Parameter Estimation in Nonlinear Mixed Effect Models Using saemix, an R Implementation of the SAEM Algorithm
Emmanuelle Comets, Audrey Lavenu, Marc Lavielle

To cite this version:

HAL Id: inserm-01502767
http://www.hal.inserm.fr/inserm-01502767
Submitted on 12 Apr 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Public Domain
Parameter Estimation in Nonlinear Mixed Effect Models Using saemix, an R Implementation of the SAEM Algorithm

Emmanuelle Comets
INSERM CIC 1414
U. Rennes-I
INSERM, IAME, UMR 1137
U. Paris Diderot

Audrey Lavenu
INSERM CIC 1414
U. Rennes-I

Marc Lavielle
INRIA Saclay, Popix
U. Paris Sud

Abstract

The saemix package for R provides maximum likelihood estimates of parameters in nonlinear mixed effect models, using a modern and efficient estimation algorithm, the stochastic approximation expectation-maximisation (SAEM) algorithm. In the present paper we describe the main features of the package, and apply it to several examples to illustrate its use. Making use of S4 classes and methods to provide user-friendly interaction, this package provides a new estimation tool to the R community.

Keywords: nonlinear mixed effect models, Stochastic Approximation EM algorithm, pharmacokinetics, pharmacodynamics, theophylline, orange tree, S4 classes.

1. Introduction

Over the recent years, a wealth of complex data has been accrued in many fields of biology, medicine or agriculture. These data often present a hierarchical structure, with correlations introduced by repeated measurements in the same individual. These correlations can be handled by modelling the evolution of a process with time and assuming subject-specific parameters to account for interindividual differences. The models used to describe the dynamics of biological processes are often nonlinear with respect to the parameters involved, and the statistical tools of choice in this context are nonlinear mixed effect models (Ette and Williams (2007)). Estimating the parameters of the models can be done through maximum likelihood or Bayesian approaches; in the present paper we will focus on the first method. In maximum likelihood approaches, the estimation process consists in defining a statistical criterion, generally minimising the likelihood or an approximation of the likelihood, and selecting an algorithm to obtain the minimum of this criterion.

Longitudinal data arise in many fields, such as agronomy, spatial analysis, imagery, clinical trials, and have been particularly prominent in the field of pharmacokinetics (PK) and pharmacodynamics (PD). As such, they play an important role in the process of developing new drugs, and PK/PD analysis is an integral part of the registration file submitted to health authority for the approval of new drugs (Lee, Garnett, Gobburu, Bhattaram, Brar, Earp,
Jadhav, Krudys, Lesko, Li, Liu, Madabushi, Marathe, Mehrotra, Tornoe, Wang, and Zhu (2011). It is also increasingly used to tailor drug treatment and guide dose adaptation in special populations such as renally impaired patients or children. Because of the importance of these models in PK/PD applications, the first estimation methods have been developed in pharmacometrics. In the context of maximum likelihood, a dedicated software, called NONMEM, was developed in the 70’s, which handles specific characteristics of pharmacologic data such as dosage regimen and other variables measured during treatment (Boeckmann, Sheiner, and Beal (1994)). The first algorithms implemented in this software relied on model linearisation to obtain an approximation of the likelihood, which cannot be computed easily in nonlinear mixed effect models. This approximation is then maximised through iterative Newton-Raphson minimisation, a general purpose algorithm involving the gradient of the function to be optimised. Different approximations to the likelihood have been proposed, including the first-order conditional method where the linearisation takes place around the current individual estimates at each iteration (Lindstrom and Bates (1990)). These estimation methods have also been implemented in mainstream statistical software such as SAS (proc NLMIXED, SAS Institute) and R (R Development Core Team (2006)), where the nlme (Pinheiro and Bates (1995)) library is now part of the base installation.

Linearisation-based methods however have both statistical and practical shortcomings. First and foremost, they do not in fact converge to the maximum likelihood estimates. While bias is generally minor for fixed effects, variance components may be significantly biased, especially with large interindividual variability. This has been shown to increase the type I error of likelihood tests (Comets and Mentré (2001); Bertrand, Comets, and Mentré (2008)), with the potential of building wrong models. Second, these methods suffer from severe bias when applied to non-continuous data, as shown by Molenberghs and Verbeke (2005), while stochastic algorithms have been shown to provide unbiased estimates (Savic, Mentré, and Lavielle (2011)). In practice, linearisation-based algorithms also exhibit convergence issues and can be tricky to use with complex models (Plan, Maloney, Mentré, Karlsson, and Bertrand (2012)). Over the past decade, new and powerful estimation algorithms have therefore been proposed to estimate the parameters of nonlinear mixed effect models (Lavielle (2014)).

The alternative to model linearisation is to compute the likelihood through numerical or statistical approximations, to preserve the statistical properties of maximum likelihood estimators. An example is the Laplace approximation, which is equivalent to a one-point adaptive Gaussian quadrature, and has been implemented in NONMEM. In R, the lme4 library uses this approximation (Bates, Maechler, and Bolker (2013)) with a penalised least square approach for the estimation algorithm. A powerful alternative to gradient-based minimisation algorithms is the EM algorithm, also an iterative algorithm, which has been developed in the context of missing data (Dempster, Laird, and Rubin (1977)). The Stochastic Approximation Expectation Maximization (SAEM) algorithm, combining a stochastic approximation to the likelihood with an EM algorithm, has proven very efficient for nonlinear mixed effect models, quickly converging to the maximum likelihood estimators (Delyon, Lavielle, and Moulines (1999)) and performing better than linearisation-based algorithms (Girard and Mentré (2005)). It has been implemented in the Monolix software (Lavielle (2014)), which has enjoyed increasingly widespread use over the last few years, more recently in the Statistics toolbox of MATLAB (nlmefitsa.m), and is also available in NONMEM version 7 (Boeckmann et al. (1994)). In the present paper, we provide an implementation of the SAEM algorithm in the R software (R Development Core Team (2006)) through the saemix package.
The package uses the S4 class system of R to provide a user-friendly input and output system, with methods like `summary` or `plot` for fitted objects. The package provides summaries of the results, individual parameter estimates, standard errors (obtained using a linearised computation of the Fisher information matrix) Wald tests for fixed effects, and a number of diagnostic plots, including VPC plots and npde (Brendel, Comets, Laffont, Laveille, and Mentré (2006)). The log-likelihood can be computed by three methods: a linearisation of the model, an importance sampling procedure, or a Gaussian quadrature. The diagnostic graphs can be tailored to the user’s individual preferences by setting a number of options, and are easily exported to a file. In the present paper we first present the statistical models, then we describe the features of the package by applying it to several examples. We conclude by simulation studies assessing the performance of `saemix` and its operating characteristics.

2. Nonlinear mixed effect models

2.1. Statistical models

A typical longitudinal dataset consists of observations collected in N individuals (for instance, subjects in a clinical trial). We assume that \( n_i \) observations \( y_{ij} = \{y_{i1}, ..., y_{in_i}\} \) have been collected in subject \( i \), at design points \( x_{ij} = \{x_{i1}, ..., x_{in_i}\} \) (in a clinical trial, \( x \) will typically be time or doses). The statistical model for observation \( y_{ij} \) is the following:

\[
y_{ij} = f(\psi_i, x_{ij}) + g(\psi_i, \sigma, x_{ij}) \epsilon_{ij}
\]  

In Equation 1, the function \( f \) represents the structural model, describing the evolution of the process being modelled, while the function \( g \) characterises the residual error model, more specifically its variance, and may depend on additional parameters \( \sigma \). We assume that the variability between subjects is captured by the vector of individual parameters \( \psi_i \), and that the errors \( \epsilon_{ij} \) are random samples from the standard normal distribution, and are independent of the \( \psi_i \).

When \( f \) is nonlinear with respect to the parameters \( \psi \), Equation (1) describes the general form of nonlinear mixed effect models. In `saemix`, we make some additional assumptions. We assume that the individual parameters \( \psi_i \) can be modelled parametrically as a function of fixed effects \( \mu \), individual random effects \( \eta_i \), and subject-specific covariates \( c_i \) and that the transformed parameters can be expressed using a linear relationship, where \( C_i \) is a matrix obtained by rearranging the covariates \( c_i \):

\[
\phi_i = h(\psi_i) = C_i \mu + \eta_i
\]  

so that the model parameters can be expressed as:

\[
\psi_i = h^{-1}(\mu, c_i, \eta_i) = h^{-1}(\phi_i)
\]
The **saemix** package

exponential model assumes that \( y_{ij} > 0 \) for all \( i, j \) and \( y_{ij} = f(\psi_i, x_{ij})e^{\theta(\psi_i, x_{ij})\epsilon_{ij}} \), where \( g = a \) (corresponding to an homoscedastic variance after log-transformation). We denote by \( \sigma \) the parameters of the residual error model (\( \sigma = \{a, b\} \)).

The unknown set of parameters in model defined by (1) are \( \theta = (\mu, vec(\Omega), \sigma) \), where \( vec(\Omega) \) is the vector of the parameters of the variance-covariance matrix \( \Omega \).

### 2.2. Parameter estimation

In the context of maximum likelihood, we are interested in the estimates of \( \theta \) which maximise the likelihood of the observations \( \ell(y; \theta) \). Assuming the different subjects are independent, \( \ell \) can be written as a product of the individual likelihoods, which are obtained as an integral over the distribution of the individual parameters \( D \):

\[
\ell(\theta ; y) = \prod_{i=1}^{N} \ell(\theta ; y_i) = \prod_i p(y_i|\theta) = \prod_i \int_D p(y_i|\eta_i, \theta)p(\eta_i|\theta)d\eta_i
\]  

This likelihood has no analytical expression when \( f \) is nonlinear. To get around this issue, two main approaches can be used. The first approach involves a linearisation of the model (Lindstrom and Bates (1990)) or of the likelihood (Wolfinger (1993)) to obtain a tractable expression, while the second approach uses numerical (Pinheiro and Bates (1995)) or stochastic (Wei and Tanner (1990)) approximations to compute the likelihood. Once an expression has been computed, the resulting likelihood is then maximised either through standard minimisation algorithms like the quasi-Newton algorithms, or through EM algorithms (Dempster et al. (1977)), where the unknown individual parameters are treated as missing data. In nonlinear mixed effect models, the E-step at the iteration \( k \) of the EM algorithm consists in computing the conditional expectation of the complete log-likelihood \( Q_k(\theta) = E(\log p(y, \psi; \theta)|y, \theta_{k-1}) \) and the M-step consists in computing the value \( \theta_k \) that maximises \( Q_k(\theta) \). Following Dempster et al. (1977) and Wu (1983), the EM sequence \( (\theta_k) \) converges to a stationary point of the observed likelihood (i.e a point where the derivative of \( \ell \) is 0) under general regularity conditions.

