Fitting Reducible SDE Models

Vignette for **resde**

Contents

1 Introduction **1**

2 The model **3**

3 Single-unit estimation **4**
3.1 Example 1, single unit with additive process noise 5
3.2 Example 2, single unit with multiplicative process noise . . 8

4 Hierarchical (two-level) models **10**
4.1 Example 3, fixed local parameters 11
4.2 Example 4, random local parameters 15

5 Additional features and advanced usage **17**
5.1 Derivatives 17
5.2 Under the hood 17
5.3 Fixing `eta` 19
5.4 On transformations 20
5.4.1 A unifying transformation 20
5.4.2 More general curves 23
5.4.3 General differential equations 25

References **25**

1 Introduction

Package **resde** computes maximum-likelihood (ML) parameter estimates for reducible stochastic differential equations (RSDEs, more on these in a

minute). Observations are at discrete points in time, not necessarily evenly spaced, and may include measurement error. Currently, **resde** handles only univariate time-independent RSDEs. In the simplest case, there is a single individual or sampling unit (*unit*, for short). More generally, there may be a set of units, with some parameters common to all units (*global*), and others specific to each unit (*local*). Estimation in these *two-level hierarchical models* can use either a fixed effects approach, or more fashionable mixed effects methods.

RSDEs are SDEs that can be reduced to linear by a change of variables. This may seem unduly restrictive, but we shall see that many important problems can be formulated this way. In particular, linearizing transformations exist for most of the univariate growth models encountered in the literature. In fact, any differential equation with an invertible closed-form solution can be linearized (section 5.4.2).

In García (2019), I showed an **R** function, `uvector()`, that generates a certain sum of squares that minimized produces ML estimates (*not* least-squares estimates). That worked, but the necessary incantations could become rather complex and less than intuitive. Package **resde** offers a more user-friendly front-end. There are two main functions: `sdemodel()` specifies the model form and initial conditions — for a unit in case of hierarchical models. That includes the variable transformation, any re-parameterizations, and the existence or not of process, measurement, and initial condition noise. Then, `sdefit()` uses `uvector()` internally to perform the estimation, given the model defined by `sdemodel()`, the data, and starting parameter guesses. For hierarchical models, one must also identify the parameters as globals or locals, and indicate if fixed or mixed effects are to be used.

Section 3 of García (2019) provides a quick introduction to SDEs. García (2013) might be useful for those not comfortable with dynamical systems.

The following section describes the model equations, and how they are specified in `sdemodel()`. Section 3 explains the use of `sdefit()` for single-unit models, and section 4 does the same for hierarchical ones. We go over all the SDE examples from García (2019), where a few additional details can be found. Look also in there for literature references. Finally, section 5 covers some non-essential odds and ends.

2 The model

We assume that there is some transformation of the variable of interest X,

$$Y = \varphi(X) \,, \tag{1}$$

possibly containing unknown parameters, such that Y for a unit follows a linear SDE

$$\mathrm{d}Y = (\beta_0 + \beta_1 Y)\,\mathrm{d}t + \mu_p \sigma_p \,\mathrm{d}W \,, \tag{2}$$

with a possibly random initial condition

$$Y(t_0) = \varphi(x_0) + \mu_0 \epsilon_0 \,, \quad \epsilon_0 \sim N(0, \sigma_0^2) \,. \tag{3}$$

That is, X obeys a *reducible* (to linear) SDE. There are n possibly noisy observations x_i such that

$$\begin{aligned} y_i = \varphi(x_i) = Y(t_i) + \mu_m \epsilon_i \,; \quad i = 1, \ldots, n \,; \\ t_0 < t_1 < \cdots < t_n \,; \quad \epsilon_i \sim N(0, \sigma_m^2) \,. \end{aligned} \tag{4}$$

The ϵ_i are mutually independent, and independent of the Wiener (aka Brownian motion) random process $W(t)$.

There may be a re-parameterization, where any of the "base" parameters β_0, β_1, t_0, y_0, μ_p, μ_0, μ_m can be replaced by functions of new parameters. For instance, an SDE

$$\mathrm{d}X^c = b(a^c - X^c)\,\mathrm{d}t + \sqrt{b}\sigma_p \,\mathrm{d}W \,,$$

with initial condition fixed at the origin, can be specified by the transformation $Y = \varphi(X, c) = X^c$ and the parameter substitutions $\beta_0 \leftarrow b a^c, \beta_1 \leftarrow -b, \mu_p \leftarrow \sqrt{b}$, and $t_0, x_0, \mu_0 \leftarrow 0$. If there are no measurement errors one would set $\mu_m \leftarrow 0$, otherwise $\mu_m \leftarrow 1$.

In **resde**, the model specification is done with function `sdemodel()`. The arguments and defaults are

```
sdemodel(phi=~x, phiprime=NULL, beta0=~beta0, beta1=~beta1, t0=0,
         x0=0, mu0=0, mup=1, mum=1)
```

The values are either constants, or formulas with an empty left-hand side. The optional `phiprime` can be an expression for the derivative of φ with

respect to x. If the derivative is not given, it is automatically generated with the **Deriv** package. The result from `sdemodel()` is used by the estimation function `sdefit()`.

Assume that **resde** is properly installed. For the example above,

```
library(resde)
exmpl <- sdemodel(phi=~x^c, beta0=~b*a^c, beta1=~-b, mup=~sqrt(b))

## Model:
##        y = phi(x, c) = x^c
##        y' = phiprime(x, c) = c * x^(c - 1)
##        dY = (b * a^c + -b * Y) dt + sqrt(b) * sigmap * dW
##        Y(0) = phi(0, c)
##        yi = Y(ti) + sigmam * ei
## Parameters:
##        a, b, c
```

The leading `##` are not displayed by **R**, but are used here to distinguish outputs from inputs. Always check the displayed result to see if that is what you meant. The display can be obtained again with `sdemodel_display(exmpl)`. Note that here the errors `ei` and `e0` have unit vsriances, so that $\epsilon_i = \sigma_m e_i$ and $\epsilon_0 = \sigma_0 e_0$. More examples below.

