Statistical principles of experimental design* (2nd edition). New York: Wiley.

Table 3: Examples of some of the most frequently used commands

<table>
<tr><th>Description</th><th>Example(s)</th></tr>
<tr><td>creation of a data list</td><td><code>citrate <- list(y=matrix(response,10,14), time=1:14,</code>
<code>   group=factor(c(-1,-1,1,...)),id=factor(1:10))</code></td></tr>
<tr><td>plotting individual profiles</td><td><code>rm.plot(y~time|group, citrate)</code>
<code>rm.plot(y~time|id,citrate)</code></td></tr>
<tr><td>plotting mean profiles with
poitwise confidence bands</td><td><code>a <- rm.mean(y~time|group, citrate, se=T)</code></td></tr>
<tr><td>plotting smoothed mean profiles</td><td><code>a <- rm.mean(y~time|group, citrate,</code>
<code>   smoothing = "loess", se=T)</code></td></tr>
<tr><td>as above, with another smoother</td><td><code>a <- rm.mean(y~time|group, citrate,</code>
<code>   smoothing = "ksmooth")</code></td></tr>
<tr><td>simple two-factor for normal
autocorrelated data</td><td><code>fit <- rm.fit(log(y)~factor(time)+group,</code>
<code>   citrate, cov.struct="AR(1)")</code></td></tr>
<tr><td>as in the previous case, adding
individual random effects</td><td><code>fit <- rm.fit(log(y)~factor(time)+group,</code>
<code>   citrate, cov.struct="AR(1)+random")</code></td></tr>
<tr><td>ANOVA for output of either of
previous two cases</td><td><code>anova(fit)</code></td></tr>
<tr><td>a linear growth curve model
with random effects</td><td><code>fit <- rm.fit(y~sex+time, PR.list,</code>
<code>   cov.struct="I+random", random.effects=y~time)</code></td></tr>
<tr><td>fitting logistic regression
to autocorrelated binary data</td><td><code>fit <- rm.fit(obese~time+(time^2), musc,</code>
<code>   family="binary")</code></td></tr>
<tr><td>examining the estimated model</td><td><code>print(fit) summary(fit)</code></td></tr>
<tr><td>obtaining fitted values</td><td><code>fitted(fit)</code></td></tr>
</table>

Appendix
S Functions Documentation

This appendix contains the documentation for the RM functions. Documentation for some standard S–PLUS functions, used in connection with the described RM functions, can be found in Becker, Chambers & Wilks (1988) and Chambers & Hastie (1992).

```
rm.plot(formula, data, subset, given.values, timelab, ylab,
timelim, ylim, panel = TRUE, ...)
```

REQUIRED ARGUMENTS

`formula`: a formula expression of the form `response` ~ `time (| covariates)` as for regression models. At most two covariates can be used in the predictor, separated be a `+` sign.

`data`: a data list in which to interpret the variables occurring in the formula. List elements are identified by a name. NA values are allowed. See the documentation of `rm.fit()` for details.

OPTIONAL ARGUMENTS

`subset`: expression stating which subset of the data rows should be used in the plot and fit. This is a logical statement of the type `covariate1 == "a" & covariate2 > x` which identifies the observations to be selected. The default selection is "all observations".

`given.values`: a numeric vector, character vector, or two- column matrix that specifies the given values when there is one given numerical predictor, or a list of two such objects when there are two. If missing, reasonable things happen.

`timelab`: time axis labels. If missing, the name of the variable in `formula` is used.

`ylab`: response axis labels. If missing, the name of the variable in `formula` is used.

`timelim`: approximate minimum and maximum values to be put on time axis.

`ylim`: approximate minimum and maximum values to be put on the response axis.

`panel`: a flag that determines the method of plotting on the graphical window. The default is TRUE.

Graphical parameters may also be supplied as arguments to this function (see `par`).

SIDE EFFECTS

a plot of the individual profiles is produced on the current graphics device. If interaction is invoked, dialog boxes showing scrolling lists of all the levels of the covariates (when used) are also presented. Levels in the scrolling lists can be interactively 'selected' and 'deselected'.

DETAILS

LAYOUT. There are two main layouts for `rm.plot()`, depending on the value of the flag `panel`. If `panel` is TRUE, then a matrix of dependence panels is created with as many panels as the number of levels for the first conditioning variable.

On each dependence panel, profiles are plotted for the subjects showing the level of the covariate written at the top of the panel. If `panel` is FALSE, then a single plotting window is created.