In cases where the regression function \( f \) does not linearly depend on the random effects, the E-step cannot be performed in a closed-form. The stochastic approximation version of the standard EM algorithm, proposed by Delyon et al. (1999) consists in replacing the usual E-step of EM by a stochastic procedure. At iteration \( k \) of SAEM, the algorithm proceeds as follows:

- **Simulation-step** : draw \( \psi^{(k)} \) from the conditional distribution \( p(\cdot|y; \theta_k) \).

- **Stochastic approximation** : update \( Q_k(\theta) \) according to

\[
Q_k(\theta) = Q_{k-1}(\theta) + \gamma_k(\log p(y, \psi^{(k)}; \theta) - Q_{k-1}(\theta))
\]  

- **Maximization-step** : update \( \theta_k \) according to

\[
\theta_{k+1} = \text{Arg max}_\theta Q_k(\theta).
\]

In the simulation step, the \( \psi^{(k)} \) are simulated at each iteration via a MCMC procedure from the current conditional distribution of the individual parameters. The same procedure can
also be used after the algorithm has converged to obtain the conditional distribution of the
individual parameters, from which we can compute the conditional modes, the conditional
means and the conditional standard deviations. In the stochastic approximation step, the
sequence of $\gamma_k$ controls the convergence of the SAEM algorithm. It should be a decreasing
sequence converging to 0 at a rate slower than 1 over the iteration number (Delyon et al.
(1999)). In practice, $\gamma_k$ is set equal to 1 during the first $K_1$ to let the algorithm explore
the parameter space without memory and to converge quickly to a neighbourhood of the
maximum likelihood estimator, and the stochastic approximation is performed during the
final $K_2$ iterations where $\gamma_k = 1/(k - K_1 + 1)$, ensuring the almost sure convergence of the
estimator. Also, the step size in the first few initial iterations (6 by default) is set to 0, as we
are not interested in computing $Q$ during the run-in sequence.

The SAEM algorithm has been implemented in the MATLAB software under the name Monolix
(Kuhn and Lavielle (2005)). It has been shown theoretically to converge to a maximum
of the likelihood of the observations under very general conditions (Delyon et al. (1999)). To
avoid local maxima, a simulated annealing step is implemented in a first series of iteration
to ensure proper exploration of the parameter space, which confers a certain robustness with
respect to the choice of initial parameter values to the SAEM algorithm. In practical ap-
plications, the SAEM algorithm implemented in Monolix has been shown to be effective and
fast (Girard and Mentré (2005); Plan et al. (2012)).

2.3. Estimation of the log-likelihood

The SAEM algorithm uses a stochastic approximation of the log-likelihood during the iteration
phase, and thus does not compute directly the log-likelihood. The likelihood in Equation (4)
can be computed at the end of the optimisation process by one of several approaches.

A first possibility is to approximate (4) by the likelihood of the Gaussian model deduced from
the nonlinear mixed effects model after linearisation of the function $f$ around the predictions of
the individual parameters ($\psi_i$, $1 \leq i \leq N$), essentially using the same approximation proposed

A second possibility is to use a Monte-Carlo approach, to obtain an estimate without any
model approximation. The integral $\ell(y; \theta)$ can be approximated via an Importance Sampling
integration method (IS), using the conditional distribution $p(\psi|y; \theta)$ which is obtained using
the empirically estimated conditional mean $E(\psi_i|y_i; \hat{\theta})$ and conditional variance $V(\psi_i|y_i; \hat{\theta})$
of $\psi_i$ for each subject $i = 1, 2, \ldots, N$. The quality of the approximation depends on the
estimates of the conditional mean and variances of the individual distributions.

A third possibility (GQ) is to use numeric integration as opposed to stochastic integration.
Gauss-Hermite quadrature methods use a fixed set of $K_{GQ}$ ordinates (called nodes) and
weights $(x_k, w_k)_{k=1,\ldots,K_{GQ}}$ to approximate the likelihood function. As for importance sam-
pling, the quality of the approximation depends on the estimates of $E(\psi_i|y_i; \hat{\theta})$ and $V(\psi_i|y_i; \hat{\theta})$.

The linearised approximation to the log-likelihood is the same approximation used in the
linearisation-based estimation methods, but is computed at the maximum likelihood esti-
mates obtained by SAEM. It is fast to compute, but becomes increasingly inaccurate as the
nonlinearity of the model increases. The two other methods provide respectively stochas-
tic and numerical approximations to the true log-likelihood, which should be asymptotically
equivalent when increasing the number of nodes (with Gaussian quadrature) and the number
of samples (with importance sampling). In practice, both these approximations have been
shown to carry a significant computational burden for large number of random effects (Pinheiro and Bates (1995)). In saemix, we have implemented fixed Gaussian quadrature with 12 nodes by default, and the number of nodes can be set to any value between 1 and 25 through the \texttt{nnodes.gq} argument, while the number of samples for the importance sampling method may be tuned by the \texttt{nmc.is} argument. Adaptive Gaussian Quadrature (AGQ) could be used to improve the approximation, as Pinheiro and Bates suggested AGQ to be more computationally efficient than IS and GQ, but is not yet implemented in saemix.

2.4. Estimation of standard errors

To assess the parameter estimates which have been obtained through the SAEM algorithm, we can consider the standard errors associated with the estimates. Estimates of these standard errors can be obtained as the inverse of the Fisher information matrix $I(\hat{\theta}) = -\partial^2 \log \ell(y; \hat{\theta})$ (see for instance (Walter and Pronzato (2007))).

As with the likelihood defined in (4), the Fisher information matrix of the nonlinear mixed effects model has no closed-form solution. Approximations to the Fisher information matrix have been proposed in the optimal design context by Mentré and others (Mentré, Mallet, and Baccar (1997); Retout, Mentré, and Bruno (2002)). In saemix we compute the Fisher information matrix by linearisation of the function $f$ around the conditional expectation of the individual Gaussian parameters ($E(\psi_i|y_i; \hat{\theta}), 1 \leq i \leq N$). The resulting model is Gaussian, and its Fisher information matrix is a block matrix (no correlations between the estimated fixed effects and the estimated variances). We compute the gradient of $f$ numerically. Alternatively, the Louis principle could be used to compute the Fisher information matrix through a stochastic approximation that does not involve a linearisation of the model (Louis (1982)).

2.5. Model evaluation

Model evaluation is a vital part of model building. Detecting model deficiencies can help guide the next model to be tested. Basic model diagnostics compare model point predictions and observations, while more sophisticated diagnostics involve simulations under the model (Brendel et al. (2006); Comets, Brendel, and Mentré (2008, 2010)). Other elements to assess model performance include predictive abilities on external datasets.

Computing predictions requires estimates of subject-specific parameters. In nonlinear mixed effect models, two sets of predictions are generally considered. Population predictions use the estimates of the population parameters and the individual design, typically dose regimen and regressors in PK/PD, while individual predictions are obtained using the individual parameters. The individual parameters ($\psi_i$) are obtained by estimating the individual normally distributed parameters ($\phi_i$) and deriving the estimates of ($\psi_i$) using the transformation $\psi_i = h(\phi_i)$. These estimates are obtained through $p(\phi_i|y_i; \hat{\theta})$, the conditional distribution of $\phi_i$ for $1 \leq i \leq N$, where $\hat{\theta}$ denotes the estimated value of $\theta$ computed with the SAEM algorithm. The MCMC procedure used in the SAEM algorithm is used to estimate these conditional distributions, and obtain for each subject $i$ the conditional mode (or Maximum A Posteriori) $m(\phi_i|y_i; \hat{\theta}) = \text{Arg max}_{\phi_i} p(\phi_i|y_i; \hat{\theta})$, the conditional mean $E(\phi_i|y_i; \hat{\theta})$, and the conditional standard deviation $sd(\phi_i|y_i; \hat{\theta})$. The number of iterations of the MCMC algorithm used to estimate the conditional mean and standard deviation is adaptively chosen to maintain the sequence of empirical means and SD within a $\rho_{mcmc}$-confidence interval during $L_{mcmc}$ iterations.
3. Overview of the package

3.1. Package structure

The `saemix` package implements the SAEM algorithm in R using S4 object-oriented programming. The main function in the package is the `saemix()` function, which estimates the population parameters of a nonlinear mixed effect model. This function requires two mandatory arguments, the (structural and statistical) model, a saemixModel object, and the data, a saemixData object created by the `saemixData()` function. An optional list of settings can be passed on as a list. The syntax of the `saemix()` function is simply:

```
saemix(model, data, control)
```

where the `control` argument is optional.

The model object is created by a call to the `saemixModel()` function, with the following main arguments:

```
saemixModel(model, psi0, description, error.model, transform.par,
            fixed.estim, covariate.model, covariance.model, omega.init, error.init,
            ...)
```

The first two arguments are mandatory, while all the others have default values.

- `model` name of an R function used to compute the structural model. The function should return a vector of predicted values given a matrix of individual parameters, a vector of indices specifying which records belong to a given individual, and a matrix of dependent variables (see example below for syntax).

- `psi0` a matrix with a number of columns equal to the number of parameters in the model, and one (when no covariates are available) or two (when covariates enter the model) rows giving the initial estimates for the fixed effects. The column names of the matrix should be the names of the parameters in the model, and will be used in the plots and the summaries. When only the estimates of the mean parameters are given, `psi0` may be a named vector.

- `description` a character string, giving a brief description of the model or the analysis.

- `error.model` type of residual error model (valid types are constant, proportional, combined and exponential).

- `transform.par` the distribution for each parameter (0 = normal, 1 = log-normal, 2 = probit, 3 = logit).

- `fixed.estim` whether parameters should be estimated (1) or fixed to their initial estimate (0).

- `covariate.model` a matrix giving the covariate model, with 0 for parameters to be estimated and 1 for parameters set to 0. Defaults to diagonal (only variance terms are estimated, and covariances are assumed to be zero).
covariance.model a square matrix of size equal to the number of parameters in the model, giving the variance-covariance matrix of the model.

omega.init a square matrix of size equal to the number of parameters in the model, giving the initial estimate for the variance-covariance matrix of the model.

error.init a vector of size 2 giving the initial value of a and b in the error model.

The data object is created by a call to the saemixData() function:

\[
\text{saemixData}(\text{name.data, header, sep, na, name.group, name.predictors, name.response, name.X, name.covariates, units, ...})
\]

The first argument is the only mandatory argument, and gives the name of the dataset. All others have default values, provided the columns of the dataset have names which can be identified by the program, although the user is encouraged to provide the names of the group, predictors and response columns for safety:

name.data name of the dataset (can be a character string giving the name of a file on disk or the name of a dataset in the R session).

header whether the dataset/file contains a header.

sep the field separator character.

na a character vector of the strings which are to be interpreted as NA values.

name.group name (or number) of the column containing the subject id.

name.predictors name (or number) of the column(s) containing the predictors (the algorithm requires at least one predictor x).

name.response name (or number) of the column containing the response variable y modelled as a function of predictor(s) x.

name.covariates name (or number) of the column(s) containing the covariates, if present (otherwise missing).

name.X (used in plots) name of the column containing the regression variable to be used on the X axis in the plots (defaults to the first predictor).

units list with up to three elements, x, y and optionally covariates, containing the units for the X and Y variables respectively, as well as the units for the different covariates.