3 Single-unit estimation

The following are the `sdefit()` arguments relevant to fitting an RSDE model to a single individual or sampling unit: `sdefit(model, x, t, data=NULL, start=NULL, known=NULL)`. The output of `sdemodel()` is passed in the argument `model`. Arguments `x` and `t` are either data vectors, or names for the relevant columns in the data frame `data`. A named vector or list `start` gives starting parameter values for the optimization. The optional named vector or list `known` can contain parameters fixed at given values for a particular estimation run, as an alternative to running `sdemodel()` again.

The output of `sdefit()` is a list with two components, named `fit` and `more`. The first, `fit`, is the result of the minimization of the sum of squares from `uvector()`, performed by the nonlinear least-squares function `nls()`.

It contains the ML parameter estimates, except for the σ's. The second component, `more`, gives the σ estimates, the maximized log-likelihood value, and AIC and BIC statistics.

3.1 Example 1, single unit with additive process noise

This example is from section 3.2.1 and Appendix C.1 of García (2019).

`Loblolly` is a data set included with **R**, containing height and age data for 14 trees. Heights $h_i = H(t_i)$ are in feet, and ages t_i are in years. For this example we use the 6 observations from the first tree, tree #301:

```
lob301 <- Loblolly[Loblolly$Seed == 301, ]
```

A suitable model is

$$\frac{\mathrm{d}H^c}{\mathrm{d}t} = b(a^c - H^c) \ ,$$

which on integration gives the commonly used Richards growth curve. The special case $c = -1$ gives the logistic, and c close to 0 approximates the Gompertz curve, two models that have been used in previous analyses of this data. Let the height be 0 at age 0.

Assume additive process noise

$$\mathrm{d}H^c = b(a^c - H^c)\,\mathrm{d}t + \sigma_p\,\mathrm{d}W \ ,$$

and measurement error

$$h_i^c = H^c(t_i) + \epsilon_i \ ,$$

where the ϵ_i are independent normally distributed with mean 0 and variance σ_m^2.

In **resde**, the model specification is

```
m <- sdemodel(phi=~x^c, beta0=~b*a^c, beta1=~-b)  # else, defaults

## Model:
##        y = phi(x, c) = x^c
##        y' = phiprime(x, c) = c * x^(c - 1)
##        dY = (b * a^c + -b * Y) dt + sigmap * dW
##        Y(0) = phi(0, c)
##        yi = Y(ti) + sigmam * ei
## Parameters:
##        a, b, c
```

Now, the parameter estimation:

```
f <- sdefit(m, x="height", t="age", data=lob301,
            start=c(a=60, b=0.1, c=1))

## Warning: Solution at boundary, it may be a local optimum.
Compare logLik with mum = 0

f

## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = NULL, beta0 = b
* a^c,     beta1 = -b, eta = eta, eta0 = 0, x0 = 0, t0 = 0, lambda
= list(c = c),     mum = 1, mu0 = 0, mup = 1, sorted = TRUE)
##    data: lob301
##       a       b       c     eta
## 72.5459  0.0967  0.5024  1.0000
##  residual sum-of-squares: 1.327
##
## Algorithm "port", convergence message: relative convergence (4)
##
## $more
##     sigma_p     sigma_m      logLik         AIC         BIC
##  0.00000000  0.04866015 -3.98808081 17.97616162 16.93495896
```

The parameter estimates are $a = 72.55$, $b = 0.0967$, $c = 0.5024$, $\sigma_p = 0$, and $\sigma_m = 0.04866$, with a maximized log-likelihood value of -4.0. I'll explain `eta` shortly. This would suggest that most of the variability arises from measurement errors. However, $\sigma_p = 0$ is at the boundary of the admissible values, and therefore this could be a local optimum different from the global one. To confirm, we force $\sigma_m = 0$ (or rather $\mu_m \sigma_m = 0$), as suggested by the warning:

```
m <- sdemodel(phi=~x^c, beta0=~b*a^c, beta1=~-b, mum=0)

## Model:
##        y = phi(x, c) = x^c
```

```
##         y' = phiprime(x, c) = c * x^(c - 1)
##         dY = (b * a^c + -b * Y) dt + sigmap * dW
##         Y(0) = phi(0, c)
##         yi = Y(ti)
## Parameters:
##         a, b, c
```

```
sdefit(m, x="height", t="age", data=lob301,
       start=c(a=60, b=0.1, c=1))
```

```
## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = NULL, beta0 =
b * a^c,     beta1 = -b, eta = 0, eta0 = 0, x0 = 0, t0 = 0, lambda
= list(c = c),     mum = 0, mu0 = 0, mup = 1, sorted = TRUE)
##    data: lob301
##       a       b       c
## 71.5940  0.1011  0.4863
##  residual sum-of-squares: 1.897
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 7.327e-07
##
## $more
##     sigma_p      logLik         AIC         BIC
##  0.03273267 -5.05854635 18.11709269 17.28413057
```

The log-likelihood is a little worse at -5.1 (the AIC and BIC are not directly comparable, because there is one less free parameter). Only log-likelihood differences are relevant, and some arguments suggest that differences of around 2 units might be considered as "significant". Therefore, this data does not provide enough information about the values of the sigmas. One could also say that the model is over-parameterized. This seems to be typical, at least with short time series.