DIALOG BOX. A dialog box is present whenever:

- `panel` is FALSE and covariates appear in `formula`;
- `panel` is TRUE and two covariates appear in `formula`.

The upper part of the box contains a menu showing the levels of the covariates on which interaction is activated. Labels of the covariates' levels as declared in the data list are used to identify the levels; intervals are used for numerical covariates, when not otherwise stated (see `given.values` and `co.intervals()`). Two different interactive options can be chosen: `Show unselected` or `Hide unselected`. A radiobox contains the selection for the interactive option; the active option is shown by the pressed button. The selection can be toggled at any time during interaction simply by clicking (the selection button) on the desired mode and the two options may be switched any number of times. The lower part of the dialog boxes is used to stop interaction. To exit, place the cursor over the button labeled "Quit" and press the left mouse button. The `rm.plot()` layout will remain, and a prompt will appear in the S–PLUS command window. No other S–PLUS commands are evaluated until `rm.plot()` is exited.

GRAPHICAL INTERACTION

Clicking on a level highlights the individual profiles of the subset of cases (if any) which present the selected level. If this subset is empty, a new graphics input is expected. In the `show unselected` mode, whenever a new level is selected, the new individual profiles identified by the selection are highlighted while the remaining profiles are kept downlighted; in the `hide unselected mode`, only the last identified profiles are shown. If two covariates are used when `panel` is TRUE, profiles of cases that satisfy the active selection on the second covariate are highlighted on each dependence panel. If two covariates are used when `panel` is FALSE, profiles of cases that satisfy the active selections on the both the covariates are highlighted. The selection process may be repeated any number of times. No graphical interaction is possible if covariates are missing.

VALUE

No numerical output is provided.

SEE ALSO

`rm.mean(), rm.fit()`

EXAMPLES

```
rm.plot(y~time, data = citrate)

rm.plot(log(y)~time|soil+site, soil.list, subset = moisture < 45,
ylim = c(1,2))
```

```
rm.mean(formula, data, counts, smoothing, control=NULL,
batch=NULL, subset, given.values, conf.int=FALSE, timelab, ylab,
timelim, ylim, panel=TRUE,...)
```

REQUIRED ARGUMENTS

`formula`: a formula expression of the form `response` $\sim$ `time (| covariates)` as for regression models. At most two covariates can be used in the predictor, separated by a `+` sign.

`data`: a data list in which to interpret the variables occurring in the formula. List elements are identified by a name. NA values are allowed. See the documentation of `rm.fit()` for details.

OPTIONAL ARGUMENTS

`counts`: vector containing the row counts when the data list represents a table of profiles. Intended for use only with discrete response.

`smoothing`: label identifying the method of smoothing to be used. If missing, no smoothing is carried out. Possible smoothing functions are `ksmooth()`, identified by `"ksmooth"`, and `loess()`, which is interfaced with .mean() through the function `rm.loess()`. See the S–PLUS documentation for details.

`control`: function which allows users to set options for the smoothers. Available functions are `loess.ctrl()` and `ksmooth.ctrl()`. Arguments of the functions can be any of the input options parameters of the corresponding smoothers. Components that are not set will be replaced by their defaults within `rm.mean()`.

`batch`: list allowing a batch use (avoiding graphical interaction) of `rm.mean()`. Elements in list contain declarations of the levels of the covariates to be used to construct the mean profiles. Declarations are of the type `expression(covariate1 == "a" & covariate2 > x)`.

`subset`: expression stating which subset of the data rows should be used in the plot and fit. This is a logical statement of the type `covariate1 == "a" & covariate2 > x` which identifies the observations to be selected. The default is "all observations".

`given.values`: a numeric vector, character vector, or two- column matrix that specifies the given values when there is one given numerical predictor, or a list of two such objects when there are two. If missing, reasonable things happen.

`se`: if TRUE, add the confidence bands to the output.

`coverage`: confidence level of the limits as a fraction. By default the 95% pointwise confidence bands are computed.

timelab: time axis labels. If missing, the name of the second variable in `formula` is used.

ylab: response axis labels. If missing, the name of the first variable in `formula` is used.

timelim: approximate minimum and maximum values to be put on time axis.

ylim: approximate minimum and maximum values to be put on the response axis.

panel: a flag that determines the method of plotting on the graphical window. The default is TRUE.

Graphical parameters may also be supplied as arguments to this function (see `par`).

