Peccary, a metaprogramming open-source R platform to improve pharmacometrics efficiency [version 1; peer review: 1 approved with reservations]

Thibaud Derippe1-3, Xavier Declèves2, Sylvain Fouliard1

1Institut de Recherches Internationales Servier, Suresnes, 75073, France
2Université Paris Cité, Inserm, UMRS-1144, Optimisation Thérapeutique en Neuropsychopharmacologie, Paris, 75006, France
3Department of Pharmaceutical Sciences, University at Buffalo, State University of New York, Buffalo, New York, 14215, USA

Abstract

Background: A pharmacometrics (PMx) workflow usually requires several software tools to cover all the steps from data analysis to model evaluation and simulations. However, these tools do not always communicate well together, compromising the efficiency of the whole process. Highly inspired by the markdown/pandoc system, we developed Peccary, an R package and its dedicated Shiny Application with the objective to accelerate the use of previously released R packages through various translation and metaprogramming processes.

Methods: Peccary was developed with an agile method, progressively aggregating snippets of R code produced during real-life pharmacometrics works. Its first subpackage, PeccAnalysis, can be used to produce and evaluate various R code for population description (using table1 package), plot exploration (ggplot) and non-compartment analysis (pknca). The second subpackage, PeccaReverse, allows writing a structural model using a minimalist (simplified deSolve) syntax, before metaprogramming model simulations (using either deSolve or RxODE) and design evaluation (through PopED), along with performing various model syntax translations (e.g., into NONMEM, Monolix or nlmixr files). Finally, the third subpackage, PeccaResult, standardizes run outputs of several PMx software (NONMEM, Monolix, Adapt, nlmixr) to perform various diagnostic evaluation tasks.

Results: The metaprogramming system used in PeccAnalysis and PeccaReverse has many advantages. First, it simplifies the use of previously mentioned packages (by reducing the required knowledge and the time needed to program the output creation). Second, it creates links between independent tools (for instance using the same
inputs for several tasks). Third, the produced R code can be reviewed for eventual manual modification, verification (quality control) or traceable report integration.

**Conclusion:** Overall, Peccary was successful in improving PMx efficiency by providing a Shiny R platform that can produce various outputs during live meetings while keeping the possibility to extract Peccary-independent R source code for further in-depth control.

**Keywords**
Pharmacometrics, pharmacokinetics, pharmacodynamics, software, R, package

This article is included in the Bioinformatics gateway.

**Corresponding author:** Thibaud Derippe (thibaud.derippe@gmail.com)

**Author roles:** Derippe T: Conceptualization, Methodology, Software, Writing – Original Draft Preparation; Declèves X: Writing – Review & Editing; Fouliard S: Supervision, Writing – Review & Editing

**Competing interests:** Two co-authors, Thibaud Derippe and Sylvain Fouliard, are affiliated with Servier, a global pharmaceutical company. Thibaud Derippe has also received PhD grant funding from Servier.

**Grant information:** This work has been supported by the Servier company PhD program to Thibaud Derippe.

**Copyright:** © 2022 Derippe T et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

**How to cite this article:** Derippe T, Declèves X and Fouliard S. **Peccary, a metaprogramming open-source R platform to improve pharmacometrics efficiency** [version 1; peer review: 1 approved with reservations] F1000Research 2022, 11:951 https://doi.org/10.12688/f1000research.123904.1

**First published:** 18 Aug 2022, 11:951 https://doi.org/10.12688/f1000research.123904.1
Introduction

Pharmacometrics (PMx) is the discipline of the description and quantification of interactions between xenobiotics and patient’s biologic system (human and non-human) using mathematical models and taking into account both beneficial and adverse effects. These models are used in various circumstances, such as supporting R&D drug development in the pharmaceutical industry, especially through the implementation of Model-Informed Drug Discovery and Development (MID3), or in hospitals for therapeutic drug monitoring. In addition to the creation of the analysis plan and the Pharmacometrics report, the typical operational workflow of Pharmacometrics analyses includes: 1) pre-modeling work with data assembly, followed by graphical and statistical exploration, 2) iterative core modeling work with equation writing, parameter estimation, and model evaluation, and 3) final model exploitation, mostly through simulations, computation of secondary parameters, and/or parameter interpretation for the patients, in order to inform decision-making for a new study or individual patient dosing.

Pharmacometricians routinely use a variety of tools ranging from generic programming software (e.g., R) to specific PMx software (e.g., NONMEM, Monolix, Phoenix WinNonLin, in addition to multiple dedicated R packages). These PMx programs’ performance can be assessed both in terms of efficacy and efficiency. Efficacy is the ability of software to produce a desired or intended output, while efficiency represents the resources (time, effort) required to generate the intended output. The efficacy of most of the PMx software is satisfying, thanks to the variety of available programs and the flexibility that they allow. Efficiency, on the other hand, can be improved, through a simplification of the model-coding or the improvement of interoperability, since switching from one software to another with the same model is standard practice to complete a PMx project, but most of the software uses different syntax for coding the same elements (e.g., structural equation or model results). This interoperability issue has already been reported, both for the overall workflow or specific tasks such as optimal design.

Several projects have been created in various fields to improve or create interoperability between software, with at least two different paradigms showing successful results. In the first one, a new robust and generally marked-up language is created, aiming to become the standard of use. It has been created for system biology with the SBML project. The Drug Disease Model Resources (DDMoRe) project (https://www.ddmore.foundation) has implemented this methodology in pharmacometrics.

The second methodology consists of creating wrappers and translation tools from existing languages. The most successful and used tool of that kind is probably pandoc, dealing with document writing. It allows notably to automatically translate and produce a LaTeX-based PDF, HTML or Word document (high efficacy but low efficiency) by simply coding in markdown (low efficacy but high efficiency syntax). The addition of a light translating tool between the two turned the system both highly effective and efficient. Looking at the markdown / (LaTeX) / PDF workflow, at least four main advantages of this system can be described:

1. It is not needed to know LaTeX to produce a LaTeX-based PDF document (reduced required knowledge).
2. It is easier and faster to write first a markdown file and translate it rather than directly coding in LaTeX (efficiency improvement at a single software level).
3. There is no additional work needed to produce an HTML or Word document if those formats are finally required (multiple software / interoperability efficiency improvement).
4. If needed, it is possible to extract the intermediary LaTeX file to verify and eventually modify the output (as a result, even imperfect or incomplete pandoc task often remains useful).

We developed Peccary, a new set of tools (R console package with its Shiny App) that aims to provide the four previously mentioned pandoc benefits but in the pharmacometrics field. The goal is thus to gather and create links between pre-existing PMx software through various translating systems. R was judged the most appropriate platform for such a project because 1) it is the most used generic programming software in PMx, 2) almost every R packages are under free licenses, meaning their source code are available and users can freely modify or redistribute them, and 3) R packages already cover most of PMx tasks, such as a non-compartmental analysis (e.g., pkncar), model diagnostic (e.g., Xpose, npde or vpc), optimal design (e.g., PFIM and PopED), simulation (e.g., simuX), parameter estimation (e.g., nlmixr or saemix) and linker to proprietary software (e.g., popKinR to manage NONMEM runs or lixoft-connector to control Monolix). Having all these packages reunited in the same platform, such as Peccary, would therefore highly increases PMx efficiency.
Methods

Opération

Development process

Software can be developed through various methods. In 1999, Eric Raymond published an influential essay in which he compared software built as a cathedral to those built as a bazaar. Briefly, software built as a cathedral are very well thought out and structured before any code integration whereas bazaar systems imply much less preparation but need an organic development through immediate code creation and aggregation.