As mentioned before, the first component in the output from `sdefit()` is the output from `nls()`. It shows the `uvector()` call used in García (2019), this time automatically generated within `sdefit()`. There, `eta` is the relative measurement variance $\eta = \frac{\sigma_m^2}{\sigma_p^2+\sigma_m^2}$. In the first run, η was constrained to

be between 0 and 1 by using algorithm `port` in `nls()`, which allows bounds on the optimization variables.

A little more information can be extracted from the `nls` fit:

```
summary(f$fit)

##
## Formula: 0 ~ uvector(x = height, t = age, unit = NULL, beta0 =
b * a^c,
##     beta1 = -b, eta = eta, eta0 = 0, x0 = 0, t0 = 0, lambda =
list(c = c),
##     mum = 1, mu0 = 0, mup = 1, sorted = TRUE)
##
## Parameters:
##     Estimate Std. Error t value Pr(>|t|)
## a   72.54593    5.06284  14.329  0.00484 **
## b    0.09670    0.01296   7.459  0.01750 *
## c    0.50244    0.03931  12.781  0.00607 **
## eta  1.00000    0.43602   2.293  0.14882
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8147 on 2 degrees of freedom
##
## Algorithm "port", convergence message: relative convergence (4)
```

Note that here `Residual standard error`, and `residual sum-of-squares` in `fit`, correspond to the values from `uvector()` and not to h_i or h_i^c.

3.2 Example 2, single unit with multiplicative process noise

Section 3.2.1 and Appendix C.2 of García (2019).

In the previous example, consider now a multiplicative process noise instead of additive:

$$\mathrm{d}H^c = b(a^c - H^c)(\,\mathrm{d}t + \sigma_p\,\mathrm{d}W) = b(a^c - H^c)\,\mathrm{d}t + b\sigma_p(a^c - H^c)\,\mathrm{d}W\ .$$

The so-called Lamperti transform leads to a model

$$\mathrm{d}Y = -b\,\mathrm{d}t + b\sigma_p\,\mathrm{d}W\ ,$$

with

$$Y = \varphi(H) = \ln|a^c - H^c| \, .$$

Assume measurement errors of the form

$$y_i = Y(t_i) + \epsilon_i \, , \quad \epsilon_i \sim \mathrm{N}(0, \sigma_m^2) \, .$$

In **resde**,

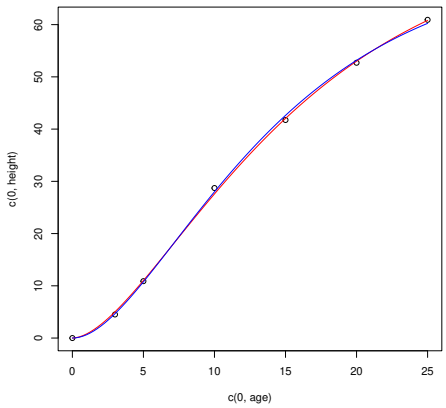
```
m <- sdemodel(~log(abs(a^c - x^c)), beta0=~-b, beta1=0, mup=~b)

## Model:
##        y = phi(x, a, c) = log(abs(a^c - x^c))
##        y' = phiprime(x, a, c) = {;    .e2 <- a^c - x^c;    -(c
* x^(c - 1) * sign(.e2)/abs(.e2));}
##        dY = (-b + 0 * Y) dt + b * sigmap * dW
##        Y(0) = phi(0, a, c)
##        yi = Y(ti) + sigmam * ei
## Parameters:
##        a, b, c

sdefit(m, x="height", t="age", data=lob301,
       start=c(a=70, b=0.1, c=1))

## Warning:  Solution at boundary, it may be a local optimum.
Compare logLik with mum = 0

## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = NULL, beta0 =
-b, beta1 = 0,     eta = eta, eta0 = 0, x0 = 0, t0 = 0, lambda =
list(a = a,          c = c), mum = 1, mu0 = 0, mup = b, sorted =
TRUE)
##    data: lob301
##        a        b        c      eta
## 77.10687  0.08405  0.54946  1.00000
##  residual sum-of-squares: 1.154
##
## Algorithm "port", convergence message: relative convergence (4)
##
```

```
## $more
##     sigma_p     sigma_m      logLik         AIC         BIC
##  0.00000000  0.01576676 -3.56821125 17.13642250 16.09521985
```

The derivative produced by `Deriv::Deriv()` is a little messy, but it works. On aesthetic grounds, one might include in `sdemodel` the simpler equivalent `phiprime = c*x^(c-1)/(x^c - a^c)`.

The log-likelihood is not significantly different from the one from the additive model. Not much difference either in the fitted curves:

```
plot(c(0,height) ~ c(0, age), data=lob301)
curve(77.10687 * (1 - exp(-0.08405 * x))^(1/0.54946), add=T,
      col="red")
curve(72.5459 * (1 - exp(-0.0967 * x))^(1/0.5024), add=T,
      col="blue")
```



As with any optimization, one should be careful and try different starting values, because local optima can occur. For instance, in the last run, changing the start to `a=60` produces a different (worse) solution.

4 Hierarchical (two-level) models

Often the data consists of several measurements on each of a number of units. For instance, the measurements on each of the 14 trees in `Loblolly`. This is known as panel, repeated measures, or longitudinal data, and gives

rise to hierarchical or multilevel models; **resde** can deal with two hierarchical levels. Some parameters may vary among units (local), while others are common to all units (global), possibly after a re-parameterization of the original model. Local parameters may be treated as fixed unknown values. Frequently, interest lies mainly on the globals, and the locals are then called nuisance parameters.