SIDE EFFECTS

a plot of the grand mean profile and its 95% confidence band are produced on the current graphics device. Dialog boxes showing scrolling lists of all the levels of the covariates (when used) are also presented. Levels in the scrolling lists can be interactively 'selected' and 'deselected'.

DETAILS

LAYOUT. There are two main layouts for `rm.mean()`, depending on the value of the flag `panel`. If `panel` is TRUE, then a matrix of dependence panels is created with as many panels as the number of levels for the first conditioning variable. On each dependence panel, mean profiles are plotted for the subjects showing the active selection for the covariate(s). The general mean profile is retained on each dependence panel. If `panel` is FALSE, then a single plotting window is created, showing the general mean profile and the mean profiles activated through interaction.

DIALOG BOX. A dialog box is present whenever:

- `panel` is FALSE and covariates appear in `formula`;
- `panel` is TRUE and two covariates appear in `formula`.

The upper part of the box contains a menu showing the levels of the covariates used in the call to the function. Labels of the covariates levels as declared in the data list are used to identify the levels; intervals are used for numerical covariates, when not otherwise stated (see `given.values` and `co.intervals()`). Two different interactive options can be chosen when `panel` is FALSE: `Incremental scan` or `Single scan`. A radiobox contains the selection for the interactive mode; the active mode is shown by the pressed button. The selection can be toggled at any time during interaction simply by clicking on the desired option and the two modes may be switched any number of times.

GRAPHICAL INTERACTION

Clicking on a level displays the mean profile and possibly the confidence band of the subset of cases (if any) which present the selected level. If this subset is empty, a new interactive selection is expected from the user. If two covariates are used, cases that satisfy the active selections on both the covariates are used to compute the highlighted mean profile.

When `panel` is FALSE and a new selection is activated in "incremental" mode, the mean profile of all the cases satisfying the new selection is superimposed to the existing plot; in the "single" mode, only the last selection is plotted. No graphical interaction is possible if covariates are not used.

VALUE

a list representing the mean profiles produced by the interactions. Elements of the list are matrices. The first row in each matrix presents the grandmean profile values; any additional row presents a sub-profile identified by the interaction. Row labels are used to help identify profiles. If quitting in `single` mode, the last active profile is included in the output matrices; if quitting in `incremental` mode, the last highlighted profiles (or those highlighted from the last single-incremental switch) are included. The matrices have as many rows as the number profiles on the plot window (including the grandmean) and a number of columns corresponding to the most frequent replicated observation among the selected profiles. The following components are present in the list:

DEFAULT VALUES

`call`: call to the function.

`time`: times of observations of the profiles values.

`fit`: fitted values of the mean profiles.

OPTIONAL VALUES. If `se = T` the following matrices are also present:

`upper`: upper values of the confidence band for the mean profiles.

`lower`: lower values of the confidence band for the mean profiles.

SEE ALSO

`rm.plot(), rm.fit(), rm.loess(), ksmooth()`

EXAMPLES

```
rm.mean(y time|x, data=citrate)
```

```
rm.mean(y time|x, citrate, smoothing = "loess",
control=loess.ctrl(span=1/5), se =T)
```

```
rm.mean(y time|x, citrate, smoothing = "loess",
batch = list(expression(x == "yes"),expression(x == "no")))
```

rm.fit()	Fit repeated measurements models	rm.fit()

```
rm.fit(formula = formula(data), data , family="gaussian",
cov.struct="I", random.effects , time, link="log", subset, start =
NULL, trace = T, contrasts = NULL, control =
nlminb.control(eval.max=50,x.tol=1e-6,rel.tol=1e-6), ...)
```

REQUIRED ARGUMENTS

`formula`: a formula expression as for regression models of the form `response ~ predictors`. See the documentation of formula for details.

`data`: a data list containing the variables occurring in the formula. List elements are identified by names. NA values are allowed. If `data` is missing, an error is produced.

OPTIONAL ARGUMENTS

`family`: a string identifying the parametric family. Supported families are "gaussian" (default), "binary" and "poisson".

`cov.struct`: a string identifying the covariance structure to use in the Gaussian case. Four possible values can occur: "I", independence, "AR(1)" first order autoregressive structure, "I+random", compound symmetry, "AR(1)+random" first order autoregressive structure+compound symmetry. The default value is "I". The last two options can be used jointly with the input parameter "random.effects".

`random.effects`: a formula identifying the structure of the random effects for the individuals. This is a formula of the type `response ~ random effects`. Used jointly with "I+random" and "AR(1)+random" allows to model random effects in the linear model for each observation.