Peccary was developed with the bazaar methodology principles. Concretely, Peccary’s development followed the main author’s daily work as a pharmacometrician. Whenever a specific need to produce an output was encountered, an immediate solution was coded in R, most of the time (but not always) trying to build a generic function with a metaprogramming system. Peccary is the aggregation of these snippets of R-code. This agile methodology allowed us to develop a consistent R package piece by piece, ensuring that each implemented feature is truly needed and tested in real life. Since Peccary’s goal is to improve the efficiency of PMx, all attention was given to optimize the time and number of steps required to produce each output.

Software availability

The source code availability of Peccary and its subpackages is described in the software availability statement.

The downloading and installation of Peccary (and all subpackages) is made directly in R using the following code:

```
devtools::install_github("Peccary-PMX/Peccary")
```

Peccary and its Shiny App have been tested both on Windows, mac OS and Linux (Arch distribution).

Overview of the workflow

The current structure of Peccary, depicted in Figure 1, comprises three subpackages. Briefly, PeccAnalysis deals with pre-modeling work, PeccaReverse with model building and simulation, and PeccaResult with model diagnostics. Peccary is the main package that gathers these three subpackages under one platform usable both in R console or inside a Shiny App.

![Peccary structure with its three subpackages](image)

**Figure 1. Peccary structure with its three subpackages.** PeccAnalysis allows performing pre-modeling work such as population description, exploratory analysis and non-compartmental analysis (NCA). PeccaReverse allows to create or import a model with a minimal syntax, before using it for simulations, design evaluation, or translation into other syntaxes. Finally, PeccaResult standardizes and compares runs from multiple software with various analysis functions. Peccary can be used during a whole PMx process (red arrows) or just as a toolbox for any specific task (e.g., model translation from NONMEM to Monolix in green arrows).
Inside the Shiny App, users can create a Peccary’s project, allowing to save in the same place metadata of both the datasets (their path, separator, decimal, and “NA” strings) and various analysis (e.g., exploratory plots or model simulations). This system allows the analyst to reload any desired datasets when reopening the Shiny App easily, switch from one dataset to another with a single action, or reload and secondarily modify any pre-saved data analysis. Of note, Peccary requires the use of tidy data\textsuperscript{34}, secondarily modifiable in the Shiny App.

**Metaprogramming system**

Most Peccary functions have been metaprogrammed\textsuperscript{35}, meaning a Peccary independent R code/expression is produced before generating the outputs. This system was conceived to make Peccary useful both during the exploratory work phases (trying many things) and during the cleaning / reporting phase (selecting a few key analyses to report). Indeed, most of the produced works in a project are generally never published because they are just intermediate results in a trial-and-error workflow. During this exploratory phase, the Shiny app can directly produce the desired outputs, similarly to a direct markdown / PDF shortcut, with fast but unverified results that the user can only informally trust. However, every time a report integration needs to be made, the corresponding R code can be copy-pasted in an R/Rmd file before being further controlled (quality control) and eventually modified. Of note, this dual-system can dynamically co-exist, for Peccary is especially efficient working in a dual-screen set-up with two parallel R sessions - one using the Peccary Shiny app, the other a free R session for direct code transfer and manipulation.

**Implementation**

**PeccAnalysis**

PeccAnalysis gathers all functions made to analyze the data before any modeling work. This mostly includes three different activities, namely 1) population description, 2) plot exploration, and 3) non-compartmental analysis (NCA).

The theophyline dataset pre-existing in R is used for demonstrative purpose, with the addition of the following columns: dose level (DL) as the rounded real-dose, the presence or absence of a side effect (true or false), and the gender (men or women) leading to Table 1:

\[
\begin{align*}
\text{demoDF} & \leftarrow \text{as_tibble(Theoph)} \%>\% \\
& \quad \text{mutate(DL = round(Dose), Subject = as.double(as.character(Subject)))} \%>\% \\
& \quad \text{left_join(tibble(Subject = 1:12, side_effect = sample(c(T, F), 12, replace = T),)} \\
& \quad \quad \quad \text{gender = sample(c(“M”, “W”), 12, replace = T))} \\
& \quad \kable(head(demoDF))
\end{align*}
\]

| Subject | Wt  | Dose | Time | conc  | DL | side_effect | gender |
|---------|-----|------|------|-------|----|-------------|--------|
| 1       | 79.6| 4.02 | 0.00 | 0.74  | 4  | TRUE        | M      |
| 1       | 79.6| 4.02 | 0.25 | 2.84  | 4  | TRUE        | M      |
| 1       | 79.6| 4.02 | 0.57 | 6.57  | 4  | TRUE        | M      |
| 1       | 79.6| 4.02 | 1.12 | 10.50 | 4  | TRUE        | M      |
| 1       | 79.6| 4.02 | 2.02 | 9.66  | 4  | TRUE        | M      |
| 1       | 79.6| 4.02 | 3.82 | 8.58  | 4  | TRUE        | M      |

**Table 1. First six rows of the R data frame further used for presenting PeccAnalysis functionalities.** Wt stands for weight, conc for drug concentration and DL for dose level. Gender values are M for men and W for women.

**Population dataset description**

The population description generally requires two steps. The first step is to extract covariate information from the full dataset. Because PK/PD datasets generally contain multiple rows per ID (one for each observation and one for each drug administration), extracting only the covariate information as a reduced dataset is needed. This has been simplified with a function (\textit{pecc_search_cov()​}), automatically creating a covariate table by only inputting...
the column names representing the individuals (column “Subject” in our example). This function extracts only the
columns with a unique value for each individual, thus considered covariates. After this dataset covariate extraction,
descriptive statistics need to be computed and organized into a descriptive table, both operations already
performed by the table1 R package[36].

