Practical introduction to the `rodeo` package

david.kneis@tu-dresden.de

June 25, 2015

Contents

1 Why would you use the `rodeo` package? 1

2 Target languages of `rodeo` 2

3 How to obtain package and documentation 2

4 `rodeo`'s input 2
4.1 Conventions . . . 2
4.1.1 Matrix notation . . . 2
4.1.2 Names of identifiers . . . 3
4.1.3 Mathematical operators . . . 3
4.2 Required tables . . . 3
4.2.1 Declaration of variables and parameters . . . 4
4.2.2 Declaration of functions . . . 4
4.2.3 Declaration of process rates . . . 5
4.2.4 Tabular representation of the stoichiometry matrix . . . 6

1 Why would you use the `rodeo` package?

The `rodeo` package can facilitate development and use of ODE-based simulation models. The basic idea is to store the ODE system (forming the model's core) in plain text files or spreadsheets. This is in contrast to the usual way where the model equations are directly implemented in a computer code.

The advantages are as follows:

- The model is formulated independently of a particular computer language. Translators could be written for any target language or modeling package. This is a key to making the model re-usable.
- Model equations and the related documentation really form a unit.
- It becomes possible to visualize the structure of a model. In addition, it may be possible to automatically validate a model (not yet done).
- The `rodeo` package currently supports code generation for the target languages listed in Sec. 2. Use of a compiled language like Fortran is indispensible for building high-performance code.

- The `rodeo` package can generate code for both zero-dimensional models (where all state variables are scalars) and spatially distributed models (e. g. reactive transport models).
- Is is possible to automatically generate user interfaces from the text/table-based model description.

2 Target languages of `rodeo`

The currently supported target languages are:

R Suitable for models with few state variables and short periods of integration.

Fortran 95 For demanding models and all cases where computation times need to be reduced to a minimum.

3 How to obtain package and documentation

The package's repository is `https://github.com/dkneis/rodeo`. At present, there is no CRAN version. To install and load the package, one can use, for example:

```
# Package allowing direct install of other packages from github
if (!("devtools" %in% installed.packages()[,1]))
  install.packages("devtools")
library("devtools")

# Now install and load rodeo
if (!("rodeo" %in% installed.packages()[,1]))
  install_github("dkneis/rodeo", build_vignettes=TRUE)
library("rodeo")
```

If the package needs to be re-installed after an update, one should use

```
remove.packages("rodeo")
```

prior to the installation commands above.

The primary documentation for `rodeo` is the packages' vignette. It can be accessed as follows:

```
library(rodeo)
vignette("rodeo")
```

4 `rodeo`'s input

4.1 Conventions

4.1.1 Matrix notation

`rodeo` expects the ODE system to be written in vector/matrix notatation (see `http://en.wikipedia.org/wiki/Petersen_matrix`). This can be illustrated

on a two-equations example with state variables Z, X and parameters k_d, k_a, X_{sat}. Using vector/matrix notation, the original differential eqations

$$\frac{d}{dt}Z = -k_d \cdot Z$$
$$\frac{d}{dt}X = -k_d \cdot Z \cdot s + k_a \cdot (X_{sat} - X)$$

can be written as

Vector of derivatives | Transposed stoichiometry matrix | Process rates

$$\begin{bmatrix} \frac{d}{dt}Z \\ \frac{d}{dt}X \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ -s & 1 \end{bmatrix} \cdot \begin{bmatrix} k_d \cdot Z \\ k_a \cdot (X_{sat} - X) \end{bmatrix}$$

By convention, the stoichiometry matrix has the names of the state variables as column headers and the names of the processes as row headers. Therefore, depending on the context, the matrix needs to be transposed (as in the example above).

4.1.2 Names of identifiers

Any user-defined names of state variables, parameters, and functions must also be valid identifiers in R and Fortran. Thus, names must start with a letter, optionally followed by additional letter(s), number(s), or underscore(s). They must not be reserved words in the mentioned langauges (like, e. g. 'for' or 'end'). A list of Fortran's reserverd words can be found, for example, at `http://fortranwiki.org/fortran/show/Keywords`.

Note that names should be regarded as case-insensitive (as in Fortran and in contrast to R). Hence, one cannot declare, e. g., two parameters 'k' and 'K'.

4.1.3 Mathematical operators

The four basic operators are guaranteed to be compatible with `rodeo` (since they are compatible with both R and Fortran). For power operations, the double asterisk (`**`) must be used instead of the acute (`^`).

Note: Language-specific operators can always be used within functions implemented in the respective target language. The same applies to functions which are not intrisic to all possible target languages (currently R and Fortran).

4.2 Required tables

To build a model with `rodeo`, the user needs to fill in several tables. There are two options how to practically do this:

1. Write plain ASCII text file with columns being separated by a designated character (recommended is either semi-colon or TAB).

2. Use spreadsheet software: Each table should be put on a single worksheet. At present, the file(s) must be saved in either `.xls` or `.xlsx` format for being imported with the R-package `XLConnect`.

Each table has its own set of mandatory column as detailed in the subsequent sections. Within a table, the columns may appear in any order.