`time`: a string that matches the name of the time variable in data. By default, the program expects a variable named "time" to be present in the data list, otherwise the name of the variable playing the role of time must be declared by assigning `time` here.

`counts`: a string that matches the name of the count vector in data. If the vector is named `"counts"`, then it is automatically used by the function, otherwise the name of the vector playing the role of counts must be declared by assigning `counts` here.

`link`: a string identifying the link for the Poisson case. Supported links are `"log"` (default) and `"identity"`.

`subset`: expression stating which subset of the data rows should be used in the fit. This is a logical statement of the type `covariate1 == "a" & covariate2 > x` which identifies the observations to be selected. The default is "all observations".

`contrasts`: a list of contrasts to be used for some or all of the factors appearing as variables in the model formula. The names of the list should be the names of the corresponding variables, and the elements should either be contrast-type matrices (matrices with as many rows as levels of the factor and with columns linearly independent of each other and of a column of one's), or else they should be functions that compute such contrast matrices.

`start`: a vector of initial values for the nuisance parameters of the likelihood. See the user manual for details on the use of start in the Gaussian case. In the binary and Poisson case, one value is required for `"start"`.

`control`: a list of algorithmic constants for the optimizer nlminb; see the S–PLUS documentation of `nlminb.control` for details on the possible control options. By default, `rm.fit()` sets the maximum number of function evaluations (`eval.max`) equal to 50, the parameter function convergence tolerance (`x.tol`) and the relative convergence tolerance (`rel.tol`) equal to 1e-6 and uses the `nlminb` standard default values for the remaining options; any setting of `control` modifies the default values of the options included in the list and causes the options that are not set in the list to be replaced by their standard default values within `nlminb`.

`trace`: logical flag: if TRUE, details of the nonlinear optimization are printed. By default the flag is set to TRUE.

VALUE

an object of class "rm" is returned. See `rm.object` for details.

DETAILS

FORMULA. The formula argument is passed around unevaluated; that is, the variables mentioned in the formula will be defined when the model matrices are computed, not when `rm.fit()` is initially called. The names of the variables used in the formula should generally match those of the variables in the data list.

DATA. Data are contained in a list. Elements of the data list can be matrices or vectors. At the very least, `data` must contain a matrix representing the response variable and a vector or matrix representing the observation times. Other elements of the list may contain covariates, possibly time dependent. Rows of matrices are associated to individuals, while column are associated to the observation times.

Each element of the data argument must be identifiable by a name. Names are assigned to the elements of a list by a standard assignment of the type:

```
data <- list(response = matrix(...), time = c(...))
```

Elements in the data list can be contained in any order, the name of the elements being the selecting argument for evaluation. `rm.fit()` expects a variable named `time` to be present in `data`; the rest of this documentation assumes this is the case. If the time component has been given a different name, the `time` optional argument should be used. Other functions like `rm.mean()` and `rm.plot()` do not need the `time` optional argument, the time component of the list being normally identified through the formula.

The simplest situation occurs when all subjects are observed at the same time points. In this case `time` can be a vector. The response variable is then a matrix with as many columns as the elements of `time`. This arrangement of the `data` list is still possible even when some of the response matrix are in fact missing. A typical situation of this kind occurs when the design of the experiment expected all subjects to be observed at the same time points, but in practice some of the subjects were not observed in some of the scheduled occasions. NA values can then be inserted in the response matrix. `rm.fit()` assumes they are missing completely at random.

A more complicated situation arises when the observation times differ for all or a large number of the individuals. In this case `time` must be a matrix with as many rows as the number of subjects. The number of columns in the response and time matrices is equal to the highest number of observed replications among individuals. Rows representing individuals with fewer replications are completed with NA values: significant values can occupy any column in a row, but the same ordering of significant and NA values must be maintained in every matrix of the list. It is recommended to set `time` as a matrix only for highly unbalanced problems; in the other cases, it is convenient to have a `time` vector, possibly inserting some NA's in the response variable matrix.

If discrete data are used, the list can represent a table of profiles. In this case, the response matrix represents the patterns of response and a vector should be included in the list, representing the counts, i.e. the replicated observations, for each profile. Typically, such vector will contain as many elements as the response matrix has rows. `rm.fit()` and `rm.mean()` expect such vector to be named "counts". If this is the case, the vector is automatically detected and used by the function and no action is required from the user. If the counts vector has been given a different name, the `counts` optional argument should be used.