Examples of how to use these functions are provided in Section 4.

The third argument in the call to saemix(), control, is optional; when given, it is a list containing options like the directory in which to save graphs and results, or the seed used to initialise the random number generator. The list of options which can be set is described in detail in the help files provided with the package.

The saemix() function returns an S4 object of class saemixObject. This object contains the following slots:
data the data object.

model the model object.

results the results object.

options a list of options containing variables controlling the algorithm.

prefs a list of graphical preferences, applied to plots.

Many methods have been programmed for saemixObject objects, including summary, print and plot, as well as functions to extract likelihoods, fitted values and residuals. Other functions are specific to saemix, and are used to compute the likelihood (llis.saemix, llgq.saemix), estimate and sample from the conditional distribution of individual parameters (conddist.saemix), compute standard errors of estimation using the linearised Fisher information matrix (fim.saemix), and simulate observations and parameters (simul.saemix). All the functions are documented in the online help.

3.2. Settings

Once a fit has been performed, the resulting object contains two named lists, which contain settings for running the algorithms and graphical preferences which are used mostly for plots. A detailed description can be found in the user documentation.

Algorithmic settings: these settings are passed on to the saemix() function through a list. All the elements in this list are optional and will be set to default values when running the algorithm if they are not specified by the user prior to the fit as elements of the control list. They include: (i) settings for the SAEM algorithm (the sequence of step sizes $\gamma$, the number of burning iterations $K_b$); (ii) settings for the MCMC algorithm (the number of Markov Chains $L$, the numbers $m_1$, $m_2$, $m_3$ and $m_4$ of iterations in the Hasting-Metropolis algorithm, the probability of acceptance $\rho$ for kernel $q^{(3)}$ and $q^{(4)}$); (iii) settings for the algorithm estimating the conditional distribution of the ($\phi_i$) (the width of the confidence interval $\rho_{mcmc}$ and the number of iterations $L_{mcmc}$); (iv) settings for the Simulated Annealing algorithm (the coefficient $\tau_1$ and $\tau_2$ defining the decrease of the temperature and the number of iterations $K_{sa}$); (v) settings for the algorithms to compute the likelihood (the Monte Carlo number $M$ for the importance sampling, number of quadrature points as well as the width of each integral for the Gaussian Quadrature algorithm); (vi) which modelling steps to perform after the estimation of the population parameters (estimation of individual parameters, estimation of the standard errors, estimation of different approximations to the likelihood).

User preferences: a list of default graphical parameters is stored in the prefs elements of the saemixObject object returned after a fit. These parameters control colours, line and symbol types and sizes, axes labels, titles, etc... and can be used to supersede default values in the plot functions. Any of these settings can be changed directly in the list, and will then affect all future plots, but can also be set on the fly when calling the plot function, then affecting only the plot being created.
4. Examples

The objective of this section is to show how to use the saemix package through several examples. In the first example, we will show the main features of a typical longitudinal dataset used in all modelling software, the Orange tree dataset. This very simple example will allow us to explain the structure of the data object and to create a simple logistic model that we can fit with saemix(). We then show the main output from the package. In the second example, we will consider a well-known pharmacokinetic example, giving an example of a covariate model, and we will use this example to showcase the diagnostic plots provided in the package. In the third example, we will study the statistical performances of the SAEM algorithm in terms of parameter estimation through a simulation study, by applying saemix to an Emax model for dose-response data, and we show the influence of tuning the parameters in the algorithm.

In the following we will assume that the saemix package has been loaded:

\[
R> \text{library("saemix")}
\]

4.1. Orange trees

The data: the orange tree data was originally presented by Draper and Smith (Draper and Smith (1981)) and was used by Pinheiro and Bates (1995) to demonstrate the nlme package. It is now available in the datasets package distributed with R, and it is also distributed with other statistical programs, most notably SAS and WinBugs. The dataset contains measurements of the circumference at chest height of 5 orange trees, measured at seven time points during a period of 4 years. The data is available in R under the name Orange.

The data object is created through the function saemixData. We need to specify the name of the dataframe, as well as the columns containing the grouping factor (indicating the subject), the predictor(s) and the response. Here, the grouping factor is Tree, and the number of the tree is given in the first column, while the second contains the age of the tree (the predictor) and the third its circumference (the response).

\[
R> \text{data("Orange")}
\]
\[
R> \text{head(Orange)}
\]

<table>
<thead>
<tr>
<th>Tree</th>
<th>age</th>
<th>circumference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>118</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>484</td>
<td>58</td>
</tr>
<tr>
<td>3</td>
<td>664</td>
<td>87</td>
</tr>
<tr>
<td>4</td>
<td>1004</td>
<td>115</td>
</tr>
<tr>
<td>5</td>
<td>1231</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>1372</td>
<td>142</td>
</tr>
</tbody>
</table>

R> orange <- saemixData(name.data = Orange, name. group = "Tree", + name.predictors = "age", name.response = "circumference", + units = list(x = "Days", y = "mm"))
Using the object called Orange in this R session as the data. The following SaemixData object was successfully created:

Object of class SaemixData
  longitudinal data for use with the SAEM algorithm
Dataset Orange
  Structured data: circumference ~ age | Tree
  Predictor: age (Days)

The units element is optional, but if given, will be used to label the plots. Note that it is also possible to give the number of the columns instead of the names, as in:

R> orange <- saemixData(name.data = Orange, name.group = 1,
+  name.predictors = 2, name.response = 3)

and that there is also some automatic name recognition built in the function (see the second example in Section 4.2).

A plot of the data can be obtained by a call to the plot function, yielding the graph shown in Figure 1.

R> plot(orange)

Figure 1: Orange tree data: tree circumference (in mm) as a function of times (in days since 1968/12/31).
Modelling: different mathematical functions can be used to describe growth curves. Pinheiro and Bates used the following three-parameters logistic model to predict $f(x)$, the circumference of the tree at age $x$:

$$f(x) = \frac{\lambda_1}{1 + e^{-(x - \lambda_2) / \lambda_3}}$$  \hspace{1cm} (6)

They assumed an additive error model with constant variance ($\epsilon_{ij} \sim N(0, \sigma^2)$). Since the dataset contained only 5 subjects, they assumed that only the intercept $\lambda_1$ varied across trees, and they set a normal distribution for $\lambda_1$ so that $\lambda_{1i} \sim N(\mu_1, \omega_1^2)$. The individual vector of parameters is then $\lambda_i = \{\lambda_{1i}, \lambda_2, \lambda_3\}$

To set up this model in saemix, we first define a general model function. This function must have 3 arguments named psi (assumed to be a matrix with a number of columns equal to the number of parameters in the model, here 3), id (assumed to be a vector of indices matching observation number with subject index) and xidep (assumed to be a matrix with as many columns as predictors, here 1). The three arguments passed to the function will be generated automatically from the model and data object within the saemix code. The function must return a vector of predictions, the length of which is equal to the number of rows in the predictor xidep. The following code snippet shows how to define the function $f$ for the orange tree model as a saemix model function:

```r
R> logistic.model <- function(psi, id, xidep) {
+ age <- xidep[, 1]
+ lambda1 <- psi[id, 1]
+ lambda2 <- psi[id, 2]
+ lambda3 <- psi[id, 3]
+ resp <- lambda1 / (1 + exp( -(age - lambda2) / lambda3))
+ return(resp)
+ }
```

The user only needs to change the computation of the response, and adjust the number of predictors and parameters. More examples of model functions can be found in the documentation online, as well as in the other examples (Sections 4.2 and 4.3).

We then build the statistical model through the function saemixModel:

```r
R> orange.model <- saemixModel(model = logistic.model,
+ description = "Logistic growth", psi0 = matrix(c(200, 800, 400),
+ ncol = 3, byrow = TRUE, dimnames = list(NULL, c("lambda1", "lambda2",
+ "lambda3"))), covariance.model = matrix(c(1, 0, 0, 0, 0, 0, 0, 0, 0),
+ ncol = 3, byrow = 3))
```

The following SaemixModel object was successfully created:

Nonlinear mixed-effects model

Model function: Logistic growth function(psi, id, xidep) {
age <- xidep[, 1]  
lambda1 <- psi[id, 1]  
lambda2 <- psi[id, 2]  
lambda3 <- psi[id, 3]  
resp <- lambda1 / (1 + exp(- (age - lambda2) / lambda3))  
return(resp)

}  

Nb of parameters: 3  
parameter names: lambda1 lambda2 lambda3  
distribution:  
Parameter Distribution Estimated  
[1,] lambda1 normal Estimated  
[2,] lambda2 normal Estimated  
[3,] lambda3 normal Estimated  

Variance-covariance matrix:  
lambda1 lambda2 lambda3  
lambda1 1 0 0  
lambda2 0 0 0  
lambda3 0 0 0  

Error model: constant , initial values: a = 1  
No covariate in the model.  
Initial values  
lambda1 lambda2 lambda3  
Pop.CondInit 200 800 400  

Note that in this model, we only estimate a variability for λ1, the other parameters are the same for all subjects. Since we don’t specify an initial value for the variance-covariance matrix, the default initialisation is used: a diagonal matrix, the diagonal elements of which are set to 1 for parameters with a log-normal, logit or probit distribution, and to the square of the initial value (set in argument psi0) for parameters with a normal distribution. Since the algorithm is stochastic, non-zero values must be set even for parameters without individual variability for the algorithm to simulate samples in the run-in phase.

Parameter estimation: to run saemix(), we pass the model and data arguments to the function. A list of options may be specified: in the code below, we set the seed for the random number generator, and we specify that we do not want graphs and results to be saved as they would be by default.

R> opt <- list(seed = 94352514, save = FALSE, save.graphs = FALSE)  
R> orange.fit <- saemix(orange.model, orange, opt)

The parameters are then estimated. The output includes a summary of the model and data, followed by the numerical results, which include parameter estimates, their standard errors and several statistical criteria. By default, the likelihood is computed both by linearisation and by importance sampling, and the corresponding Akaike (AIC) and Schwarz (BIC) information criteria are computed. In this very simple model, both are practically identical. For each parameter estimated in the model, estimates of the standard error are reported, as an absolute
The saemix package

value (SE) and relative to the estimate, as a coefficient of variation (% CV). In the present case, all the fixed parameters are well estimated, with coefficients of variations less than 10%, and the standard deviation of parameter $\lambda_1$ (measuring its variability) is estimated to be about 33, with an estimation error somewhat higher (65%) reflecting the limited information available with only 5 subjects in the dataset.

... 