Overall, this whole process of population description has been embedded into a single PeccAnalysis function
(pecc_table1_original()). In the Shiny App, where it is not needed to know the function’s name, the metapro-
grammed R code is provided in addition to its corresponding table. The user can choose the output format in the
console: either the R code, the final table, or both (through outputExpr argument). An example of an automatic
covariate extraction and statistical description by providing only the source data frame, the name of the
column representing the individuals, and an optional covariate name to create the columns (other than “Overall”) is
provided leading to Table 2:

pecc_table1_original(df = demoDF, col = DL, reduceBy = Subject, outputExpr = "Both")

## table1::table1(~Wt + Dose + side_effect + gender | DL, data = demoDF %>%
##   distinct(Subject, Wt, Dose, DL, side_effect, gender))

Table 2. Result of the R code (chunk) displayed above: statistical description of extracted
covariates ("Wt", "Dose", "DL", "side_effect" and "gender") for each dose level ("DL"). Wt stands for
weight, M for men and W for women. The corresponding metaprogrammed code was printed before its
evaluation.

| metric       | 3     | 4     | 5     | 6     | Overall |
|--------------|-------|-------|-------|-------|---------|
| Wt (N=1)     | 86.4  | 76.2  | 66.2  | 56.4  | 69.6    |
| Mean (SD)    | NA    | 4.19  | 4.29  | 2.55  | 9.50    |
| Median [Min, Max] | 86.4 [86.4, 86.4] | 76.2 [72.4, 80.0] | 65.0 [60.5, 70.5] | 56.4 [54.6, 58.2] | 70.5 [54.6, 86.4] |
| Dose (N=4)   | 3.10  | 4.21  | 4.85  | 5.68  | 4.63    |
| Mean (SD)    | NA    | 0.225 | 0.325 | 0.255 | 0.747   |
| Median [Min, Max] | 3.10 [3.10, 3.10] | 4.21 [4.00, 4.40] | 4.92 [4.53, 5.30] | 5.68 [5.50, 5.86] | 4.53 [3.10, 5.86] |
| side_effect  |       |       |       |       |         |
| Yes (N=5)    | 0 (0%)| 2.00  | 4.00  | 2.00  | 8.00    |
| No (N=2)     | 1.00  | 2.00  | 1.00  | 0 (0%)| 4.00    |
| gender       |       |       |       |       |         |
| M (N=2)      | 1.00  | 3.00  | 4.00  | 1.00  | 9.00    |
| W (N=12)     | 0 (0%)| 1.00  | 1.00  | 1.00  | 3.00    |

Of note, the “...” argument (non-used above) allows the user to manually choose the desired covariates to
display (instead of printing them all) but also to transform the numeric covariates into factors, even though the
main interest of this function relies on its ability to detect the covariates automatically.

Specific case of Boolean covariate
If a covariate is a Boolean (only possible values are TRUE/FALSE or 0/1), a specific function counts how
many patients have this covariate equal to TRUE (or 1) in regards to one or two other covariates. For instance,
this is used to analyze the number of good responder patients to treatment, or, as in the following example, for
counting the number of patients with side effects in regard to their genders and dose levels leading to Table 3:

pecc_search_cov(demoDF, Subject) %>% # the function to extract cov
dpecc_count(metric = side_effect, col_cov = gender, row_cov = DL) %>%
kable()
Of note, this function is one of the few from PeccAnalysis that is not yet metaprogrammed. The first attempts led to a too cumbersome R code compared to the easiness of manually checking the results.

**Graphical exploration of longitudinal data**

Another PeccAnalysis function allows visualization of longitudinal data via appropriate ggplot objects. The function includes most of the classical ggplot customization (coloring observations through a covariate, performing subplot via wrap functions). Additional features have been added, such as a direct output standardization by a continuous column, or an improved wrapping system where other profiles are printed in grey background (improving the comparisons between groups). The following code displays both the metaprogrammed code and its results (Figure 2):

```r
plot_spagh(demoDF, Time, conc, group = Subject, facetwrap = DL, col = gender, bkgrdalpha = 0.6, output = "Both", ylabs = "Concentration (units)")
```

An example of using this function inside the Shiny App is presented in Figure 3, where both the plot, the peccary dependent code (the `plot_spagh(...)` code as shown above) and the peccary independent code (metaprogrammed R expression) are displayed simultaneously.

**Table 3. Results of the pecc_count() function: counts of patients with side effects according to patient's dose levels "DL" (rows) and gender (columns).** Gender values are M for men and W for women.

| DL/gender (side_effect) | M   | W   | Total |
|-------------------------|-----|-----|-------|
| 3                       | 0/1 | -   | 0/1   |
| 4                       | 1/3 | 1/1 | 2/4   |
| 5                       | 3/4 | 1/1 | 4/5   |
| 6                       | 1/1 | 1/1 | 2/2   |
| Total (0 NA)            | 5/9 | 3/3 | 8/12  |

**Figure 2. Example of a longitudinal ggplot produced by PeccAnalysis.** The corresponding R expression was printed before its evaluation.
Non-compartmental analysis (NCA)

Peccary simplifies the use of the pknca package for performing NCA in several ways. As a first step, the function automatically creates the dose dataset, either using an EVID column, or assuming dose administration at time 0 (depending on the inputs). Second, it simplifies the computation of AUC from time 0 to X (X being a numeric argument) by using the pknca “intervals” system. Integrating “intervals” also makes it easier to toggle the computation of more NCA parameters by modifying the metaprogrammed code (for instance setting vss. last to True in the following example). Third, it automatically transforms the final pknca results into an organized data frame, directly including all possible covariates (extracted with pecc_search_cov()) for further analysis. Results are shown in Table 4.

demoNCA <- peccary_pknca(demoDF, Time, conc, Subject = Subject, dose = Dose, AUC0_x = 24, computeMedian = F, outputExpr = "both")

## NCA_eval <- {
##     conc_obj <- PKNCAconc(data = demoDF, formula = conc ~ Time | Subject)
##     dose_obj <- PKNCAdose(data = demoDF %>% group_by(Subject) %>% slice(1) %>% mutate(`:=`(Time, 0), `:=`(conc, 0)), formula = Dose ~ Time | Subject)
##     intervals_manual <- data.frame(start = 0, end = c(24, Inf), cmax = c(F, T), tmax = c(F, T), auclast = c(T, T), aucinf.obs = c(F, T), cmin = c(F, T), tlast = c(F, T), vss.last = c(F, T), cl.obs = c(F, F))
##     data_obj_automatic <- PKNCAdata(conc_obj, dose_obj, intervals = intervals_manual)
##     pk.nca(data_obj_automatic)
## }
## left_join(NCA_eval$result %>% filter(end != 24) %>% select(Subject, PPTESTCD, PPORRES, end) %>% distinct() %>% spread(key = PPTESTCD, value = PPRES), NCA_eval$result %>% filter(end == 24) %>% select(Subject, PPORRES) %>% rename(\'AUC24\' = PPORRES))) %>%
## left_join(demoDF %>% distinct(Subject, Wt, Dose, DL, side_effect, gender))

demoNCA[1:5, c("Subject","gender","cmax", "auclast", "AUC24", "tmax")]

Figure 3. Two screenshots reattached (after mouse scrolling) of the Shiny app interface for creating longitudinal plots. The inputs, produced plot, and metaprogrammed code (both Peccary dependent and Peccary independent) are displayed on the same page. In addition, each dataset can store the metadata of each analysis, such as a plot that can be recreated when reopening the Shiny app, before eventually being re-modified. On the left panel, the user can efficiently switch from one dataset to another (among the ones stored in a project or inputted since the beginning of the session).
Of note, a very similar function (`peccary_pknca()`) metaprograms the NCA computation from scratch, manually computing all classical NCA parameters that do not require extrapolation (Cmax, Tmax, AUCTlast, ...).

```r
# demoNCA <- peccary_NCA(demoDF, timecol = Time, obscol = conc, Subject, auc_0_X = 24)
```