Covariates which are constant over time must be represented by vectors with as many elements as the rows of the response matrix. Time dependent covariates which are constant across subjects must be represented by vectors with as many elements as the number of columns of the response matrix. Covariates varying simultaneously with time ad subject are represented by matrices. In the event that the number of subjects coincides with the number of replicated observations on each individual, i.e. the response matrix is a squared matrix, it is recommended to code time dependent covariates which are constant across subjects in the form of a matrix with all rows equal.

Factors need to be declared (see the documentation of `factor`, `as.factor`, `is.factor` for details). Since S-PLUS does not allow factors to be contained in matrices, categorical variables varying with the time must be contained in vectors. The length of such vectors is equal to the product of the number of rows and columns of the response variable matrix. Observations inside the vector should be grouped by individuals. For example, if we are observing two individuals each of which shows four replications, vectors of the categorical variables varying with the time should have eight elements of which the first fours represent the replicated observations of the first individual and the second fours those of the second individual. When these vectors are created by evaluating `factor` (`as.factor`) on an existing matrix `m` satisfying the requirements previously described, the right call to obtain the vectors in the desired form should be:

```
a <- factor(t(m))
```

SUBSET. The subset argument, like the terms in formula, is evaluated in the context of the data list. The specific action of the argument is as follows: before any evaluation starts, the appropriate subset of cases identified by the condition is extracted. Remember that the correct syntax for a multiple logical statement is of the type:

```
variable 1 == "a" & variable2 > 3 | variable3 == "c"
```

OPTIMIZATION. Unless `cov.struct="I"`, the model is fitted using `nlminb` (Non-linear Minimization subject to Box Constraints). See the documentation of the function for details.

SEE ALSO

`rm.object, rm.mean(), rm.plot(), rm.loess(), ksmooth()`

EXAMPLES

```
rm.fit(y~x, data = citrate)

rm.fit(y~factor(time), citrate, cov.struct = "AR(1)+random")

rm.fit(y~age*sex,PR.list,cov.struct="I+random",
random.effects=y~age)
```

rm.object	Repeated measurements model object	rm.object

DESCRIPTION

The object "rm" represents the fit of a repeated measurement model.

GENERATION

This class of objects is returned from the `rm.fit` function to represent a fitted linear model.

METHODS

The "rm" class of objects has methods for the following generic functions:
print, summary, residuals, fitted, anova.

INHERITANCE

The class "rm" does not inherit from any other S-PLUS class object.

STRUCTURE

The following components must be included in a legitimate rm object.

`call`: call to the function.

`coefficients`: matrix of coefficients of the maximum likelihood estimation. The names of the coefficients are the names of the single-degree-of-freedom effects (the columns of the model matrix).

`s.e`: vector of standard errors of the estimated parameters.

`covariance`: estimated covariance matrix. In the binary and Poisson case it is computed via numerical second derivatives of the log likelihood.

`log.likelihood`: the maximum of the log-likelihood function.

`message`: a statement of the reason for terminating the optimization task.

`data`: the original data list, restricted to the individuals used in the fit if necessary.

`n.cases`: number of profiles used in the fit.

`residuals`: the residuals from the fit.

`fitted.values` the fitted values from the fit.

`response`: a character value identifying the name of the response variable in the data list.

`family`: the parametric family.

`sigma2`: the 'within subject' estimated variance. Available in the Gaussian case only.

`rho`: the autoregressive parameter. Available in the Gaussian case only.

`delta`: the estimated variance(s) for the random effect(s), if present. When only an individual random component is present this is the ratio of the 'between subjects' over the 'within subjects' variance component. Available only in the Gaussian case.

`anova.aux`: list containing quantities relevant to the analysis of deviance table in the Gaussian case (see also `anova.rm()`). Available in the Gaussian case only.

DETAILS

All the components in a "rm" object can be extracted by the extractor operator `$`. The residuals and the fitted values could also be extracted by the generic functions `residuals` and `fitted`. These functions create a list containing, for the individuals used in the fit, all the variables included in the original data list except the response variable, which in turn is substituted either by the residuals or by the fitted values.

SEE ALSO

`rm.fit()`, `summary()`, `residuals.rm`, `fitted.rm`, `print.rm`, `anova.rm`

rm.loess	Smooth loess curves	rm.loess

```
rm.loess(x, y, span = 2/3, degree = 1, family = c("symmetric",
"gaussian"), evaluation = 50, coverage = 0.95)
```

ARGUMENTS

`x`: abscissas of points to be plotted in the plot window.

`y`: ordinates of points to be plotted in the plot window.