--- Results ---

----------------------------------------------------
----------------- Fixed effects ------------------  
----------------------------------------------------

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>SE</th>
<th>CV(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,] lambda1</td>
<td>196.2</td>
<td>16</td>
<td>8.2</td>
</tr>
<tr>
<td>[2,] lambda2</td>
<td>748.3</td>
<td>38</td>
<td>5.1</td>
</tr>
<tr>
<td>[3,] lambda3</td>
<td>362.7</td>
<td>28</td>
<td>7.7</td>
</tr>
<tr>
<td>[4,] a</td>
<td>7.9</td>
<td>1</td>
<td>12.9</td>
</tr>
</tbody>
</table>

----------------------------------------------------
----------- Variance of random effects -----------  
----------------------------------------------------

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>SE</th>
<th>CV(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda1 omega2.lambda1</td>
<td>1043</td>
<td>673</td>
<td>65</td>
</tr>
</tbody>
</table>

----------------------------------------------------
------ Correlation matrix of random effects ------  
----------------------------------------------------

omega2.lambda1 omega2.lambda1 1

--------- Statistical criteria ---------

Likelihood computed by linearisation

-2LL = 263.4706  
AIC = 273.4706  
BIC = 271.5178

Likelihood computed by importance sampling

-2LL = 263.48  
AIC = 273.48  
BIC = 271.5272

During the fit, the progression of convergence is assessed through plots, shown in Figure 2. The vertical line in the plots correspond to $K_1$ iterations, and delineates two phases in the algorithm. During the first $K_1$ iterations, the algorithm explores freely the parameter space, while during the second phase ($K_2$ iterations), the stepsize $\gamma_k$ decreases slowly to ensure parameters convergence. The individual parameters of each subject are estimated during the second phase ($K_2$ iterations), and used for the diagnostics. These initial estimates can be
further refined after the fit through the `conddist.saemix()` function.

![Convergence plots for the Orange tree data.](image)

Figure 2: Convergence plots for the Orange tree data.

Finally, we can check that the model is able to describe the observed data by producing plots of the individual fits:

```r
R> plot(orange.fit, plot.type = "individual", smooth = TRUE)
```

this code adds the `smooth = TRUE` argument to use more time-points on the X-axis for the plot, and computes the prediction at each time-point using the estimated individual parameters for each tree. The result is shown in Figure 3 and illustrates a very good fit of the model for all subjects.

Corresponding code for `nlme` and `nlmer`: the same model can be fit to the Orange data using either `nlmer` from `lme4` or `nlme` from `nlme`. The following code, adapted from the online help for `nlme`, loads the two libraries and defines the model function as a user-defined function. It is also possible to use the built-in `SSlogis` function, but the purpose of this paragraph is to show how models in `saemix` relate to other modelling packages to facilitate switching from one to the other. The same function can be used for both `nlmer` and `nlme`, but `nlmer` needs the gradients to be defined in the function through the `deriv` function:

```r
R> library("nlme")
R> library("lme4")
```
Figure 3: Plots showing the individual predictions (solid line) overlaid on the observations (black dots) for each tree in the dataset.

```r
R> modelform <- ~ lambda1 / (1 + exp( -(age - lambda2) / lambda3))
R> nlmer.orangemod <- deriv(modelform, namevec = c("lambda1", "lambda2", + "lambda3"), function.arg = c("age", "lambda1", "lambda2", "lambda3"))
R> startvec <- c(lambda1 = 200, lambda2 = 800, lambda3 = 400)
```

The following code performs the fit using `nlmer`:

```r
R> nlmer.orangefit <- nlmer(circumference ~ nlmer.orangemod(age, lambda1, + lambda2, lambda3) ~ lambda1 | Tree, Orange, + start = startvec, random = lambda1)
```

The following code performs the fit using `nlme`, using a slightly different syntax to specify the structure of the fixed and random effects:

```r
R> nlme.orangefit <- nlme(circumference ~ nlmer.orangemod(age, lambda1, + lambda2, lambda3), fixed = lambda1 + lambda2 + lambda3 ~ 1, + random = lambda1 ~ 1, data = Orange, start = startvec)
```

The output from the `nlmer` fit is given by:

```r
R> summary(nlmer.orangefit)
```
Nonlinear mixed model fit by maximum likelihood ['nlmerMod']
Formula: circumference ~ nlmer.orangemod(age, lambda1, lambda2, lambda3) ~
         lambda1 | Tree
Data: Orange

AIC      BIC   logLik deviance df.resid
273.1    280.9  -131.6   263.1     30

Scaled residuals:
        Min 1Q Median 3Q Max
-1.9170 -0.5421  0.1754 0.7116 1.6820

Random effects:
Groups  Name    Variance  Std.Dev.
       Tree  lambda1    1001.49     31.646
          Residual     61.51      7.843
Number of obs: 35, groups: Tree, 5

Fixed effects:
        Estimate  Std. Error   t value
lambda1   192.05      15.58     12.32
lambda2   727.90      34.44     21.14
lambda3   348.07      26.31     13.23

Correlation of Fixed Effects:
        lambda1 lambda2
lambda2  0.384
lambda3  0.362  0.762

Warning messages:
1: In vcov.merMod(object, use.hessian = use.hessian) :
   variance-covariance matrix computed from finite-difference Hessian is
not positive definite: falling back to var-cov estimated from RX
2: In vcov.merMod(object, correlation = correlation, sigm = sig) :
   variance-covariance matrix computed from finite-difference Hessian is
not positive definite: falling back to var-cov estimated from RX

The output from the nlme fit is given by:

R> summary(nlme.orangefit)
The *saemix* package

Formula: `lambda1 ~ 1 | Tree`

<table>
<thead>
<tr>
<th></th>
<th>lambda1 Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>StdDev</td>
<td>31.48254 7.846255</td>
</tr>
</tbody>
</table>

Fixed effects: `lambda1 + lambda2 + lambda3 ~ 1`

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Std.Error</th>
<th>DF</th>
<th>t-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda1</td>
<td>191.0455</td>
<td>16.15380</td>
<td>28</td>
<td>11.82666</td>
<td>0</td>
</tr>
<tr>
<td>lambda2</td>
<td>722.5357</td>
<td>35.14849</td>
<td>28</td>
<td>20.55666</td>
<td>0</td>
</tr>
<tr>
<td>lambda3</td>
<td>344.1529</td>
<td>27.14659</td>
<td>28</td>
<td>12.67757</td>
<td>0</td>
</tr>
</tbody>
</table>

Correlation:

<table>
<thead>
<tr>
<th></th>
<th>lambda1</th>
<th>lambda2</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda1</td>
<td>0.375</td>
<td></td>
</tr>
<tr>
<td>lambda2</td>
<td></td>
<td>0.755</td>
</tr>
<tr>
<td>lambda3</td>
<td>0.354</td>
<td>0.755</td>
</tr>
</tbody>
</table>

Standardized Within-Group Residuals:

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Q1</th>
<th>Med</th>
<th>Q3</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1.9147537</td>
<td>-0.5351318</td>
<td>0.1436489</td>
<td>0.7309596</td>
<td>1.6615020</td>
</tr>
</tbody>
</table>

Number of Observations: 35
Number of Groups: 5

In this example, the estimates of the parameters as well as the statistical criteria are very close across the three packages.
4.2. Theophylline pharmacokinetics

The data: this is again a well-known dataset, distributed in NONMEM, Monolix and R, in nonlinear mixed effect models, and is a classic example of pharmacokinetic data. The data was collected in a study by Upton of the kinetics of the anti-asthmatic drug theophylline and analysed by Boeckmann et al. (1994). Twelve subjects were given oral doses of the anti-asthmatic drug theophylline, then serum concentrations (in mg/L) were measured at 11 time points over the next 25 hours. In package saemix, we removed the data at time 0 to avoid some unexplained non-zero values in a supposedly single-dose study; in addition we transformed the doses to body doses in mg instead of doses by weight mg/kg as in the original dataset, and we added a dummy variable for gender with values 0 and 1 to illustrate fitting categorical covariate model. The resulting dataset is available under the name theo.saemix in the saemix package.

Figure 4: Concentration of theophylline versus time in 12 subjects.

The data is shown in Figure 4. In the following code, we use the dataset from the library to create the saemixData object:

R> data("theo.saemix")
R> theo.data <- saemixData(name.data = theo.saemix, header = TRUE,
+ sep = " ", na = NA, name.group = c("Id"),
+ name.predictors = c("Dose", "Time"), name.response = c("Concentration"),
+ name.covariates = c("Weight", "Sex"),
+ units = list(x = "hr", y = "mg/L", covariates = c("kg", ")),
The data includes 2 predictors, Dose and Time; we use the `name.X = "Time"` argument to specify which of the predictor will be used as the independent variable in the plots. We read the two covariates in the data object, and we specify units for the independent and dependent variables, as well as for the covariates. The last line in this code is used to plot the data, changing a few options such as plot type, colour and title by passing these arguments to the `plot` function.

**Modelling:** these data were analysed in Davidian and Giltinan (1995)) and Pinheiro and Bates (1995) using a two-compartment open pharmacokinetic model. Subject \(i\) receives an initial dose \(D_i\) at time 0. Serum concentrations \(y_{ij}\) measured at time \(t_{ij}\) are modelled by a first-order one compartment model, according to the following equation:

\[
y_{ij} = \frac{D_i k_{ai} k_{ei}}{CL_i (k_{ai} - k_{ei})} \left( e^{-k_{ai} t_{ij}} - e^{-k_{ei} t_{ij}} \right) + \epsilon_{ij}
\]  

where \(CL_i\) is the clearance of subject \(i\), \(k_{ai}\) is the absorption rate constant, \(k_{ei}\) is the elimination rate constant and is expressed as a function of \(CL_i\) and the volume of distribution \(V_i\) as \(k_{ei} = \frac{CL_i}{V_i}\). For subject \(i\), the vector of regression (or design) variables is \(x_{ij} = (D_i, t_{ij})\), and the vector of individual parameters is \(\theta_i = (k_{ai}, CL_i, V_i)\). \(k_{ai}\), \(CL_i\) and \(V_i\) are assumed to be independent log-normal random variables (corresponding to \(h(x) = \ln(x)\)), and we assumed a relationship between the clearance and the subject’s body weight \(BW_i\):

\[
\begin{align*}
\ln(k_{ai}) &= \mu_1 + \eta_1 \\
\ln(V_i) &= \mu_2 + \eta_2 \\
\ln(CL_i) &= \mu_3 + \beta_{WT,CL} BW_i + \eta_3
\end{align*}
\]  

We used a simple homoscedastic error model where \(V(\epsilon_{ij}) = a^2\)

The following code was used to write a model function, and to create a `saemixModel` object:

```r
R> model1cpt <- function(psi, id, xidep) {
+  dose <- xidep[, 1]
+  tim <- xidep[, 2]
+  ka <- psi[id, 1]
+  V <- psi[id, 2]
+  CL <- psi[id, 3]
+  k <- CL / V
+  ypred <- dose * ka / (V * (ka - k)) * (exp(-k * tim) -
+          exp(-ka * tim))
+  return(ypred)
}
R> theo.model <- saemixModel(model = model1cpt,  
+  description = "One-compartment model with first-order absorption",  
+  psi0 = matrix(c(1, 20, 0.5, 0.1, 0, -0.01), ncol = 3, byrow = TRUE,  
+  dimnames = list(NULL, c("ka", "V", "CL"))), transform.par = c(1, 1, 1), 
+  covariate.model = matrix(c(0, 0, 1, 0, 0, 0), ncol = 3, byrow = TRUE))
```
The covariate model is specified as a matrix, and is reported in the model object as:

```
Covariate model:
  ka  V  CL
  [1,] 0 0 1
  [2,] 0 0 0
```

This indicates that the first covariate in the dataset (Weight in our example) will be assumed to have a relationship with parameter $CL$. In the code, $psi0$ now includes a second line, which is used to specify initial values for the parameter-covariate relationships. The same initial values would be used for all covariates, but we could specify different starting values for each parameter-covariate relationship by adding more lines to the matrix $psi0$. Alternatively, we can also let the program use default values of 0 by letting $psi0$ be a one-line matrix as in the Orange trees example.