In both cases, a table with NCA individual parameters is computed. In the Shiny app, a table1 function summarizes the results by possible covariates, along with the generation of either a boxplot or a scatter plot depending on the specific covariate the user wants to analyze. Those plots are generated using the GGpubr package, directly integrating statistical tests. The following code generates boxplots (Figure 4) using NCA results:

```r
plot_boxplot(df = demoNCA, x = gender, auclast, cmax, tmax, AUC24,
             outputExpr = "both", methodCompar = "wilcox.test")
```

Table 4. First five rows and a sample of columns produced by the pknca package, with the corresponding R expression.

| Subject | gender | cmax  | auclast  | AUC24   | tmax |
|---------|--------|-------|----------|---------|------|
| 1       | M      | 10.50 | 147.23475| 92.36544| 1.12 |
| 2       | M      | 8.33  | 88.73128 | 67.23456| 1.92 |
| 3       | M      | 8.20  | 95.87820 | 70.58886| 1.02 |
| 4       | M      | 8.60  | 102.63362| 72.84350| 1.07 |
| 5       | W      | 11.40 | 118.17935| 84.39951| 1.00 |

Figure 4. Boxplot comparing non-compartmental analysis (NCA) results in regards to patients’ gender using the Wilcoxon test (other tests can be selected with “methodCompar” argument). Of note, plot_boxplot() is a generating boxplot function that can be used in other circumstances. All auclast, cmax, tmax, and AUC24 belong to the “...” arguments, allowing to add more parameters in the `facet_wrap()` code.
The following code produces a scatter plot (Figure 5) to visualize the relationship between a continuous covariate and NCA parameters:

```r
plot_correlation(df = demoNCA, x = Wt, auclast, cmax, tmax, AUC24, outputExpr = "both", cor.method = "spearman")
```

```
## demoNCA %>% gather(auclast, cmax, tmax, AUC24, key = "key", value = "value") %>%
##     ggscatter(x = "Wt", y = "value", add = "reg.line", cor.coef = TRUE,
##         cor.method = "spearman", conf.int = FALSE) + theme_bw() +
##     facet_wrap(~key, scales = "free") + scale_y_log10() + scale_x_log10() +
##     labs(x = "Wt")
```

**Figure 5.** Scatter plots comparing non-compartmental analysis (NCA) results in regards to patients weight using the Spearman test (other tests can be selected with "cor.method" argument). Of note, plot_correlation() is a generating scatter plot function that can be used in other circumstances. All auclast, cmax, tmax, and AUC24 belong to the "..." arguments, allowing to add more parameters in the facet_wrap() code.

The Shiny app gathers all metaprogrammed R codes in a single bloc (NCA generation, table1 generation, and covariate analyses).

**PeccaReverse**

PeccaReverse is the second Peccary subpackage whose role is to simplify model creation, dynamic exploration, simulations, and translation into other software syntaxes. A simplified deSolve syntax has been used as the core language of PeccaReverse. For example, to create two compartments for a PK model with oral drug absorption, one can simply write the structural model as follows:

```r
model_demo <-

"dGut <- - ka * Gut
dCentral <- ka * Gut + Periph * k21 - Central * k12 - ke * Central
dPeriph <- Central * k12 - Periph * k21
cconc_output <- Central/vd"
```

PeccaReverse also contains a PK/PD library to avoid writing frequently used models from scratch. Then, the `deSolve_pecc()` function automatically extracts all the model compartments, parameters, and initial states (if provided) from the equations, and also reconstitutes both deSolve and RxODE expressions:

```
deSolve_pecc(model_demo)
```
## $model
## function(t, state, parameters) {
##   with(as.list(c(state, parameters)), {
##     dGut <- -ka * Gut
##     dCentral <- ka * Gut + Periph * k21 - Central * k12 -
##                 ke * Central
##     dPeriph <- Central * k12 - Periph * k21
##     conc <- Central/vd
##     list(c(dGut, dCentral, dPeriph), c(conc = conc))
##   })
## }
##
## $modelrxode
## RxODE({
##   d/dt(Gut) <- -ka * Gut
##   d/dt(Central) <- ka * Gut + Periph * k21 - Central * k12 -
##                     ke * Central
##   d/dt(Periph) <- Central * k12 - Periph * k21
##   conc <- Central/vd
## })
##
## $state
## [1] "Gut"     "Central" "Periph"
##
## $parameter
## [1] "ka"  "k21" "k12" "ke"  "vd"
##
## $output_manual
## [1] "conc"
##
## $toplot
## character(0)
##
## $initialCond
## # A tibble: 0 x 2
## # ... with 2 variables: Cmt <chr>, value <chr>

Briefly, deSolve_pecc() function works by analyzing the text through regular expression manipulations (grep, grepl, gsub, . . . ). Everything not recognized as a compartment (here, Gut, Central and Periph, recognized with the initial “d” and some other conditions to distinguish them from variables starting with “d”), a computed variable (every word before “<-”, here conc), or a reserved word (such as log, exp, if, else . . . ) is directly considered as a parameter (ka, k12, k21, ke and Vd). Adding "_output" or "_plot" (automatically removed in the produced deSolve/RxODE code) at the end of a variable means this variable is respectively an output of the model or that it needs to be reported after simulations for plotting.

Within the Shiny app, once the structural model has been analyzed (through an action button), various input tables are automatically created and pre-filled, as depicted in Figure 6. Filling the missing parts of these tables (further described) allows the users to perform most of PeccaReverse tasks, namely 1) simulations, 2) design evaluation, and 3) various syntax translations. The following sections describe the main console functions internally invoked while using the Shiny App. It is important to note that this system is much less efficient in console mode because it requires writing input tables from scratch, plus the need to know how to use the internal functions (compared to a more intuitive and dynamic use in the Shiny app).

Simulations
Performing simulations and plots was divided into three steps: 1) programming the parameters sets, 2) programming the simulations, and 3) programming the plots. For each step, several methods exist, leading to multiple possible combinations. The metaprogrammed R code is incremented after each of the three steps to produce the final R code.

The first step consists of programming the sets of parameter values to simulate. It requires at least one table containing the population values, the distributions (normal, log-normal) and the information on whether each parameter is estimated or fixed. Importantly, several values can be attributed to a parameter to perform a local
sensitivity analysis automatically. The user can choose between simulating those exact values or sampled values within a distribution. A variance/covariance matrix (coded in standard deviation or variance) should be inputted in the late case. Finally, a function generates the code for sampling parameters sets, in the following example asking for 100 parameters sets for three possible k21 values:

Figure 6. Three screenshots reattached (after mouse scrolling) of the Shiny app interface for PeccaReverse. The model equations are first written (or imported from the model library) in the top left bloc. After clicking on an action button below the equations, the model is analyzed, and various input tables are created and pre-filled. After completing these tables (e.g., attributing a value for each parameter value), the user can perform simulations in the corresponding module, here displaying a tumor growth inhibition model with three different dose levels. On each panel, the corresponding observations were added. The whole Peccary independent code for performing this plot can be seen below the plot (by unrolling the collapsed box). All other PeccaReverse modules (design evaluation, syntax translation, . . . ) are in the other collapsed boxes below on the same page.
```r
# Parameter value table
param_df <- tribble(~Param, ~Value, ~ Distrib, ~E,
  "ka", 0.3, "logN", "Esti",
  "ke", 0.1, "logN", "Esti",
  "k12", 0.3, "logN", "Esti",
  "k21", c(0, 0.1, 0.3), "logN", "Esti",
  "vd", 5, "Norm", "Esti")

# Variance/covariance matrix (here coded in sd)
matrix_sd <- tribble(~ ka,~ k21,~ k12,~ ke,~ vd,
  0.3,NA,NA,NA,  # ka
  0,0.3,NA,NA,  # k21
  0,0,0.3,NA,  # k12
  0,0,0,0.3,  # ke
  0,0,0,0,  # vd
)

# Final function for parameter sampling
paramInd <- random_etas(n = 100, param_df, matrix_sd, sd = T, returnExpr = T)
```

```
## {  
##     parameters <- tibble(ka = c(0.3, 0.3, 0.3), ke = c(0.1, 0.1,
##     0, 0.1, 0.3, 0.1), k12 = c(0, 0, 0, 0, 0, 0), k21 = c(0, 0, 0,
##     0, 0, 0), vd = c(5, 5, 5), nameparset = c("k21 = 0", "k21 = 0.1",
##     "k21 = 0.3")  
##   )  
##   matrix_eta <- tibble(k12 = c(0.09, 0, 0, 0, 0), k21 = c(0,
##   0.09, 0, 0, 0), ka = c(0, 0, 0.09, 0, 0), ke = c(0, 0, 0,
##   0.09, 0), vd = c(0, 0, 0, 0, 0.09)) %>% as.matrix  
##   eta <- matrix(rnorm(100 * nrow(matrix_eta)), ncol = nrow(matrix_eta)) %*%
##     chol(matrix_eta)  
##   eta <- apply(eta, 2, function(x) rep(x, nrow(parameters)))  
##   ind_param <- map_dfc(parameters, ~rep(.x, 100)) %>% arrange(nameparset)  
##   parameterslogN <- c("ka", "ke", "k12", "k21")  
##   ind_param[parameterslogN] <- ind_param[parameterslogN] *
##     exp(eta[, parameterslogN])  
##   parametersNorm <- "vd"  
##   ind_param[parametersNorm] <- ind_param[parametersNorm] +
##     eta[, parametersNorm]  
## }
```

Of note, no distribution sampling would have been performed if n (number of simulations) was equal to 0: the produced code would have stopped after the “parameters <- [... ]” line.

The second step consists of programming the simulations using the sets of previously coded parameter values. For the simulations, two additional tables are needed. First, a table containing the initial states of compartments is required. Initial states can be inputted directly as numeric values or formulas involving parameters (e.g., “kin/kout”). Initial states mentioned directly in the structural equations are automatically included in the pre-filled table. Second, a table must contain all information regarding one or several protocols of administrations (times of administrations, amount, inputted compartment, . . . ). Each protocol can contain one or several administrations. Of note, the time column can include any R code (text) which produces an atomic vector after evaluation (e.g., “0:10”, “seq(0,240,12)”, . . . ), allowing to create several administrations in a single line. Tlag, bioavailability and perfusions information are also set up in this table (in the Shiny app, new parameters corresponding to Tlag or bioavailability are automatically added to the parameters values table).

The simulations can then be performed using the deSolve package or the faster and more PMx-oriented RxODE package. Every possible combination of parameter values and protocols is simulated.

```r
# Initial conditions for compartments
states <- tibble(Cmt= c("Gut", "Central", "Periph"), t0 = 0)

# Drug protocol definition (extension of deSolve events)
events <- tribble(~Proto, ~var, ~time, ~value, ~method, ~ADM, ~tlag, ~F, ~Perf, ~use,
  "1", "Gut", "0", 4.5 , "add","1", F, F, "none", T,
  "2", "Gut", "seq(0,100,24)", 1 , "add","1", F, F, "none", T
)
```
# Function to add the RxODE expressions to paramInd
simult <- make_simulations_rxode(parameter = paramInd, model = model_demo, states = states, events = events, times = c(seq(0, 100, 1)), Progress = F, returnExpr = T)

# Here we Print only the new expressions, as the
# first nine ones correspond to paramInd
expr({!!! simult[10:length(simult)]})

##
##     parameters_df <- ind_param %>% crossing(Proto = c("Prot 1","Prot 2")) %>% rownames_to_column("id") %>% mutate(id = as.double(id))
##     times <- c(seq(0, 100, 1))
##     events_df <- tibble(Proto = c("1", "2"), cmt = c("Gut", "Gut"), time = c("0", "seq(0,100,24)"), amt = c(4.5, 1), method = c("add", "add"), ADM = c("1", "1"), Perf = c("none", "none"), evid = c(1, 1)) %>% mutate(Proto = paste0("Prot ", Proto)) %>% mutate(time = map(time, ~eval(parse_expr(.x)))) %>% bind_rows(crossing(time = c(seq(0, 100, 1)), evid = 0, cmt = "Gut", Proto = c("Prot 1", "Prot 2"))) %>% group_by(Proto) %>% nest() %>% full_join(parameters_df %>%
##     select(id, Proto) %>% mutate(Proto = as.character(Proto))) %>% %>% unnest() %>% mutate(amt = as.double(amt), time = as.double(time)) %>%
##     arrange(id, time) %>% mutate(amt = as.double(amt), time = as.double(time)) %>%
##     arrange(id, time)
##     model <- RxODE({
##         d/dt(Gut) <- -ka * Gut
##         d/dt(Central) <- ka * Gut + Periph * k21 - Central * k12 - ke * Central
##         d/dt(Periph) <- Central * k12 - Periph * k21
##         conc <- Central/vd
##     })
##     simulations <- as_tibble(model$solve(parameters_df %>% select(-nameparset, -Proto), events_df)) %>% mutate(securejoin = 1) %>% left_join(parameters_df %>% mutate(securejoin = 1))
##     model <- RxODE({
##         d/dt(Gut) <- -ka * Gut
##         d/dt(Central) <- ka * Gut + Periph * k21 - Central * k12 - ke * Central
##         d/dt(Periph) <- Central * k12 - Periph * k21
##         conc <- Central/vd
##     })
##     simulations <- as_tibble(model$solve(parameters_df %>% select(-nameparset, -Proto), events_df)) %>% mutate(securejoin = 1) %>% left_join(parameters_df %>% mutate(securejoin = 1))
##     model <- RxODE({
##         d/dt(Gut) <- -ka * Gut
##         d/dt(Central) <- ka * Gut + Periph * k21 - Central * k12 - ke * Central
##         d/dt(Periph) <- Central * k12 - Periph * k21
##         conc <- Central/vd
##     })
##     simulations <- as_tibble(model$solve(parameters_df %>% select(-nameparset, -Proto), events_df)) %>% mutate(securejoin = 1) %>% left_join(parameters_df %>% mutate(securejoin = 1))
## }
##
## simulations <- as_tibble(model$solve(parameters_df %>% select(-nameparset, -Proto), events_df)) %>% mutate(securejoin = 1) %>% left_join(parameters_df %>% mutate(securejoin = 1))
##

Not involved in the above example, tlag and bioavailability are applied directly by modifying the “events_df” table, respectively by an increase of the time of administration and by multiplying the amount of injected drug by the individual bioavailability. Perusions are natively handled by RxODE package, while it was hard-coded (with an on/off time dependent zero-order input) when producing a deSolve code.