OPTIONAL ARGUMENTS

`span`: smoothing parameter. The default value is 2/3.

`degree`: overall degree of locally-fitted polynomial. 1 is locally-linear fitting and 2 is locally-quadratic fitting.

`family`: The values are "gaussian" or "symmetric". In the first case, local-fitting methods are used. In the second case, the default, local fitting is used together with a robustness feature that guards against distortion by outliers.

`evaluation`: number of values at which the loess curve is evaluated.

`coverage`: confidence level of the limits as a fraction. By default the 0.95 pointwise confidence interval is computed.

VALUE

This is a support routine for performing loess smoothing inside `rm.mean()`. See the documentation of `loess` for details on the smoothing procedure. It returns a list of values at which the `loess` curve is evaluated, and the corresponding evaluations.

SEE ALSO

`rm.mean()`

EXAMPLES

```
rm.mean(y time|x, citrate, smoothing = "loess",
control=loess.ctrl(span=1/5), se =T)
```

print.rm	Use print() on a rm object	**print.rm**

USAGE

```
print.rm(object, ...)
```

ARGUMENTS

`object`: any object produced by `rm.fit()`.

VALUE

This is a method for the function `print` for objects inheriting from class "rm". See `print` or `print.default` for the general behavior of this function.

SEE ALSO

`rm.fit(), rm.object, summary.rm()`

fitted.rm	Use fitted() on a rm object	**fitted.rm**

USAGE

```
fitted.rm(object)
```

ARGUMENTS

`object`: any object produced by `rm.fit()`.

VALUE

This is a method for the function `fitted` for objects inheriting from class "rm". The fitted values of the fit of a repeated measurements model to a data list are extracted from the input `object` and printed. The function returns a list containing, for the individuals included in the repeated measurements fit, all the variables appearing in the original data list except the response component, substituted by the fitted component. The `fitted` component can be extracted from the output list thorough the `$fitted` operator.

SEE ALSO

`rm.fit(), rm.object, residuals.rm`

residuals.rm	Use residuals() on a rm object	**residuals.rm**

USAGE

```
residuals.rm(object)
```

ARGUMENTS

`object`: any object produced by `rm.fit()`.

VALUE

This is a method for the function `residuals` for objects inheriting from class "rm". The residuals values of the fit of a repeated measurements model to a data list are extracted from the input `object` and printed. The function also returns a list containing, for the individuals included in the repeated measurements fit, all the variables appearing in the original data list except the response component, substituted by the residuals component. The `residuals` component can be extracted from the output list thorough the `$residuals` operator.

SEE ALSO

`rm.fit(), rm.object, fitted.rm`

anova.rm	Use anova() on a rm object	**anova.rm**

USAGE

```
print.rm(object, ...)
```

ARGUMENTS

`object`: any object produced by `rm.fit()`.

VALUE

This is a method for the function `anova()` for objects inheriting from class rm. See `anova()` for the general behaviour of this function and rm.tools manual for the interpretation of the arguments.

SEE ALSO

`rm.fit(), rm.object`

summary.rm	Use summary() on a rm object	summary.rm

USAGE

```
summary.rm(object, ...)
```

ARGUMENTS

`object`: any object produced by `rm.fit()`.

VALUE

a list is returned with the following components.

`call`: as contained in `object`.

`n.cases`: number of profiles used in the fit.

`coefficients`: A matrix with four columns, the first of which contains the label identifying the parameters in the following components of the output. In the Gaussian case it contains the coefficients, their standard errors and the corresponding t statistic. In the other cases, it contains along with the coefficients, their standard errors and their t-ratio (the ratio between the coefficients and their corresponding standard errors), also the estimate of the likelihood parameter, its standard error and its t-ratio.

`coef.cor`: matrix of correlations between the parameters estimates.

`sigma2`: the 'within subject' estimated variance. Available only in the Gaussian case.

`rho`: the autoregressive parameter. Available only in the Gaussian case.

`delta`: the estimated variance(s) for the random effect(s), if present. When only an individual random component is present this is the ratio of the 'between subjects' over the 'within subjects' variance component. Available only in the Gaussian case.

`log.likelihood`: the maximum of the log-likelihood function.

`message`: a statement of the reason for terminating the optimization task.

DETAILS

This function is a method for the generic function `summary()` for class `rm`. It can be invoked by calling `summary()` for an object of the appropriate class, or directly by calling `summary.rm()`.

SEE ALSO

`rm.fit()`, `summary()`, `rm.object`