**Parameter estimation:** as previously, we run this model using the `saemix()` function:

```R
R> opt <- list(save = FALSE, save.graphs = FALSE)
R> theo.fit <- saemix(theo.model, theo.data, opt)
```

```
-----------------------------------
---- Results ----
-----------------------------------
----------------------------------------------------
----------------- Fixed effects ------------------
----------------------------------------------------

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>SE</th>
<th>CV(%)</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,] ka</td>
<td>1.5786</td>
<td>0.2947</td>
<td>18.7</td>
<td>-</td>
</tr>
<tr>
<td>[2,] V</td>
<td>31.6605</td>
<td>1.4322</td>
<td>4.5</td>
<td>-</td>
</tr>
<tr>
<td>[3,] CL</td>
<td>1.5521</td>
<td>0.9683</td>
<td>62.4</td>
<td>-</td>
</tr>
<tr>
<td>[4,] beta_Weight(CL)</td>
<td>0.0082</td>
<td>0.0089</td>
<td>108.3</td>
<td>0.18</td>
</tr>
<tr>
<td>[5,] a</td>
<td>0.7429</td>
<td>0.0569</td>
<td>7.7</td>
<td>-</td>
</tr>
</tbody>
</table>

----------------------------------------------------
----------- Variance of random effects -----------
----------------------------------------------------

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>SE</th>
<th>CV(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ka omega2.ka</td>
<td>0.368</td>
<td>0.1668</td>
<td>45</td>
</tr>
<tr>
<td>V omega2.V</td>
<td>0.017</td>
<td>0.0096</td>
<td>57</td>
</tr>
<tr>
<td>CL omega2.CL</td>
<td>0.065</td>
<td>0.0324</td>
<td>50</td>
</tr>
</tbody>
</table>

----------------------------------------------------
------ Correlation matrix of random effects -----
----------------------------------------------------

<table>
<thead>
<tr>
<th>omega2.ka</th>
<th>omega2.V</th>
<th>omega2.CL</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega2.ka</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>omega2.V</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>omega2.CL</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
The saemix package

------------------ Statistical criteria ------------------

Likelihood computed by linearisation
-2LL = 343.427
AIC = 359.427
BIC = 363.3063

Likelihood computed by importance sampling
-2LL = 344.8205
AIC = 360.8205
BIC = 364.6997

Again fixed effects are well estimated, except for the fixed effect of weight on clearance, which is estimated with a variability of 108%.

Covariate model: we can test whether the covariate effect is significant. In nonlinear mixed effect models, standard tests are the Wald test and the log-likelihood ratio test (LRT). The Wald test compares the value of a parameter estimate $\hat{\Psi}(k)$ to the standard error of estimation $SE(\hat{\Psi}(k))$ through a $\chi^2$ statistic with one degree of freedom:

$$W = \left( \frac{\hat{\Psi}(k)}{SE(\hat{\Psi}(k))} \right)^2 \sim \chi^2_1$$

(9)

This is the test performed in the output above for the fixed effects involved in a covariate model, and here we see that in this example, the fixed effect representing the influence of weight on CL is not significant ($p = 0.18$, NS according to the Wald test), in keeping with the large estimated SE.

The LRT compares two nested models through the difference in twice the log-likelihood ($LL$). Assuming the more parsimonious model $M_1$ has $p$ parameters and the larger model $M_2$ $p + q$ parameters, asymptotically the difference in $-2LL$ follows a $\chi^2$ statistic with $q$ degree of freedom:

$$Z = -(2LL_1 - 2LL_2)^2 \sim \chi^2_q$$

(10)

In saemix, the likelihood can be computed by three different approximations. Two of these are computed by default with the standard settings of the algorithm. To compute also the likelihood by Gaussian quadrature, we use the `llgq.saemix()` function:

```R
R> theo.fit <- llgq.saemix(theo.fit)
R> theo.fit

This adds the log-likelihood computed by Gaussian quadrature to the object:

...  

Likelihood computed by Gaussian quadrature
-2LL = 344.7868
AIC = 360.7868
BIC = 364.6997
To test the covariate model using the LRT, we first need to fit a model without the covariate, then we compute the LRT statistic. Here we use the log-likelihoods computed by linearisation, but in this example we get similar p-values with all approximations.

```r
R> theo.model.base <- saemixModel(model = model1cpt,
+   description = "One-compartment model with first-order absorption",
+   psi0 = matrix(c(1., 20, 0.5, 0.1, 0, -0.01), ncol = 3, byrow = TRUE,
+   dimnames = list(NULL, c("ka", "V", "CL"))), transform.par = c(1, 1, 1))
R> opt <- list(save = FALSE, save.graphs = FALSE)
R> theo.base <- saemix(theo.model.base, theo.data, opt)
R> ll1 <- theo.base["results"]"ll.lin" * (-2)
R> ll2 <- theo.fit["results"]"ll.lin" * (-2)
R> 1 - pchisq(ll1 - ll2, 1)
[1] 0.4365312
```

**Model selection:** Statistical tools for model selection include information criteria and hypothesis tests such as the Wald test and likelihood ratio test (LRT) for nested models. Non-nested models can be compared through Akaike criterion (AIC) and Schwarz’s information criterion, also called Bayes information criterion or BIC. These two statistics are reported in the output for each likelihood computed. Their value is given by:

\[
AIC = -2LL + 2p \\
BIC = -2LL + p \ln(n_{tot})
\]

(11)

where \(p\) is the number of parameters in the model and \(n_{tot}\) the total number of observations. AIC and BIC criteria are justified based on asymptotic criteria (Burnham and Anderson (2002)), that is, when the sample size increases and the model dimension stays fixed. In the context of mixed effects models, the effective sample size is not clearly defined and the formula for calculating the BIC differs from software to software. The penalty using \(n_{tot}\) is implemented in the R package nlme and SPSS procedure MIXED, while the number of subject, \(N\), is used in Monolix and SAS proc NL MIXED. An optimal BIC penalty based on two terms proportional to \(\log(N)\) and \(\log(n_{tot})\) and that is consistent with the random effect structure in a mixed effects model was recently proposed by Delattre, Poursat, and Lavielle (2014).

Other criteria have been proposed and studied for nonlinear mixed effect models (see review in Bertrand, Comets, Chenel, and Mentré (2012)); they can be computed from the estimate of the log-likelihood, which can be accessed through elements of the results object. For instance, the log-likelihood computed by importance sampling can be obtained as:

```r
R> logLik(theo.fit)
```

LL by is "]172.41 (df=8)"

We should also note that a conditional AIC (cAIC) has been proposed by Vaida and Blanchard (2005) for linear mixed effects models. Its theoretical properties have been investigated...
by Greven and Kneib (2010) and implemented for lme4 in an R package. In nonlinear mixed effect models, small sample corrections have been proposed for the Wald test by Bertrand et al. (2012), who also suggest using permutation tests to control type I error inflation in model selection with the LRT and the Wald test.

Model diagnostics: diagnostics are destined to evaluate model properties and help guide model building. A number of diagnostic graphs are produced automatically when running saemix(), and are output by a call to the plot() function applied to the object resulting from the fit. Specific plots can also be created using the plot.type argument to the function. When called individually, each graph can be tailored to user preferences through options as in the previous example (Figure 4). The next code snippet shows how to obtain specific graphs, given in Figure 5 to 8. The first graph, in Figure 5, displays the individual fits for the first 4 subjects in the theophylline example, including a smoothed prediction line, and changing the color of the line and the plotting symbol. A logarithmic scale is used for the Y-axis. Figure 6 plots the observations versus the predictions, adding the unity line to assess deviations. Figure 7 assesses the distribution of npde, a residual adapted to nonlinear mixed effect models (Brendel et al. 2006; Comets et al. 2010)), while Figure 8 gives the Visual Predictive Check.

R> par(mfrow = c(2, 2))
R> plot(theo.fit, plot.type = "individual.fit", new = FALSE, ilist = 1:4,
+ smooth = TRUE, ylog = T, pch = 1, col = "Blue", xlab = "Time in hr",
+ ylab = "Theophylline concentrations (mg/L)"
) 
R> plot(saemix.fit, plot.type = "observations.vs.predictions")
R> plot(saemix.fit, plot.type = "npde")
R> plot(saemix.fit, plot.type = "vpc")

Overall, these plots show good model adequacy, and individual profiles indicates that the model is able to reproduce observed data reasonably well. Some points to note may be the departure of the npde from the theoretical normal distribution seen in Figure 7, as well as a slight tendency in the plots of npde versus time. The VPC plot in Figure 8 shows that the model predicts the median evolution very well; the variability may be slightly overestimated but with only 12 subjects in the dataset this may be expected.
Figure 5: Individual plots for the first 4 subjects in the study, with different options.

Figure 6: Observations versus predictions for the theophylline data, with population predictions on the left and individual predictions on the right.
The *saemix* package

Figure 7: Individual plots for the first 4 subjects in the study, with different options.