Regardless of the simulation software used (deSolve, RxODE), the simulated values can be plotted with two additional pieces of information: 1) a table containing which outputs the user wants to display and if observations points should be added (with various filtering systems); 2) a table containing for each plot (several can be produced simultaneously) additional information such as the scales to use (logarithmic or continuous) or the wrapping systems. It is possible to wrap the plot either by output, protocol, set of parameter values, a mix of them or none. In the following example, we wrapped the plot both by the set of parameter values and by protocol, every other characteristic (here the outputs) being dissociated using different colors (Figure 7).

# What to plot (output), and should we add observation points (not used here)
plot_table <- tribble(~Plot, ~Todisplay, ~Point, ~Filter_of_dataset, ~YTYPE, ~Check, 1, "Periph", F, "", NA, T, 1, "conc", F, "", NA, T)

# Y logarithmic scale and grid "Parameters vs Events" ("P|E")
plot_table_cov <- tribble(Plot = 1, wrap = "P|E", ylog = T, xlog = F)

# Function to add the plotting expressions to simult
plotcode <- plot_simulations(simult, plot_table = plot_table, plot_table_cov = plot_table_cov, returnExpr = T)

# Here also we print only the new expressions
expr({!!! plotcode[15:20]})
##
for (a in names(simulations)[!names(simulations) %in% c("id", "nameparset", "time", "Proto")]) {
    simulations[[a]] <- if_else(simulations[[a]] < 1e-04, 0, simulations[[a]])
}

quantiles <- simulations %>% mutate(nameparset = fct_inorder(nameparset)) %>%
    gather(-id, -nameparset, -Proto, -time, key = "key",
    value = "value") %>% group_by(time, key, nameparset, Proto) %>%
    nest() %>% crossing(tibble(q1 = seq(0.1, 0.8, 0.1), q2 = q1 + 0.1)) %>%
    mutate(qx = map2_dbl(q1, data, function(q1, data) {
        unname(quantile(data$value, q1, na.rm = T))
    })), qx2 = map2_dbl(q2, data, function(q2, data) {
        unname(quantile(data$value, q2, na.rm = T))
    })) %>% select(-data)

breaks2 <- map(-40:40, ~1:9 * 10^.x) %>% reduce(c)
labels2 <- as.character(breaks2)
labels2[-seq(1, length(labels2), 9)] <- ""

quantiles %>% mutate(color = key) %>% filter(key %in% c("Periph", "conc")) %>%
    mutate(forfct = as.double(nameparset)) %>%
    ggplot + geom_ribbon(aes(x = time, ymin = qx, ymax = qx2,
    alpha = paste0(q1, "-", q2), fill = fct_reorder(color,
    forfct))) + geom_line(data = quantiles %>% mutate(color = key) %>%
    filter(key %in% c("Periph", "conc"), q1 == 0.5), aes(time,
    qx, col = fct_reorder(color, as.double(nameparset)))) +
    labs(x = "Time", y = "", alpha = "quantiles", fill = "",
    col = "") + scale_alpha_manual(values = c(0.2, 0.4, 0.6, 0.8, 0.8, 0.6, 0.4, 0.2)) +
    theme_bw() + facet_grid(fct_reorder(nameparset, as.double(gsub(".+=" "", "", nameparset))) ~ Proto) +
    scale_y_log10()

# But we evaluate everything at the same time
eval(expr(list(!!plotcode)))

---

**Figure 7.** Final simulations of both “conc” (red) and “Periph” (blue) variables, simulated simultaneously with two different protocols (columns) and three different values of k21 parameter (rows). This plot corresponds to the evaluation of the final R expression, gathering both the parameter sampling, RxODE simulations, quantiles generation, and final ggplot pieces of codes.
Overall, this system is highly flexible with many different possible uses, such as exploring the dynamics of a model by looking at several variables simultaneously, exploring the impact of modifying parameter values, comparing different dose regimens. Because it is possible to add observation plots (with various systems to control in which panels some subsets of data are displayed), different model structures or initial values can be tried for “manually” capturing the data before moving to a parameter estimation software. A recent feature allows having a first parameter estimation based on naive pool methods, using the R Optim function to optimize square residual sums.

Finally, Peccary can import and translate NONMEM, Monolix, or ADAPT files into simplified deSolve syntax. If the user provides a path to an estimated Monolix or NONMEM run, Peccary can extract the observation values, the estimated parameter values and the study designs for each individual (along with estimated typical parameters). Only the simplest study designs can be extracted for now. More complex input (e.g., NONMEM’s “SS” or “II” columns) are not covered yet and would require additional programming work. If the extraction is successful, all Peccary tables are automatically filled, with the possibility to switch between the population or any individual sets of values at any time. The main author particularly used this for first recreating the same individual profile as estimated by the parameter estimation software, before trying various modifications to explore what could improve the model performances (new structural equations and/or parameter values).

**Design evaluation using PopED**

PeccarReverse can also produce and evaluate a PopED code to perform design evaluation. The only additional required information is a table containing the desired sampling schedule per group (Figure 8). Several groups can be added and specific covariate values can be assigned to each. In the following example, two groups are created, respectively receiving protocols number one and two. The first group has two observation types (Gut and conc) while the second group has only one (conc). Two rows corresponding to the same group cannot have different protocols or numbers of ids, such as Non Available data (NA) are produced automatically in case of discrepancies (only the first values are kept). Similarly, each output can have a single set of error values (additive and/or proportional). A “F” after a number (in this table or in the variance/covariance matrix) stipulates that the corresponding value is fixed. The information of estimated or fixed typical parameter values is included in the parameter value table (column “E” of param_df).

```r
# To keep only one k21 values (vs three previously)
param_df$Value <- c(0.3, 0.1, 0.3, 0.1, 5)

OD_input <- tribble(~Output, ~Group, ~Proto, ~ TimeSample, ~ add, ~ prop, ~nIdgroup, ~cov,  
                   "conc", 1, 1, "c(0.2,1,5,10)", "0F", "0.2", 30, "",  
                   "Gut", 1, NA, "c(0.2,1,5,10)", "0F", "0.1", NA, "",  
                   "conc", 2, 2, "seq(0,48,12)", NA, NA, 15, "")

resPopEd <- pecc_PopEd(model = model_demo, OD_input = OD_input, states = states,  
                        events = events, parameters = param_df,  
                        diagOmega = matrix_sd, outputExpr = "Both")

## library(PopED)  
## modelpopEd <- function(Time, State, Pars) {  
## with(as.list(c(State, Pars)), {  
##   dGut <- -ka * Gut  
##   dCentral <- ka * Gut + Periph * k21 - Central * k12 -  
##   ke * Central  
##   dPeriph <- Central * k12 - Periph * k21  
##   conc <- Central/vd  
##   list(c(dGut, dCentral, dPeriph), c(conc = conc))  
## })  
## }  
## events_input <- data.frame(var = c("Gut", "Gut", "Gut",  
##                             "Gut", "Gut"), time = c(0, 0, 24, 48, 72, 96), value = c(4.5,  
##                             1, 1, 1, 1, 1), method = c("add", "add", "add", "add",  
##                             "add", "add"), Proto = c(1L, 2L, 2L, 2L, 2L, 2L), Group = c(1L,  
##                             2L, 2L, 2L, 2L, 2L))
```
model2 <- function(model_switch, xt, parameters, poped.db) {
  with(as.list(parameters), {
    A_ini <- c(Gut = 0, Central = 0, Periph = 0)
    times_xt <- drop(xt)
    eventdat <- events_input %>% filter(Group == group)
    times <- sort(c(times_xt, eventdat$time))
    out <- ode(A_ini, times, modelpopEd, parameters,
               events = list(data = eventdat))
    out <- out[match(times_xt, out[, "time"]), ]
    y <- xt * 0
    y[model_switch == 1] <- out[, "conc"][model_switch == 1]
    y[model_switch == 2] <- out[, "Gut"][model_switch == 2]
    y = cbind(y)
    return(list(y = y, poped.db = poped.db))
  })
}