A popular alternative is to think of the local parameters as, in some sense, "random". For instance, the data may be thought of as a random sample from some hypothetical super-population, in which the local parameters have a Normal distribution. In mixed-effects terminology, the globals and the means of the locals are *fixed effects*, while the deviations of the locals from their means are *random effects*, usually normally distributed. The units are called *groups* (of observations). The advantage is that then there are less parameters to be estimated: instead of one local value for each unit, there is now only a mean and a variance, and possibly also covariances between locals. On the other hand, all these are additional assumptions. In particular, the assumptions are not realistic if the units are not a simple random sample from the population. Also, estimation is more complicated, and not as robust as minimizing a sum of squares.

4.1 Example 3, fixed local parameters

Section 3.3.1 and Appendix E of García (2019).

Consider fitting Richards SDE models simultaneously to all the 14 trees in the `Loblolly` data set. The parameterization can be important here, so we use the Box-Cox transformation, ensuring that a and b are proper scale parameters:

$$\begin{aligned} Y =& (H/a)^{(c)} \\ \mathrm{d}Y =& -Y\,\mathrm{d}(bt) + \sigma_p\,\mathrm{d}W(bt) = -bY\,\mathrm{d}t + \sqrt{|b|}\sigma_p\,\mathrm{d}W(t)\ , \end{aligned}$$

where $x^{(c)}$ denotes the Box-Cox transformation

$$x^{(c)} = \begin{cases} \frac{x^c-1}{c} \text{ if } c \neq 0\ , \\ \ln x \text{ if } c = 0\ . \end{cases} \tag{5}$$

It can be seen that for $c \neq 0$ one obtains the same Richards differential equation as before, but now the Gompertz model is also included, when $c = 0$.

Assume that the measurement error is negligible compared to the process noise, i.e., $\sigma_m = 0$, and that the curves start at the origin $t = 0, H = 0$.

Using the Box-Cox transformation `bc()` included in **resde**, the model is

```
m <- sdemodel(phi=~bc(x/a, c), beta0=0, beta1=~-b,
              mup=~sqrt(abs(b)), mum=0)

## Model:
##        y = phi(x, a, c) = bc(x/a, c)
##        y' = phiprime(x, a, c) = if (abs(c) < 1e-300) 1/x else
exp(c * (log(x) - log(a)))/x
##        dY = (0 + -b * Y) dt + sqrt(abs(b)) * sigmap * dW
##        Y(0) = phi(0, a, c)
##        yi = Y(ti)
## Parameters:
##        a, b, c
```

Again, the generated derivative is a little more complex than necessary. One could have included `phiprime = (x/a)^(c-1)/a`, or `phiprime = bc_prime(x/a, c)/a`.

For hierarchical models, one must indicate a variable that identifies the units in the parameter `unit` of `sdefit()`, `"Seed"` in this case. Also, `global` and `local` are used instead of `start`. Starting values for locals can be vectors with one value for each unit, or a single value that applies to all.

First, take a as local, that is, the asymptotes a_j vary from tree to tree:

```
alocal <- sdefit(m, x="height", t="age", unit="Seed",
          data=Loblolly, global=c(b=0.1, c=0.5), local=c(a=70))
alocal

## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = Seed, beta0 = 0,
beta1 = -b,     eta = 0, eta0 = 0, x0 = 0, t0 = 0, lambda = list(a
= a[Seed],          c = c), mum = 0, mu0 = 0, mup = sqrt(abs(b)),
sorted = TRUE)
##    data: Loblolly
```

```
##         b         c        a1        a2        a3        a4        a5
##   0.09472   0.49182 68.36652 69.11597 71.87593 70.69004 70.44041
##        a6        a7        a8        a9       a10       a11       a12
## 71.38287 72.90629 70.92201 74.01903 74.77265 75.44945 76.41765
##       a13       a14
## 76.91872 78.84127
##  residual sum-of-squares: 40.35
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 8.614e-06
##
## $more
##       sigma_p       logLik          AIC          BIC
##    0.03358892 -88.39580749 210.79161498 252.11550056
```

Now try a global and b local:

```
(blocal <- sdefit(m, x="height", t="age", unit="Seed",
           data=Loblolly, global=c(a=70, c=0.5), local=c(b=0.1)))

## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = Seed, beta0 =
0, beta1 = -b[Seed],     eta = 0, eta0 = 0, x0 = 0, t0 = 0, lambda =
list(a = a, c = c),     mum = 0, mu0 = 0, mup = sqrt(abs(b[Seed])),
sorted = TRUE)
##    data: Loblolly
##         a         c        b1        b2        b3        b4        b5
## 73.08143   0.49156   0.08912   0.09082   0.09495   0.09053   0.08915
##        b6        b7        b8        b9       b10       b11       b12
##   0.09111   0.09496   0.08957   0.09680   0.09819   0.09843   0.09984
##       b13       b14
##   0.09984   0.10313
##  residual sum-of-squares: 37.35
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 2.77e-06
##
```

```
## $more
##      sigma_p       logLik          AIC          BIC
##   0.03231109 -85.15200926 204.30401852 245.62790410
```

The log-likelihood (or equivalently, the AIC or BIC) indicates that this model fits the data slightly better than the one with a local.