Figure 8: VPC for the theophylline data.
Corresponding code for \texttt{nlme} and \texttt{nlmer}: again, we can cast the same model for use in \texttt{nlme} and \texttt{nlmer}. For \texttt{nlme}, the relationship between parameters and covariates is set in the \texttt{fixed} argument within the function call:

\begin{verbatim}
R> startvec.ke <- c(lKe = -2.99, lKa = 0.5, lCl = 0.75, lClSex = 0)
R> nform <- ~Dose * exp(lKe+lKa-lCl) * (exp(-exp(lKe)*Time) -
+ exp(-exp(lKa) * Time)) / (exp(lKa) - exp(lKe))
R> nlme.theomod.ke <- deriv(nform, namevec = c("lKe", "lKa", "lCl"),
+ function.arg = c("Dose", "Time", "lKe", "lKa", "lCl"))
R> nlme.fit.sex <- nlme(Concentration ~ nlme.theomod.ke(Dose, Time, lKe,
+ lKa, lCl), data = theo.nlmerdat,
+ fixed = list(lKe ~ 1, lKa ~ 1, lCl ~ Sex),
+ random = pdDiag(lKe + lKa + lCl ~ 1), start = startvec.ke)
\end{verbatim}

For \texttt{nlmer} on the other hand, the model function needs to be rewritten to explicitly include the covariate effects, as follows:

\begin{verbatim}
R> nform.sex <- ~ Dose * exp(lKe + lKa - lCl - lClSex * Sex) *
+ (exp( -exp(lKe) * Time) - exp( -exp(lKa) * Time)) /
+ (exp(lKa) - exp(lKe))
R> nlmer.theomod.ke.sex <- deriv(nform.sex,
+ namevec = c("lKe", "lKa", "lCl", "lClSex"),
+ function.arg = c("Dose", "Time", "lKe", "lKa", "lCl", "lClSex", "Sex"))
R> nlmer.fit.sex <- nlmer(Concentration ~ nlmer.theomod.ke.sex(Dose, Time,
+ lKe, lKa, lCl, lClSex, Sex) ~ (lKe | Id) + (lKa | Id) + (lCl | Id),
+ data = theo.saemix, start = list(nlpars = startvec.ke))
\end{verbatim}

In both cases, the model had to be reparameterised in terms of log-parameters ($lKe$, $lKa$ and $lCl$) as both packages only handle normal distributions for the random effects. In addition, both \texttt{nlme} and \texttt{nlmer} fail to converge when the model is parameterised with the volume of distribution ($V$) instead of the elimination rate constant $Ke$, which is equal to $CL/V$. Although we can reparameterise the model as we did here, it is worth noting that this changes the mixed effect model, because it modifies the structure of the interindividual variability. Another point to note is that for \texttt{nlmer}, a new function needs to be created for each parameter-covariate relationship that the user wishes to test.

4.3. Evaluating the performances of saemix

\textbf{Data and model:} In this section, we use simulated data to evaluate the performance of \texttt{saemix}. The simulated datasets were generated by Plan et al. (2012) in a paper comparing different software to estimate parameters in nonlinear mixed effect models, and were obtained directly from the authors of that paper. The link is provided in Acknowledgments.

The model used for this simulation is a sigmoid $E_{\text{max}}$ model, a standard model in dose-response studies where the effect of a drug in response to a dose $d$, $E(d)$, is a sigmoid function corresponding to the following equation:

$$E(d) = E_0 + E_{\text{max}} \frac{d^l}{d^l + ED_{50}}$$

(12)
The `saemix` package

This model involves 4 parameters, the initial effect \( E_0 \), the maximum effect \( E_{\text{max}} \), the concentration at which half the maximum effect is achieved \( EC_{50} \) and the sigmoidicity factor \( \gamma \) which controls the nonlinearity of the model through the curvature. Interindividual variability was modelled through a log-normal distribution for all parameters, except for \( \gamma \) which was assumed to be the same for all subjects (no IIV). A correlation was simulated between \( E_{\text{max}} \) and \( EC_{50} \). In Plan et al., 3 values of \( \gamma \) were tested, 1, corresponding to the \( E_{\text{max}} \) model, and 2 and 3, involving increasing amounts of nonlinearity (Plan et al. (2012)), along with two residual error models, additive or proportional error. In the present work, we focus on the scenarios with \( \gamma = 3 \) with constant variance \( \sigma^2 \) (scenarios R3A for the rich and S3A for the sparse design), but full results on the 8 scenarios simulated in Plan et al. (2012) are available on request. The parameters used in the simulation are given in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_0 ) (-)</td>
<td>5</td>
<td>( \omega^2_{E_0} )</td>
<td>0.09</td>
</tr>
<tr>
<td>( E_{\text{max}} ) (-)</td>
<td>30</td>
<td>( \omega^2_{E_{\text{max}}} )</td>
<td>0.49</td>
</tr>
<tr>
<td>( EC_{50} ) (mg)</td>
<td>500</td>
<td>( \omega^2_{EC_{50}} )</td>
<td>0.49</td>
</tr>
<tr>
<td>( \gamma ) (-)</td>
<td>3</td>
<td>( \text{cov}(E_0, E_{\text{max}}) )</td>
<td>0.245</td>
</tr>
<tr>
<td>( \sigma ) (-)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Parameters used in the simulation for the E\( _{\text{max}} \) model (see Plan et al. (2012)).

Simulation study: the design of the simulation study mimicked that of a clinical trial including 100 individuals and investigating four dose levels (0, 100, 300, and 1000 mg). Two sampling designs were evaluated, a rich design (\( R \)) in which all individuals were sampled at the four dose levels, and a sparse design (\( S \)) where each individual was randomly allocated to only two of the four dose levels.

We evaluated the influence of the starting values and of the tuning parameters on the performance of `saemix`. Initial parameter estimates were either set to the values used in the simulation (true) or to different values (false), where the fixed parameters were multiplied by 2 while the variability parameters were set to 0.1. We also used either the default settings of the algorithm, or tuned the settings by increasing the number of chains to 5 and increasing the number of iterations to 300 and 150 respectively in the burn-in and convergence phases of the algorithm. Combining starting values and tuning parameters, we therefore performed 4 successive parameter estimations for each dataset. In all settings, we used the same random seed for all runs.

Evaluation: the performance of both algorithms was evaluated by computing the relative bias and root mean square error (RMSE) over the 100 simulations, using the simulation parameters as true values. Denoting \( \Psi_0 \) the true value of parameter \( \Psi_k \), and \( \hat{\Psi}_k^l \) the value estimated in the \( l \)th simulated dataset, the relative bias for this parameter was given by:

\[
RB(\Psi^k) = \frac{\sum_{l=0}^{100} \hat{\Psi}_k^l - \Psi_0^k}{\Psi_0^k} \quad (13)
\]
and the relative RMSE by:

\[ RRMSE(\Psi^k) = \frac{100}{\Psi_0^2} \sum_{l=0}^{100} (\hat{\Psi}_l^k - \Psi_0^k)^2 \]  \hspace{1cm} (14)

**Results:** tables 2 and 3 show the relative bias and root mean square error obtained for both methods in the two scenarios, in the four settings.

<table>
<thead>
<tr>
<th>Settings</th>
<th>Default</th>
<th>Tuned</th>
<th>Default</th>
<th>Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial parameters</td>
<td>saemix</td>
<td>saemix</td>
<td>saemix</td>
<td>saemix</td>
</tr>
<tr>
<td>(E_0)</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>(E_{\text{max}})</td>
<td>0.43 (4.8)</td>
<td>-0.04 (4.74)</td>
<td>-0.14 (4.37)</td>
<td>-0.23 (4.28)</td>
</tr>
<tr>
<td>(E_{50})</td>
<td>1.34 (17.52)</td>
<td>4.54 (17.56)</td>
<td>1.47 (13.68)</td>
<td>2.95 (13.54)</td>
</tr>
<tr>
<td>(\gamma)</td>
<td>-0.56 (13.03)</td>
<td>1.70 (13.07)</td>
<td>-0.24 (10.53)</td>
<td>0.88 (10.84)</td>
</tr>
<tr>
<td>(\sigma)</td>
<td>3.35 (10.04)</td>
<td>0.97 (9.32)</td>
<td>1.79 (6.94)</td>
<td>0.54 (6.46)</td>
</tr>
<tr>
<td>(\omega_{E_0}^2)</td>
<td>3.74 (10.04)</td>
<td>0.40 (10.07)</td>
<td>-0.48 (6.4)</td>
<td>-0.32 (6.37)</td>
</tr>
<tr>
<td>(\omega_{E_{\text{max}}}^2)</td>
<td>-5.15 (37.39)</td>
<td>0.9 (28.38)</td>
<td>0.32 (27.93)</td>
<td>-5.15 (37.39)</td>
</tr>
<tr>
<td>(\omega_{E_{50}}^2)</td>
<td>-0.53 (21.45)</td>
<td>3.30 (27.86)</td>
<td>-1.54 (20.88)</td>
<td>0.23 (21.14)</td>
</tr>
<tr>
<td>(\omega_{E_{\text{max}}}^2)</td>
<td>-2.98 (36.71)</td>
<td>3.58 (37.47)</td>
<td>-1.86 (31.2)</td>
<td>1.27 (30.08)</td>
</tr>
<tr>
<td>(\gamma_{E_{\text{max}}} E_{50})</td>
<td>-1.47 (55.93)</td>
<td>10.69 (65.08)</td>
<td>-0.88 (48.28)</td>
<td>4.85 (48.72)</td>
</tr>
<tr>
<td>(\rho)</td>
<td>-4.9 (32.64)</td>
<td>0.71 (33.45)</td>
<td>-2.66 (27.6)</td>
<td>0.69 (26.85)</td>
</tr>
</tbody>
</table>

Table 2: Relative bias (RRMSE, %) in the scenario with rich sampling.

In the rich sampling scenario, all parameters are estimated without bias (less than 5% relative bias) irrespective of whether initial parameters were set to the true value or not, or whether the algorithm was tuned. The relative root mean square errors are less than 20% for the fixed effects, and between 20 and 50% for the variability parameters. Figure 9 shows a boxplot of the biases observed in the 100 datasets, when running the algorithm with the default settings and false starting values: for most of the datasets and all parameters except the covariance the bias is less than 20% in the majority of the datasets.

In the sparse sampling scenario (Table 3) on the other hand, some bias is apparent for several parameters: the fixed parameters \(E_{50}\) and \(E_{\text{max}}\) are poorly estimated when the starting parameters are moved away from the true values used in the simulation, even when the number of chains and iterations is increased. Their variabilities are also poorly estimated, with a large RMSE. This is consistent with the results found in Plan et al. (2012). In that paper, most algorithms had difficulties estimating \(E_{50}\) and \(E_{\text{max}}\) with the sparse design, which was quite challenging as it combined limited individual information and high model nonlinearity, and all methods showed biases over 20 to 30% on several parameters, worsening when perturbed initial values were used. This is illustrated in Figure 10 with the same settings as in Figure 9. This time most of the datasets show bias for \(E_{\text{max}}\) and \(E_{50}\), and the variability between the datasets in bias is also much larger, especially for variabilities.
Table 3: Relative bias (RRMSE, %) in the scenario with sparse sampling.