fg <- function(x, a, bpop, b, bocc) {
  parameters = c(ka = bpop[1] * exp(b[1]), ke = bpop[2] *
                 exp(b[4]), k12 = bpop[3] * exp(b[3]), k21 = bpop[4] *
                 exp(b[2]), vd = bpop[5] + b[5], group = a[1])
  return(parameters)
}

feps_ODE_compiled <- function(model_switch, xt, parameters, epsi, poped.db) {
  MS <- model_switch
  y <- model2(model_switch, xt, parameters, poped.db)[[1]]
  Output1 <- y * (1 + epsi[, 1])
  Output2 <- y * (1 + epsi[, 2])
  y[MS == 1] <- Output1[MS == 1]
  y[MS == 2] <- Output2[MS == 2]
  return(list(y = y, poped.db = poped.db))
}

poped.db <- create.poped.database(ff_fun = "model2", fError_fun = "feps_ODE_compiled",
                                  fg_fun = "fg", groupsize = c(30, 15), m = 2L, sigma = c(conc_prop = 0.2,
                                  Get_prop = 0.1), notfixed_sigma = c(conc_prop = 1,
                                  Get_prop = 1), bpop = c(ka = 0.3, ke = 0.1, k12 = 0.3,
                                  k21 = 0.1, vd = 5), notfixed_bpop = c(1, 1, 1, 1, 1,
                                  1), d = c(ka = 0.3, k21 = 0.3, k12 = 0.3, ke = 0.3,
                                  vd = 0.3), notfixed_d = c(ka = 1, k21 = 1, k12 = 1,
                                  ke = 1, vd = 1), xt = list(c(c(0.2, 1, 5, 10), c(0.2,
                                   1, 5, 10)), seq(0, 48, 12)), model_switch = list(c(1,
                                  1, 1, 1, 2, 2, 2, 2), c(1, 1, 1, 1, 1)), a = list(group = 1,
                                  group = 2))
  popedResult <- list()
  popedResult$plot_0D <- plot_model_prediction(poped.db, model_num_points = 500,
                                           facet_scales = "free")
  popedResult$result_0D <- evaluate_design(poped.db)
  names(popedResult$result_0D$rse) <- c("ka_pop", "ke_pop",
            "k12_pop", "k21_pop", "vd_pop", "ka_omega", "ke_omega",
            "k12_omega", "k21_omega", "vd_omega", "conc_prop", "Gut_prop")
  popedResult
}

# Print the predicted Relative Standard Errors (RSE)
resPopEd$result_0D$rse

# Print the study design plot
resPopEd$plot_0D
Translation into other syntaxes

The model can also be translated into various other syntaxes such as Monolix, NONMEM, ADAPT, and nlmixr. All the corresponding translation functions are based on regular expression manipulation (mostly using `gsub` function) and take into consideration the information included in the previous tables (e.g., parameter distribution, initial parameter values, . . . ). A final table can also mention the outputs (with associated YTYPE and error model). For example, the following code translates our demonstrative model into a Monolix model file.

```r
mb_output <- tibble(
  output = "conc",
  YTYPE = 1,
  err_add = 0.1,
  err_prop = 0.3,
  export = TRUE,
  rm = FALSE)

ode_monolix(model_demo, parms = param_df, y = states, mb_output = mb_output, depot = events)
```

\[
\begin{align*}
\text{## [LONGITUDINAL]} \\
\text{## input =\{ka, ke, k12, k21, vd\}} \\
\text{## PK:} \\
\text{## depot(target = Gut, adm = 1)} \\
\text{## EQUATION:} \\
\text{## ; Initial conditions} \\
\text{## t0 = 0} \\
\text{## Gut_0 = 0} \\
\text{## Central_0 = 0} \\
\text{## Periph_0 = 0} \\
\text{## ; Dynamical model} \\
\text{## ddt_Central = ka * Gut + Periph * k21 - Central * k12 - ke * Central} \\
\text{## ddt_Periph = Central * k12 - Periph * k21} \\
\text{## conc = Central / vd} \\
\text{##} \\
\text{## OUTPUT:} \\
\text{## output = \{conc\}}
\end{align*}
\]

A function for creating the corresponding mlxtran file (to use directly through R with the Lixoft connector\cite{footnote}) exists and works for the simplest models (e.g., one or several YTYPE with normal or log-normal variabilities but without covariate), but was in practice judged not efficient enough compared to simply copy/pasting the structural model code directly in the Monolix suite, and thus not further developed.

For NONMEM, minor manual modifications would be required after the translation, for instance if the user wants to change the used ADVAN or the estimation method. Parameters can be coded with or without MU referencing, and a block for handling data below the levels of quantification can be automatically added.
ode_nonmem(model demo, omega = matrix_sd, parms = param_df, y = states, mb_output = mb_output, 
mu_referencig = F)

## $PROB name of the model and/or description
##
## $INPUT
##
## $DATA your_file.data IGNORE = I
##
## $SUBROUTINE ADVAN13 TOL9
##
## $MODEL
##
## $PK
##
## ka = THETA(1) * exp(ETA(1))
## ke = THETA(2) * exp(ETA(2))
## k12 = THETA(3) * exp(ETA(3))
## k21 = THETA(4) * exp(ETA(4))
## vd = THETA(5) + ETA(5)
##
## A_0(1) = 0 ; Gut initialisation
## A_0(2) = 0 ; Central initialisation
## A_0(3) = 0 ; Periph initialisation
##
## $DES
##
## DADT(1)  = -ka * A(1)
## DADT(2)  = ka * A(1) + A(3) * k21 - A(2) * k12 - ke * A(2)
## DADT(3)  = A(2) * k12 - A(3) * k21
## conc = A(2) / vd
##
## $ERROR
##
## ADD = THETA(6)
## PROP = THETA(7)
## OUTPUT =conc
## IPRED = OUTPUT
## W = (OUTPUT * OUTPUT * PROP**2 + ADD**2)**0.5 + 0.0001
## IRES = DV - IPRED
## IWRES = (DV - IPRED)/W
##
## $THETA
##
## (0,0.3) ; ka - theta 1
## (0,0.1) ; ke - theta 2
## (0,0.3) ; k12 - theta 3
## (0,0.1) ; k21 - theta 4
## (0,0.5) ; vd - theta 5
## (0,0.1) ; err_add - theta 6
## (0,0.3) ; err_prop - theta 7
##
## $OMEGA
## 0.3 ; ka - eta 1
## 0.3 ; ke - eta 2
## 0.3 ; k12 - eta 3
## 0.3 ; k21 - eta 4
## 0.3 ; vd - eta 5
##
##
Various software tools allow the estimation of population parameters, whether they are proprietary (NONMEM, Monolix, ADAPT) or open-source (Saemix, nlmixr), producing similar outputs (parameter estimates, objective function, simulated values, . . . ), but often stored with different architecture. The goal of PeccaResult is thus to provide a tool to easily compare various runs together independently of the parameter estimation software by first standardizing the output of each software in a uniformized structure before manipulating those structures with new model diagnostic functions. However, metaprogramming such uniformization of software is very challenging and would probably result in a final code far too complex compared to the direct use of other specific software (e.g., Monolix Suite or Xpose for NONMEM runs’). Therefore, PeccaResult was designed to be used only during the modeling process (the trial-and-error phase), and not for being integrated into a final report: its integration into a traceable workflow is currently not available.