Finally, with both a and b locals,

```
(ablocal <- sdefit(m, x="height", t="age", unit="Seed",
           data=Loblolly, global=c(c=0.5), local=c(a=70, b=0.1)))

## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = Seed, beta0
= 0, beta1 = -b[Seed],      eta = 0, eta0 = 0, x0 = 0, t0 = 0,
lambda = list(a = a[Seed],         c = c), mum = 0, mu0 = 0, mup =
sqrt(abs(b[Seed])), sorted = TRUE)
##    data: Loblolly
##        c       a1       a2       a3       a4       a5       a6
##  0.49062 68.38819 67.44865 68.64479 73.63619 76.07754 74.86101
##       a7       a8       a9      a10      a11      a12      a13
## 72.01170 76.95452 72.13086 71.98484 73.72215 74.17723 75.82487
##      a14       b1       b2       b3       b4       b5       b6
## 75.91534  0.09488  0.09798  0.10084  0.09010  0.08617  0.08933
##       b7       b8       b9      b10      b11      b12      b13
##  0.09646  0.08569  0.09818  0.09977  0.09783  0.09867  0.09670
##      b14
##  0.09976
##  residual sum-of-squares: 30.81
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 6.465e-07
##
## $more
##      sigma_p       logLik          AIC          BIC
##   0.02936758 -77.06103324 214.12206648 287.04657045
```

Here the AIC or BIC have to be used for the comparison, because the number of free parameters is different. For AIC and BIC smaller is better. They indicate that this is worse than the one-local versions.

Other structures could be defined by re-parameterization, substituting functions of other global and local parameters for a, b or c.

By the way, if you are mystified by the strangely-looking Box-Cox transformation, eq. (5), don't be. It is basically a power transformation with a twist, considering that linear (or more precisely affine) transformations are often "uninteresting". The point of it is that the limit $\lim_{c\to 0}(x^c-1)/c = \ln x$ makes the transformation continuous at $c = 0$, as a function of c. In the process including the logarithm as a special case. Hiding those details inside the definition, using $x^{(c)}$ saves us the hassle of having to talk about the special case all the time. And the logarithmic transformation comes along for free. We'll come back to this, with a vengeance, in Section 5.4.1. Physicists discovered the transformation independently, looking from the other end, calling it a *generalized logarithm*, denoted by $\ln_c(x)$.

4.2 Example 4, random local parameters

Section 3.3.2 and Appendix F of García (2019).

The mixed effects method uses `nlme()` instead of `nls()`. This is chosen by setting `method = "nlme"` in `sdefit()`. The default is `method = "nls"`.

Let us fit the b-local version from Example 3:

```
(blocal_mx <- sdefit(m, x="height", t="age", unit="Seed",
             data=Loblolly, global=c(a=70, c=0.5), local=c(b=0.1),
               method="nlme"))

## Error in nlme.formula(frml, data, fixed = fixed, random = random,
groups = groups, : step halving factor reduced below minimum in
PNLS step
```

Convergence fails. There is an optional argument `control` in `sdefit()`, which accepts a list of control parameters to be passed on to `nlme()` or `nls()`. Use it to increase the "PNLS tolerance" to 0.01, from the default 0.001:

```
(blocal_mx <- sdefit(m, x="height", t="age", unit="Seed",
            data=Loblolly, global=c(a=70, c=0.5), local=c(b=0.1),
             method="nlme", control = nlme::nlmeControl(pnlsTol =
              0.01)))

## $fit
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: frml
##   Data: data
##   Log-likelihood: -101.7808
##   Fixed: fixed
##           a           c           b
## 73.43552332  0.49382756  0.09380411
##
## Random effects:
##  Formula: b ~ 1 | Seed
##                    b  Residual
## StdDev: 0.003814885 0.7307106
##
## Number of Observations: 84
## Number of Groups: 14
##
## $more
##       sigma_p        logLik           AIC           BIC
##    0.03316396 -101.78079272  213.56158544  225.71566943
```

This time it worked. The log-likelihood is not comparable to the one for fixed locals (different numbers of parameters), but we can compare the AIC and BIC criteria. The AIC values, 214 *vs.* 204 in Example 3, would suggest that in this instance fixed locals is better than mixed effects. However, the opposite is true according to the BIC, 226 *vs.* 246. This is because the BIC penalizes the difference in parameter numbers more heavily than the AIC.

In this formulation b is a random variable, so that it does not make sense to speak of *estimates*, but `nlme()` provides "predictions":

```
coef(blocal_mx$fit)

##           a         c          b
```

```
## 329 73.43552 0.4938276 0.08972474
## 327 73.43552 0.4938276 0.09095012
## 325 73.43552 0.4938276 0.09399673
## 307 73.43552 0.4938276 0.09086502
## 331 73.43552 0.4938276 0.08972086
## 311 73.43552 0.4938276 0.09128954
## 315 73.43552 0.4938276 0.09406932
## 321 73.43552 0.4938276 0.09009376
## 319 73.43552 0.4938276 0.09535720
## 301 73.43552 0.4938276 0.09630472
## 323 73.43552 0.4938276 0.09646256
## 309 73.43552 0.4938276 0.09737925
## 303 73.43552 0.4938276 0.09741947
## 305 73.43552 0.4938276 0.09962427
```

5 Additional features and advanced usage

5.1 Derivatives

`Deriv()` seems to do a good job of producing transformation derivatives, although as we have seen, sometimes not in the simplest possible form. If desired, perhaps for troubleshooting, the name of a user-supplied derivative function can be given in the argument `phiprime` of `sdefit()`. The same can be done for the transformation `phi`. See `phi.R` for suitable function templates.

5.2 Under the hood

The aim of **resde** is to facilitate the application of the function `uvector()` from García (2019). That function uses tricks based on work by Furnival and by Box and Cox to compute values such that minimizing the sum of their squares produces maximum-likelihood parameter estimates. The sum of squares is minimized with `nls()` from package **stats**, or with `nlme()` from package **nlme**. Setting up a call to `uvector()`, as done in García (2019), can be rather complicated, a process that is "mechanized" by `sdemodel()` and `sdefit()`. The generated call to `uvector()` can be

seen with `formula(f$fit)` or `(f$fit)$call`, where `f` is the output from `sdefit()`. Conceivably, there may be applications where it might be necessary to use `uvector()` directly.