<table>
<thead>
<tr>
<th>Settings</th>
<th>Default</th>
<th>Tuned</th>
<th>Default</th>
<th>Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E_0$</td>
<td>2.5 (7.97)</td>
<td>0.96 (8.31)</td>
<td>0.31 (6.26)</td>
<td>-2.03 (6.75)</td>
</tr>
<tr>
<td>$E_{max}$</td>
<td>-4.28 (22.69)</td>
<td>38.18 (68.24)</td>
<td>-1.2 (21.75)</td>
<td>33.5 (46.78)</td>
</tr>
<tr>
<td>$ED_{50}$</td>
<td>-2.72 (19.96)</td>
<td>34.88 (59.25)</td>
<td>-1.86 (18.72)</td>
<td>29.52 (43.07)</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>12.17 (30.54)</td>
<td>-12.18 (23.26)</td>
<td>6.58 (19.89)</td>
<td>-16.93 (21.18)</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>3.8 (16.9)</td>
<td>6.05 (19.58)</td>
<td>-0.99 (11.32)</td>
<td>-0.16 (11.3)</td>
</tr>
<tr>
<td>$\omega^2_{E_0}$</td>
<td>-17.5 (79.45)</td>
<td>-19.6 (86.07)</td>
<td>3.27 (54.95)</td>
<td>5.58 (60.81)</td>
</tr>
<tr>
<td>$\omega^2_{E_{max}}$</td>
<td>-5.78 (37.85)</td>
<td>21.06 (67.68)</td>
<td>-0.24 (33.97)</td>
<td>29.38 (62.72)</td>
</tr>
<tr>
<td>$\omega^2_{ED_{50}}$</td>
<td>-18.51 (47.84)</td>
<td>29.3 (72.66)</td>
<td>-6.36 (42.43)</td>
<td>43.91 (71.99)</td>
</tr>
<tr>
<td>$cov(E_{max}, ED_{50})$</td>
<td>-31.22 (82.42)</td>
<td>51.6 (139.77)</td>
<td>-5.84 (71.81)</td>
<td>80.19 (131.59)</td>
</tr>
<tr>
<td>$\rho$</td>
<td>-35.2 (67.43)</td>
<td>2.91 (59.09)</td>
<td>-11.9 (50.82)</td>
<td>25.35 (44.63)</td>
</tr>
</tbody>
</table>
Figure 9: Bias over 100 simulations for the parameters estimated in the rich design. The solid line signals an absence of bias ($y = 0$) and the two dotted lines are plotted at +/- 20% bias.
Figure 10: Bias over 100 simulations for the parameters estimated in the sparse design. The solid line signals an absence of bias ($y = 0$) and the two dotted lines are plotted at +/- 20% bias.
5. Operational characteristics

**saemix** is one of several packages in R that can be used to fit nonlinear mixed effect models. In this section, we first provide a comparison of **saemix** to **nlme** and **lme4** in terms of bias and precision of parameter estimates, as well as convergence. In a second part, we investigated runtimes when model complexity and dataset size increases to evaluate the scalability of the approach.

5.1. Comparing **saemix** to **nlme/nlmer**

**Simulation study:** the datasets from section 4.3 have been fit using **nlme** by Plan et al. (2012), who report convergence for only 5% (rich design) and 16% (sparse design) of the datasets. We had the same issue when trying to fit the same data (results not shown). We attempted to use **nlmer** from **lme4** and had the same convergence problems; with **nlmer** we also had a technical issue, in that the derivative with respect to $\gamma$ is not defined for a dose of 0, which required some tinkering with the data to bypass. Because we could fit so few datasets, we could not compare meaningfully the performance of **saemix** to those of the other two modelling packages. In (Plan et al. (2012)), the rate of convergence decreased as $\gamma$ increased, suggesting it was related to model nonlinearity. In this section, we therefore decided to use the datasets simulated with $\gamma = 1$ and we fit them using an $E_{\text{max}}$ model corresponding to Equation 12 with $\gamma = 1$ (fixed, not estimated). The datasets for the sparse design were derived from the datasets with a rich design (scenario R1A) by using for each dataset the same sampling times as in scenario S3A analysed in the previous section.

Initial conditions for **nlmer** were set to the true values used in the simulation, given in Table 1, as for **saemix**, except for $\text{cov}(E_0, E_{\text{max}})$ and $\sigma$ which could not be initialised in the other packages. As we encountered convergence issues again, we changed the initial parameter estimates within 20% of the original values; this was repeated a maximum of 10 times per dataset.

**Results:** tables 4 and 5 shows the bias and RMSE for the parameters estimated by each of three packages for the rich and sparse scenarios respectively.

<table>
<thead>
<tr>
<th></th>
<th>saemix</th>
<th>nlme</th>
<th>nlmer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_0$</td>
<td>-2.51 (5.84)</td>
<td>0.29 (4.69)</td>
<td>2.12 (5.46)</td>
</tr>
<tr>
<td>$E_{\text{max}}$</td>
<td>1.09 (9.43)</td>
<td>-1.92 (7.98)</td>
<td>-2.56 (8.57)</td>
</tr>
<tr>
<td>$ED_{50}$</td>
<td>-2.54 (14.41)</td>
<td>-8.05 (13.48)</td>
<td>-6.34 (12.36)</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>1.12 (6.86)</td>
<td>1.94 (6.26)</td>
<td>-0.95 (6.11)</td>
</tr>
<tr>
<td>$\omega^2_{E_0}$</td>
<td>1.09 (42.38)</td>
<td>7.72 (31.12)</td>
<td>0.76 (32)</td>
</tr>
<tr>
<td>$\omega^2_{E_{\text{max}}}$</td>
<td>-1.84 (19.21)</td>
<td>-8.5 (18.22)</td>
<td>-1.66 (18.32)</td>
</tr>
<tr>
<td>$\omega^2_{ED_{50}}$</td>
<td>-2.13 (37.46)</td>
<td>-27.6 (34.83)</td>
<td>-16.6 (30.95)</td>
</tr>
<tr>
<td>Failed runs</td>
<td>0</td>
<td>0</td>
<td>56</td>
</tr>
</tbody>
</table>

Table 4: Relative bias and RRMSE for the three packages in the scenario with rich sampling.

A first result concerns convergence rates: while **saemix** provided estimates for all datasets, we could not obtain parameter estimates for more than half of the datasets with **nlmer** despite
retrying a number of times. With \texttt{nlme}, all datasets could be fit in the rich scenario but 7% failed with a sparse design. In terms of estimation errors, \texttt{saemix} provided unbiased estimates in the rich design, while both \texttt{nlme} and \texttt{nlmer} had difficulties to estimate $ED_{50}$ and its variability. In the sparse scenario, both \texttt{saemix} and \texttt{nlme} tended to slightly overestimate $\sigma$ and underestimate variabilities. \texttt{saemix} tended to underestimate the variability on $E_0$ and $ED_{50}$, while \texttt{nlme} strongly underestimated the variabilities on $E_{\text{rmmax}}$ and $ED_{50}$. The results of \texttt{nlmer} were not meaningful considering estimates could be obtained in only 16 datasets out of 100, but they also indicated similar issues.

<table>
<thead>
<tr>
<th></th>
<th>saemix</th>
<th>nlme</th>
<th>nlmer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_0$</td>
<td>-1.33 (9.06)</td>
<td>-5.05 (11.1)</td>
<td>0.76 (7.79)</td>
</tr>
<tr>
<td>$E_{\text{max}}$</td>
<td>1.07 (16.56)</td>
<td>-2.13 (19.51)</td>
<td>0.11 (6.91)</td>
</tr>
<tr>
<td>$ED_{50}$</td>
<td>0.42 (30.3)</td>
<td>-14.42 (40.09)</td>
<td>-3.11 (17.26)</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>6.71 (18.71)</td>
<td>9.05 (21.09)</td>
<td>-9.11 (23.17)</td>
</tr>
<tr>
<td>$\omega^2_{E_0}$</td>
<td>-16.9 (80.13)</td>
<td>1.55 (72)</td>
<td>-1.57 (60.01)</td>
</tr>
<tr>
<td>$\omega^2_{E_{\text{rmmax}}}$</td>
<td>-3.05 (42.46)</td>
<td>-19.34 (33.61)</td>
<td>-14.66 (32.75)</td>
</tr>
<tr>
<td>$\omega^2_{ED_{50}}$</td>
<td>-17.34 (107.92)</td>
<td>-48.7 (81.95)</td>
<td>-37 (59.48)</td>
</tr>
</tbody>
</table>

Table 5: Relative bias and RRMSE for the three packages in the scenario with sparse sampling.

These results show convergence issues with \texttt{nlmer} and to a lesser extent with \texttt{nlme}, in keeping with the findings reported in (Plan et al. (2012)), which had been obtained with a more nonlinear model. In our simplified settings, on a very simple $E_{\text{max}}$ model, \texttt{nlme} converged more often than with high sigmoidicity, but showed more estimation bias than \texttt{saemix}.

### 5.2. Computation times

We performed two series of simulations to evaluate the scalability of the algorithm. In the first, we investigated the effect of the size of the datasets by simulating datasets using the $E_{\text{max}}$ model from section 4.3, and varying the number of samples and the number of subjects. In the second, we assessed the effect of increasing model complexity on the runtimes. For this second simulation, we simulated three models with one, two and three exponential terms (respectively 2, 4 and 6 parameters). Each dataset was simulated assuming 100 subjects and 6 sample times, which had been optimised by the PFIM software (Retout, Comets, Samson, and Mentré (2007)) to ensure reasonable identifiability of the most complex model. The model with two exponentials is similar to the $E_{\text{max}}$ model in terms of model complexity. In each simulation, 100 datasets were generated, and the runtimes for the fit of the 100 datasets were collected through the \texttt{Rprof()} function in R, and summarised. Table 6 shows the total run times in seconds reported by \texttt{Rprof()}; the same information is displayed in Figure 11, with the graph on the left showing the evolution of run times with the number of subjects, stratified by the number of samples, and the graph on the right showing the influence of model complexity.

The values in Table 6 were obtained on a personal computer and are purely indicative, as they depend on CPU, memory and concurrent tasks. The relative values between the different settings however show clearly how the algorithm scales with increasing model complexity (roughly linearly with respect to the number of parameters to estimate), and with the size of
Emmanuelle Comets, Audrey Lavenu, Marc Lavielle

<table>
<thead>
<tr>
<th>$E_{\text{max}}$ model</th>
<th>Nsuj</th>
<th>nsamples</th>
<th>Run time, 100 simulations (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>3</td>
<td></td>
<td>340</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td></td>
<td>419</td>
</tr>
<tr>
<td>30</td>
<td>12</td>
<td></td>
<td>582</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
<td></td>
<td>654</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td></td>
<td>823</td>
</tr>
<tr>
<td>100</td>
<td>12</td>
<td></td>
<td>1191</td>
</tr>
<tr>
<td>200</td>
<td>3</td>
<td></td>
<td>1198</td>
</tr>
<tr>
<td>200</td>
<td>6</td>
<td></td>
<td>1580</td>
</tr>
<tr>
<td>200</td>
<td>12</td>
<td></td>
<td>2289</td>
</tr>
<tr>
<td>500</td>
<td>3</td>
<td></td>
<td>2878</td>
</tr>
<tr>
<td>500</td>
<td>6</td>
<td></td>
<td>3718</td>
</tr>
<tr>
<td>500</td>
<td>12</td>
<td></td>
<td>5325</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exponential Number of random effects</th>
<th>Run time, 100 simulations (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>348</td>
</tr>
<tr>
<td>4</td>
<td>662</td>
</tr>
<tr>
<td>6</td>
<td>1157</td>
</tr>
</tbody>
</table>

Table 6: Run times in seconds for different models, exploring the effect of increasing the number of samples and the number of subjects in the $E_{\text{max}}$ model (30 to 500 subjects, 3 to 12 samples), and of increasing the number of parameters in the model (exponential models with 2, 4 and 6 parameters).