4.2.1 Declaration of variables and parameters

The explicit declaration of variables and parameters serves two purposes. First, it forces the model developer to document all items occuring in the model's equations. Second, it eliminates the need for naming conventions to distinguish parameters from state variables. The columns typically appearing in the tables of variable or parameter declaration are listed in Tab. 1. Note that only the first three columns are required for basic uses of `rodeo`. The remaining non-mandatory columns are only needed when a model is run through the `rodeoApp` graphical user interface (GUI) or in a Monte-Carlo simulation (MCS).

Table 1: Columns required in tables declaring variables or parameters.

Column name	**Required**	**Contents**
name	always	See Sec. 4.1.2 for conventions.
unit	always	Text; strongly suggested but not currently checked.
description	always	Free text description.
user	GUI only	Must be `TRUE` or `FALSE`. Controls whether the item can be modified in GUI.
label	GUI only	Label used in GUI.
default	GUI only	Suggested value in GUI.
upper	MCS only	Upper limit of sampling range for MCS.
lower	MCS only	Lower limit of sampling range for MCS.

In general, it is advisable to declare variables and parameters individually using two separate tables. However, it is also possible to declare items of both categories in a common table. Such a table must have an additional column (e.g. named 'type') to distinguish between the two categories.

4.2.2 Declaration of functions

Any functions appearing in the process rate expressions or stoichiometry factor expressions need to be declared. This is true even for functions being intrinsic to the target languages of the `rodeo` code generator (see Sec.2). The required declaration information is summarized in Table 2.

Table 2: Columns required in the table of function declaration.

Column name	**Contents**
name	See Sec. 4.1.2 for conventions.
unit	Text; strongly suggested but not currently checked.
description	Free text description.

The sole declaration of a function is not sufficient to make it callable from within a source code: There must also be a definition of the function. In the context of `rodeo`, the following rules apply:

- If the function is intrinsic to all target languages (see Sec. 2), the function still needs to be declared but it must not be defined. Typical examples: $exp()$, $log()$.
- If the function is not intrinsic to any of the target languages (see Sec. 2), the function must be declared and defined. Typical examples: $fancyFunction()$.
- If the function is only intrinsic to only one/some of the target languages (see Sec. 2), a wrapper function must be declared under a different name and a definition must be supplied as well. The definition should make use of the intrinsic function(s) where applicable.

When using R as the target language, function definitions must be in the R language. When using another target language, functions needing a definition must be implemented in both the particular target language and R[1].

In cases where no functions appear in mathematical expressions at all, it is usually appropriate to provide an empty table (or a dummy table) with the column names listed in Table 2.

Important: Although the functions `min` and `max` are intrinsic to both R and Fortran, they behave differently in the two languages when fed with vector arguments (e. g. in spatially distributed models). If `rodeo` is used to generate R-code, one must generally use the functions `pmin` and `pmax` instead of the 3-letter variants. Only the latter functions work properly in the case of spatially distributed models (multi-box models). In order to write mathematical expressions is a language-independent manner, it is recommended to define the following wrapper functions in R

```
minimum = function (x,y) { pmin(x,y) }
maximum = function (x,y) { pmax(x,y) }
```

and Fortran

```
double precision function minimum (x,y)
double precision, intent(in):: x, y
minimum= min(x,y)
end function

double precision function maximum (x,y)
double precision, intent(in):: x, y
maximum= max(x,y)
end function
```

respectively.

4.2.3 Declaration of process rates

The table used to declare the process rates has four mandatory fields as described in Tab. 3.

[1]The R-implementation is required to compute stoichiometry factors outside the ODE-solver.

Table 3: Columns required in the table of process rates.

Column name	Contents
name	See Sec. 4.1.2 for conventions.
unit	Text; strongly suggested but not currently checked.
description	Free text description.
expression	Mathematical expression containing operators and declared identifiers (i. e. state variables, parameters, and functions). See also Sec. 4.1.

4.2.4 Tabular representation of the stoichiometry matrix

The stoichiometry matrix is not actually written as a matrix but as a table holding the concatenated columns (or rows). The required format is described in Tab. 4.

In real-world models, the stoichiometry matrix often contains many cells with zeros. These zeros indicate that the derivative of a particular state variable (denoted in the column header) is not *directly* linked to a particular process (denoted in the row header). Importantly, in the tabular representation of the matrix, all rows with a stoichiometry factor of zero can be skipped (don't need to be written). Hence, a complete stoichiometry table should consist of as many rows as there are non-zero cells in the stoichiometry matrix.

The rows of the stoichiometry table can be in any order. It may be a useful convention, however, to first provide all records for a state variable 'a' before continuing with another state variable 'b'.

Table 4: Columns required in the stoichiometry table.

Column name	Contents
variable	Names of state variables that have been declared (see Tab. 1).
process	Names of processes that have been declared in Tab. 3.
expression	Mathematical expressions for the stoichiometry factors that link processes and variables. These expressions may contain operators and declared identifiers (i. e. state variables, parameters, and functions). See also Sec. 4.1.