When there is both process and measurement noise, **resde** uses internally an additional parameter, `eta`, that corresponds to the relative measurement error $\eta = \sigma_m^2/(\sigma_p^2 + \sigma_m^2)$. It must take values between 0 ($\sigma_m = 0$), and 1 ($\sigma_p = 0$). With `nls()`, algorithm `port` performs the constrained optimization. For mixed effects, `nlme()` does not allow constraints. Therefore, `optimize()` is used in that case to perform a one-dimensional optimization over `eta`, calling `nlme()` at each step.

Here is an example, freeing σ_m in the b-local model of Example 3:

```
m <- sdemodel(phi=~bc(x/a, c), beta0=0, beta1=~-b,
              mup=~sqrt(abs(b)))

## Model:
##        y = phi(x, a, c) = bc(x/a, c)
##        y' = phiprime(x, a, c) = if (abs(c) < 1e-300) 1/x else
exp(c * (log(x) - log(a)))/x
##        dY = (0 + -b * Y) dt + sqrt(abs(b)) * sigmap * dW
##        Y(0) = phi(0, a, c)
##        yi = Y(ti) + sigmam * ei
## Parameters:
##        a, b, c

sdefit(m, x="height", t="age", unit="Seed", data=Loblolly,
       global=c(a=70, c=0.5), local=c(b=0.1))

## Warning:  Solution at boundary, it may be a local optimum.
Compare logLik with mup = 0

## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = Seed, beta0
= 0, beta1 = -b[Seed],      eta = eta, eta0 = 0, x0 = 0, t0 = 0,
lambda = list(a = a,           c = c), mum = 1, mu0 = 0, mup =
sqrt(abs(b[Seed])), sorted = TRUE)
##    data: Loblolly
##        a        c      eta       b1       b2       b3       b4
```

```
## 73.08144  0.49156  0.00000  0.08912  0.09082  0.09495  0.09053
##       b5       b6       b7       b8       b9      b10      b11
##  0.08915  0.09111  0.09496  0.08957  0.09680  0.09819  0.09843
##      b12      b13      b14
##  0.09984  0.09984  0.10313
##  residual sum-of-squares: 37.35
##
## Algorithm "port", convergence message: relative convergence (4)
##
## $more
##      sigma_p      sigma_m       logLik          AIC          BIC
##   0.03231109   0.00000000 -85.15200926 206.30401852 250.05872090
```

Note the call to `uvector()` in the *model* item from `nls()`, and the presence of the parameter `eta`. Estimates were the same as before. The AIC and BIC differ because of the additional parameter. We explore further the error structure next.

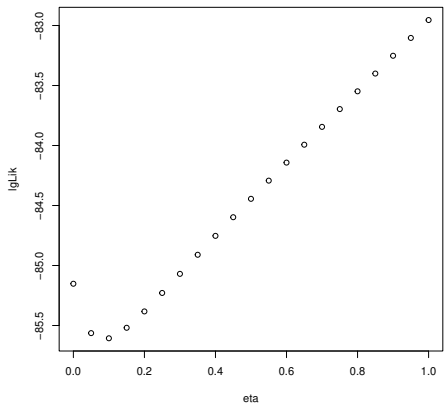
5.3 Fixing `eta`

We already saw how to specify models without measurement error or without process noise by setting the sigma multipliers `mum=0` or `mup=0`, respectively. For more flexibility, there is a "hidden" feature for specifying a relative error magnitude through η (`eta`): the argument `known` in `sdefit()`, which fixes parameters at given values, accepts also a value for `eta`.

Let us use this to investigate the relationship between the maximized log-likelihood and η, plotting η's *profile log-likelihood*:

```
lgLik <- eta <- seq(from=0, to=1, by=0.05)
for(i in seq_along(eta)){
  lgLik[i] <- (sdefit(m, x="height", t="age", unit="Seed",
                data=Loblolly, global=c(a=73, c=0.49),
                local=c(b=0.095), known=c(eta=eta[i]))$more
               )["logLik"]
}

## Warning:  Solution at boundary, it may be a local optimum.
Compare logLik with mup = 0
```

```
## Warning: Solution at boundary, it may be a local optimum.
Compare logLik with mum = 0

plot(lgLik ~ eta)
```



We see that there are local optima at $\eta = 0$ ($\sigma_m = 0$), and at $\eta = 1$ ($\sigma_p = 0$). This is what was causing trouble with the optimizations. However, the significance of the log-likelihood differences is marginal. One may conclude that the data cannot tell us much about the error structure, so that we are justified in choosing it from prior knowledge. E.g., if the height observations were derived from tree rings, the measurement errors may be negligible compared to the environmental noise, and $\sigma_m = 0$ would be reasonable.

5.4 On transformations

5.4.1 A unifying transformation

It has been found that, allowing linear transformations of x and t, nearly all the growth curve equations in the literature can be unified in a family of functions with two shape parameters:

$$U(x, \alpha, \beta) \equiv -[-x^{(\alpha)}]^{(\beta)} = t \ , \tag{6}$$

in terms of the Box-Cox transformation defined in eq. (5). For instance, the Richards is the spacial case $\beta = 0$, and the Hosfeld IV corresponds to

$\alpha = -1, \beta < 0$ (Chakraborty (2019), and `https://github.com/ogarciav/grex/`).