Figure 11: Evolution of runtimes with the number of subjects, the number of samples, and model complexity.

the dataset. In Figure 11, there is a slight bend in the curve between a number of subjects equal to 30 and a number of subjects higher than 100. This is because when there are less than 50 subjects in the datasets, more chains are generated by the algorithm in order to improve model stability, increasing the number of ‘individuals’ considered in the algorithm.
Other than that, run times are seen to increase linearly with the number of subjects and more slowly with the number of samples. Run times also increase approximately linearly with model complexity. These results show the algorithm behaves as expected, as the number of random effects to be generated at each iteration of the algorithm depends on both model complexity and number of subjects, while the number of samples influences the time spent to compute model predictions and many intermediate computations. These profiling results will also be used to identify computational bottleneck in further developments of saemix.

6. Discussion

The use of modelling and simulation in clinical drug development is now well established. Regardless of whether a single outcome is considered at the end of the study, clinical trials routinely collect longitudinal data, with each subject providing several measurements throughout the study. Longitudinal data is a staple in particular of pharmacokinetic (PK) and pharmacodynamic (PD) studies, which are a required part of a new drug application file (Lee et al. (2011)). Nonlinear mixed effect models can help to characterise and to understand many complex nonlinear biological processes, such as biomarkers or surrogate endpoints, and are crucial in describing and quantifying the mechanisms of drug action and the different sources of variation, e.g., the interindividual variability (Ette and Williams (2007)). As a result, these models have been extensively studied over the past two decades, with many developments in methods for parameter estimation, experimental design, or model evaluation. Parameter estimation can be performed through maximum likelihood (ML), restricted maximum likelihood (REML), or Bayesian approaches.

The saemix package provides an implementation of one of these recent estimation algorithms, SAEM, in R, to obtain maximum likelihood estimates. It offers an interesting alternative to linearisation-based estimation methods as implemented for instance in nlme and lme4. The different packages all have their own specificities and limitations. The oldest, nlme, is included in the base packages distributed with R, and as lme4, can handle multiple levels of variability. They both implement maximum and restricted maximum likelihood estimation. lme4 also offers an approach based on Adaptive-Gaussian Quadrature but only when the model is linear. A limitation of nlmer is that there is currently no way to fit heteroscedastic error models, which are frequently used in pharmacokinetics. In practice however, nlme and nlmer often run into convergence issues, especially as model complexity increases. Both nlme and nlmer require the model to be specified in closed form; an extension nlmeODE has been developed to handle structural models defined by a differential equation system (Tornoe, Agersø, Jonsson, Madsen, and Nielsen (2004)), but it also suffers from convergence issues. saemix also has some limitations in the models that can be set up. Covariates must enter the parameter model linearly (after one of the available transformations), and there is no automated treatment of categorical covariates, although this can be circumvented by users creating sets of binary covariates. In addition, for the moment saemix can only handle one response. Finally, the current version of the package is better suited to models with closed-form solutions, although there is no limitations from an algorithmic point of view on the models that can be fit. Using a system of differential equations (ODE) can be implemented quite easily within R using deSolve for instance (code available on request), and this approach could be used to implement a model with Michaelis-Menten elimination, for which no closed form solution of the model exists. We however recommend against integrating ODE models
Emmanuelle Comets, Audrey Lavenu, Marc Lavielle

with \texttt{ode()}. Indeed, runtimes are exceedingly long even with the theophylline example, which is a very simple model first-order elimination model with only 12 subjects. They would in addition increase linearly with the number of subjects, quickly becoming intractable. It is also possible with \texttt{deSolve} to implement the structural model in \texttt{Fortran} to improve the computational time but this is beyond the scope of this paper.

The SAEM algorithm has been shown to have good statistical properties in various theoretical (Kuhn and Lavielle (2005)) and practical applications (Girard and Mentré (2005); Panhard and Samson (2009); Comets, Lavenu, and Lavielle (2011)). We show here that the \texttt{R} implementation \texttt{saemix} performs well on reference datasets used to compare different softwares by Plan et al. (2012). Most of the scenarios run with \texttt{nlme} showed various degrees of convergence issues in that paper, whether started from the true or from altered parameters. Estimates were obtained for only 5% of the datasets simulated under a rich design, and 16% of those simulated under a sparse design in (Plan et al. (2012)). Convergence issues were related to the nonlinearity between dose and response, with a $\gamma$ of 3. Of note, FOCE in NONMEM or SAS did not show the same convergence problems, suggesting the \texttt{R} implementation of the FOCE algorithm could probably be improved. We attempted here to fit the same datasets with \texttt{nlmer} (data not shown), but encountered even more convergence problems. Note also we could not fit the original dataset directly with \texttt{nlmer}, as the gradient of the model is undefined for a dose of 0. We had to change the dataset by assuming a dose of 0.01 instead of 0 to avoid this numerical problem. \texttt{nlme} for the same datasets also showed major convergence issues. In the present paper, we therefore compared \texttt{saemix} to \texttt{nlme} and \texttt{nlmer} on a less challenging setting using an E\textsubscript{max} model instead of the Hill model. However, \texttt{nlmer} still failed in over half of the datasets we attempted to fit, despite trying different initial parameter estimates. \texttt{nlme} fared better in terms of convergence, but still exhibited more bias than \texttt{saemix}. Both packages were run with the default settings, which might explain in part the poor results we found here.

The SAEM algorithm involves repeated evaluations of the model function, and runtimes increase with the number of subjects in the dataset. One possibility to improve the speed of \texttt{saemix} will be to implement a stopping criterion to move into the second phase of the algorithm when the estimates of the parameters stabilise, as has been done in Monolix (Lavielle (2014)). A sizeable portion of the time spent in the main algorithm is devoted to evaluations of the model predictions. Model predictions could be obtained from more efficient codes in \texttt{C}. Other computationally intensive functions include the computation of the log-likelihood by importance sampling, which again requires repeated model evaluations. These sections could again be externalised to \texttt{C} in future versions of \texttt{saemix}.

\texttt{saemix} has a number of settings that the user can change. Most of the time these settings can be left at the default values, with the exception of the following. With small sample sizes, the number of chains (argument \texttt{nb.chains}) can be increased as the results are then averaged over the different chains and yield more stable estimates from run to run. The number of iterations (argument \texttt{nbiter.saemix}) in both burn-in and convergence phases of the algorithm can also improve convergence, as shown in section 4.3. In practice, the convergence plots can be used to assess the stability of the parameter estimates at the end of the burn-in phase to detect whether to increase the number of iterations. The number of samples used to compute the log-likelihood by importance sampling (argument \texttt{nmc.is}) can be increased to compute a more precise estimate of the log-likelihood if the estimate does not appear stable yet in the convergence plots. Obviously, increasing the values of these settings will have an impact on
The `saemix` package

runtimes. Finally, the robustness of the estimated parameters can be assessed by changing the random seed (argument `seed`) and initial estimates of the parameters around the final estimates. Practical illustrations are proposed in (Lavielle (2014)).

To use `saemix`, the user needs to define two specific objects, one containing the hierarchical data structure and the other containing the elements of the mixed effect models. The structural model is defined through a R function that the user must create, which is one of the difference with `nlme` or `lme4` which both use R formulas. Although this requires a bit more work for the user to create this function, it is clear from the examples in 4.2 and 4.3 that this it not very difficult. Specifying the interindividual variability in saemix is very straightforward: the random effects are sampled from normal distributions and the user only needs to specify the transformation for each random effect, as well as the joint covariance structure. This allows great flexibility, compared to eg `nlme` where changing from a normal to a log-normal distribution requires to reparameterise the model and its derivatives in terms of log-parameters. The other advantage is a clearer separation between regressors and covariates, which are defined in the data object, and parameters, which belong to the model. This is a much more flexible structure to include for instance complex dose regimens or other inputs to the system. The structure of the `saemixModel` object is also more easy to interface with other outputs (C programs, specialised libraries) which may not comply with R standards. The future of `saemix` will indeed be tied in with larger developments that are ongoing in the pharmacometrics community. There is currently a European consortium called DDMoRe working on developing a general language for pharmacometrics, encompassing structural models from pharmacokinetics to system biology and statistical models ranging from frequentist to Bayesian approaches (Harnisch, Matthews, Chard, and Karlsson on behalf of the DDMoRe consortium Partners and Contributors (2013)). Two authors of this paper (Comets and Lavielle) are part of this endeavour and `saemix` will be extended to support this language.

In conclusion, we have developed the `saemix` algorithm in an R package to estimate the parameters of nonlinear mixed effect models. We do not intend to suggest that it is better than the other modelling packages available in R, merely that it can be used to complement the existing array of estimation methods. It is very easy to use, as we show in the current paper how to cast models in `nlme` and `nlmer` in terms of `saemix`, so that it can be used to compare the results for different estimation methods and evaluate the robustness of the analysis. However, the underlying algorithm, SAEM, has been shown to have superior statistical properties and good operational performances when compared to gradient-based algorithms. We hope to extend `saemix` to more complicated models by using the MLX-TRAN language and solver, which are being externalised to C/C++ so that the models can be called and model predictions obtained from different software. Other extensions will include more complex data, involving simultaneous modelling of several responses, left-censored data, as well as allow the user to model the interindividual variability through other parameter distributions and write his or her own residual error model. These more advanced features are already available in the Monolix software, which is implemented in MATLAB and has many specific features adapted to PK/PD applications as well as an extensive library of PK/PD models and a modelling language where ODE models can be easily coded. However, the objective in developing the `saemix` was not to provide a complete rewrite of Monolix, but to offer the SAEM estimation algorithm to the R community, to be used in conjunction with or as an alternative to other
packages for nonlinear mixed effect models. As such, it has many potential applications for the analysis of longitudinal data, which are encountered for instance in agronomy, chemistry, medical trials and of course pharmacokinetics and pharmacodynamics.

Acknowledgments

We thank Elodie Plan and her co-authors Alan Maloney, France Mentré, Mats Karlsson, Julie Bertrand, for kindly providing us the datasets used in Plan et al. (2012) to evaluate saemix in Section 4.3. The datasets can be downloaded from: https://www.dropbox.com/sh/haag3i20Ifu/yhb/C6kam13Ngs

References


The saemix package


Affiliation:
Emmanuelle Comets
CIC 1414, Laboratoire de pharmacologie expérimentale et clinique
Faculté de Médecine
2 av. Pr Léon Bernard
35 000 Rennes, France
Telephone : +33/0/2-23-23-47-65
Fax : +33/2-23-23-46-05
E-mail: emmanuelle.comets@inserm.fr
URL: http://www.saemix.biostat.fr/