In eq. (6), x ranges between 0 and 1. If x goes from 0 up to an asymptote a, we have $F^{-1}(x) = U(x/a, \alpha, \beta)$. It is also possible to have negative scale factors, which reverse the x and t axes, and then $F^{-1}(x) = U(1 - x/a, \alpha, \beta)$.

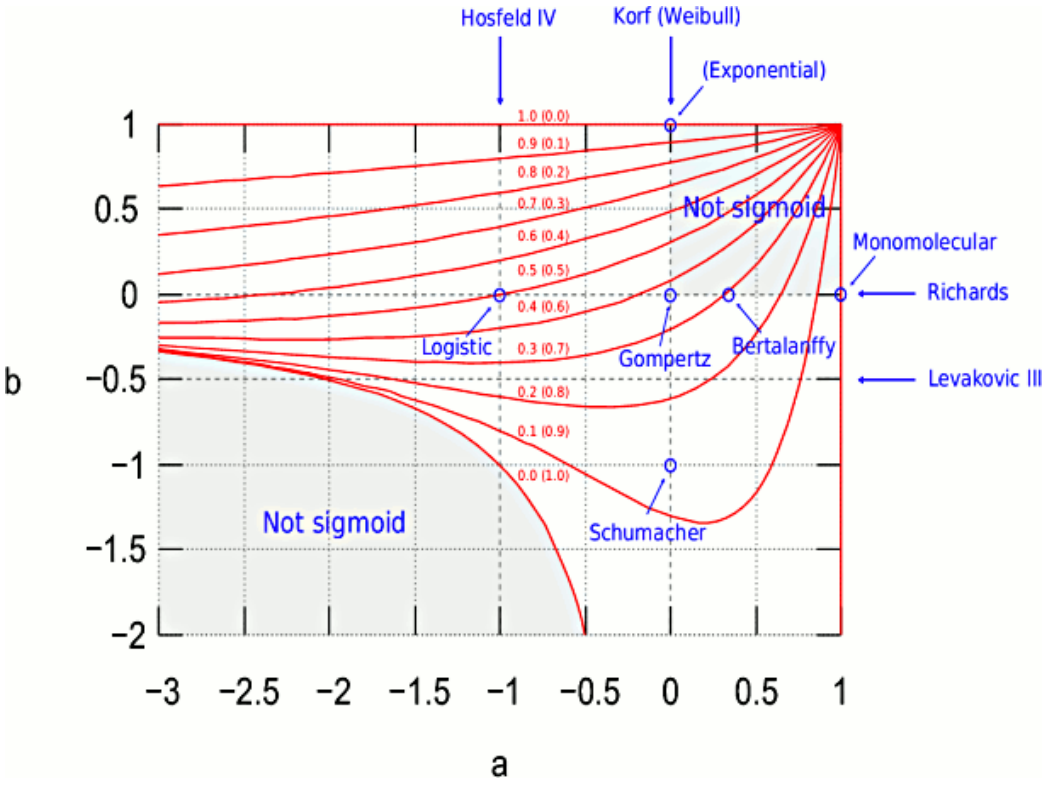
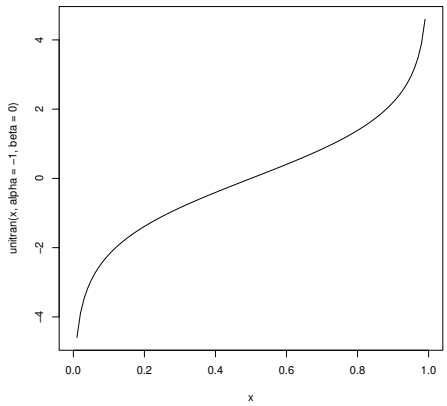


Figure 1: Sigmoid growth equations determined by the unified transformation with parameters $a = \alpha$ and $b = \beta$. Items in parenthesis correspond to negative scale factors on x and t (reversed axes). Contours indicate the height of the inflection point relative to the range of x. From García (2005, 2008), see `https://web.unbc.ca/~garcia/growth&yield/grex/` or Chakraborty (2019) for model references.

Figure 1 shows the useful range of α and β, and the correspondence to common growth equation models. The unifying transformation $U(x, \alpha, \beta)$ has been implemented in the function `unitran()`, see `?unitran` for details. As an example, the following shows the inverse logistic:

```
curve(unitran(x, alpha=-1, beta=0))
```



The function can also be called with names: `curve(unitran(x, "logistic"))` produces the same result.

The unified transformation can be useful when hunting for a suitable model form. With tree #301 again, applying eq. (7), let us try $y = U(x/a, \alpha, \beta)$. This is similar to what we did in Example 2 for the Richards model.

```
m <- sdemodel(phi=~unitran(x/a, alpha=alpha, beta=beta,
  reverse="no"), phiprime=~unitran_prime(x/a, alpha=alpha,
  beta=beta, reverse="no")/a, beta0=~b, beta1=0, mum=0)

## Model:
##        y = phi(x, a, alpha, beta) = unitran(x/a, alpha = alpha,
beta = beta, reverse = "no")
##         y' = phiprime(x, a, alpha, beta) = unitran_prime(x/a,
alpha = alpha, beta = beta, reverse = "no")/a
##        dY = (b + 0 * Y) dt + sigmap * dW
##        Y(0) = phi(0, a, alpha, beta)
##        yi = Y(ti)
## Parameters:
##        a, alpha, b, beta

(f <- sdefit(m, x="height", t="age", data=lob301, start=c(a=70,
  b=0.1, alpha=0.5, beta=0)))
```

```
## $fit
## Nonlinear regression model
##   model: 0 ~ uvector(x = height, t = age, unit = NULL, beta0
= b, beta1 = 0,      eta = 0, eta0 = 0, x0 = 0, t0 = 0, lambda =
list(a = a, alpha = alpha,         beta = beta), mum = 0, mu0 = 0,
mup = 1, sorted = TRUE)
##    data: lob301
##         a         b     alpha      beta
## 132.39046   0.04348   0.33035  -1.23246
##  residual sum-of-squares: 0.7469
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 2.275e-07
##
## $more
##     sigma_p      logLik         AIC         BIC
##  0.00316209 -2.26269562 14.52539123 13.48418858
```

As expected, `summary(f$fit)` indicates over-parameterization. But the result might suggest trying the Levacovic, Korf or Schumacher models as more parsimonious named alternatives, e.g., by moving `beta` out of `start` and into `known=c(beta=-0.5)`.

One could also try the reversed form with `1-x/a`. The default `reverse="auto"` in `unitran()` does that automatically, but the discontinuity at $\beta = 0$ causes the optimization to fail. Or perhaps one could experiment with the form corresponding to eq. (8) below. There are limits to what is worth doing with 6 data points, but you get the idea.

5.4.2 More general curves

Consider any growth curve (or other type of) model with trajectories given by some function

$$x = F(t) \ .$$

One can write

$$F^{-1}(x) = t \ .$$

Therefore, the transformation on the left-hand side obeys a (trivial) linear differential equation:

$$y = F^{-1}(x) \quad \rightarrow \quad \frac{\mathrm{d}y}{\mathrm{d}t} = 1 \ .$$

Any linear function of that transformation will also work:

$$y = pF^{-1}(x) + q \quad \rightarrow \quad \frac{\mathrm{d}y}{\mathrm{d}t} = p \ , \tag{7}$$

for any constants $p \neq 0$ and q.

Somewhat more interesting is

$$y = \exp[F^{-1}(x)] \quad \rightarrow \quad \frac{\mathrm{d}y}{\mathrm{d}t} = \exp[F^{-1}(x)] \frac{\mathrm{d}F^{-1}(x)}{\mathrm{d}t} = y \ .$$

Or more generally,

$$y = p \exp[qF^{-1}(x)] + r \quad \rightarrow \quad \frac{\mathrm{d}y}{\mathrm{d}t} = q(y - r) \ . \tag{8}$$

In particular cases, p, q, r can be chosen to simplify the transformation.

E.g., for the Richards (or Bertalanffy-Richards) equation,

$$x = F(t) = a[1 - \mathrm{sgn}(c)\mathrm{e}^{-b(t-t_0)}]^{1/c} \ ,$$
$$t = F^{-1}(x) = t_0 - \tfrac{1}{b} \ln|(x/a)^c - 1| \ .$$

Some transformations that follow from eq. (7):

$$y = \ln|(x/a)^c - 1| \ , \quad y = \ln|x^c - a^c| \ , \quad y = \ln \frac{a^c - x^c}{c} \ .$$

For eq. (8), the simplest transformation is $y = x^c$.

As another example, in the Hosfeld IV equation,

$$x = F(t) = \frac{at^c}{t^c + b} \ ,$$
$$t = F^{-1}(x) = \left(\frac{bx}{a - x} \right)^{1/c} \ .$$

For eq. (7), one could use

$$y = \left(\frac{x}{a - x} \right)^k = (a/x - 1)^{-k} \ ,$$

and for eq. (8),

$$y = \exp\left[\left(\frac{x}{a - x} \right)^k \right] = \exp[(a/x - 1)^{-k}] \ .$$

5.4.3 General differential equations

A univariate time-invariant (aka autonomous) differential equation has the form

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x) \; .$$

These are often preferred to equations that include both x and t on the right-hand side, because natural laws are not supposed to change from one week to the next. From $\mathrm{d}x/f(x) = \mathrm{d}t$ we get

$$\int \frac{\mathrm{d}x}{f(x)} = t \; .$$

Defining this integral as $F^{-1}(x)$ we are back to the situation above. Of course, luck is needed for the integral to have a closed form (analytical solution). On the other hand, it might be interesting to see how well the estimation method works if `phi()` and `phiprime()` are calculated by numerical integration.

Note: These identities are sometimes useful:

$$F'(t) = 1/(F^{-1})'[F(t)] \; , \quad (F^{-1})'(x) = 1/F'[F^{-1}(x)] \; .$$

Proof: Differentiate $F^{-1}[F(t)] = t$ or $F[F^{-1}(x)] = x$.

For SDEs, it may be possible to linearize the deterministic part (aka the *trend* or *drift*). Or reduce to a constant the stochastic term (*diffusion* or *volatility*), through the Lamperti transform (García, 2019, sec. 3). In general, it is not possible to get the desired form for both, but often one of the terms is less important than the other. At any rate, there are good theoretical reasons for why linearity and Gaussianity should go well together.

References

Chakraborty, B., Bhowmick, A. R., Chattopadhyay, J., and Bhattacharya, S. (2019) A novel unification method to characterize a broad class of growth curve models using relative growth rate. *Bulletin of Mathematical Biology 81*(7) 2529–2552. (`https://doi.org/10.1007/s11538-019-00617-w`).

García, O. (2013) Forest stands as dynamical systems: An introduction. *Modern Applied Science 7*(5), 32–38. (https://doi.org/10.5539/mas.v7n5p32).

García, O. (2019) Estimating reducible stochastic differential equations by conversion to a least-squares problem. *Computational Statistics*, 2019, 34, 23–46. (https://doi.org/10.1007/s00180-018-0837-4).

Oscar García
May 18, 2023
(First version: November 13, 2020)