The main idea of PeccAnalysis is thus to use various functions to standardize the output of each software and store their data into a unique structure, namely a “run” S4 object. This object contains essentially a list of various tables: one for estimates of population parameters, one for the individual parameters obtained through posthoc analysis, for the individual and population predictions, etc. By giving a path to a run (a NONMEM or ADAPT result file, a Monolix output folder . . . ), a function automatically recognizes the software and performs this run standardization. Each software requires unique steps for extracting the possible information. More information is available on GitHub under the corresponding source code. Each run can be created from scratch or from a second S4 object, called “folder”, containing several runs. Concretely, the user must use a specific closure function that takes the path of any folder:

demo <- createFolder("D:/.../anyfolder")

Several internal steps occur using createFolder() function. First, an algorithm searches for every run (for now Monolix 2016 and similar, Monolix 2019 and similar, NONMEM, ADAPT and nlmixr) contained in this folder or any subfolders. It produces a table containing one run per row, with its complete path, source software, objective function value and an ID number (Figure 9A). Second, this table is injected in an S4 folder object, containing an initially empty list used later to store desired runs. Third, a function is returned to manipulate this enclosed folder object. This function takes as an argument the ID number(s) of desired runs:

demo(1)  # This returns the run object of run number 1
demo(1:2) # This returns the folder object containing run number 1 and 2

The first time a run is requested, its creation (which takes a few seconds) is directly followed by its storage inside the “folder” object. The next time this run is requested, its creation is not re-performed.

Various functions have then been created to analyze either a single run (if a “run” object is passed as an argument), or to compare runs (if a “folder” is inputted), for instance:

results(demo(1)) # return the result of run number 1
results(demo(c(1,3))) # compare the results of run 1 and 3

This results() function provides the main results of a model, namely the criteria score (objective function, AIC, BIC), the parameter estimates with precision and, if provided, the shrinkage. If several run IDs are entered, a full join is automatically created to compare the run values (Figure 9B). An individual prediction (IPRED) function plots IPRED and PRED vs. time for each individual and each YTYPE. If several run IDs are inputted, the predictions will be plotted in the same graph with different colors (Figure 9C). The goodness of fit plots can be stratified by covariate, and users can specifically compare any subplot between desired runs (Figure 9D). The same system applies for random effect analysis (variance-covariance plots) or covariate analysis (Figure 9E). vpc and coveffectsplot packages have also been implemented in the Shiny app to simplify the production of visual predictive checks and forest plots, respectively.
Figure 9. PeccaResult is used for model diagnostics and model comparisons. A) The user provides a folder path (left dark panel), clicks on “go”, and a table is created (right) containing basic information and a unique ID for every runs found on the root folder and its subfolders. B) By selecting desired run IDs (here, 1, 6 and 8), a comparative table is created with one run per column and one result value per row. C) Individual (straight lines) and populational (dashed lines) fits with runs comparisons (one color per run). D) Sample of two Goodness of fit plots (one per row) for two runs (one per column). E) Covariate analysis (here, the dose considered as categorical) with runs comparison (one color per run).

The strength of this system is the universality of these functions. If one creates a new function working with runs, this function will directly work with all software (NONMEM, Monolix, . . . ). Reciprocally, if one adds a new software standardization (e.g., from Phoenix WinNonLin), then all Peccary functions will be directly available. The currently available functions cover the most common analysis performed after a run estimate. However, more complex projects often require very specific outputs. Users can then manually create a function using standardized run and import them into a project. Of final note, some parameter estimation software, for instance NONMEM, allows syntax flexibility, creating “personal ways of coding”. Those syntax variabilities can create difficulties for PeccaResult to extract all the needed information effectively. If previous effort has been made to consider several “personal ways” of coding in NONMEM, more work is required to make those extractions as flexible and adaptive as possible. This challenge and the difficulty of metaprogramming the system explain why PeccaResult should be used with caution and not integrated into a report. It remains, nevertheless, very useful to perform quick “informal” run results, visualization, and run comparisons.

Discussion

Generality

The use of multiple software is essential for pharmacometricians to improve efficiency, as no single program can perform all desired analyses. However, this creates significant efficiency issues. In this context, Peccary is an
R project created with the sole objective of improving the efficiency of pharmacometricians. So far, the most complex model on which Peccary has been used in real conditions (using both PeccAnalysis, PeccaReverse and PeccaResult functions) was a Monolix PK/PD model containing complex if-else structures, multiple covariates, 51 parameters, 18 ODEs, and five observation types fitted simultaneously.

Pandoc legacy
From many aspects, Peccary tried to reproduce the markdown pandoc efficiency but applied in PMx. First, Peccary can accelerate the use of various R packages (pknca, RxODE, PopED, table1, . . . ) by simplifying their syntax, and hence reducing 1) the required knowledge and 2) the time needed before launching these analyses. Second, Peccary can create links between the different packages. For instance, the same input tables in PeccaReverse are used both for simulations, design evaluation and syntax translations, improving the efficiency at the multi-software (or interoperability) level. The Shiny application is fast enough to be used during a live meeting, dynamically producing plots or analyses. Similarly, the Shiny app has educational potential, especially with its ability to easily reload any dataset or model simulations. In a second time, and thanks to its metaprogramming systems, it is possible to extract the peccary independent R code from most analyses, before copy/pasting it into a report to be checked and, if needed, secondarily modified. It is almost always faster to create a code through Peccary than coding it from scratch, even if secondary modifications are needed. If knowing the pre-existing packages integrated in Peccary is not mandatory to produce many outputs, it is nevertheless useful to learn them in order to reach the full potency of the system.

Toward a collaborative project
The Peccary project started in mid-2018 as a personal side project supporting the modeling work of the lead author, with no release ambition. After four years of adding different features, the current version of Peccary significantly improves the efficiency of PMx on a daily basis. Consequently, Peccary may be of interest to other pharmacometricians and is now aiming to be released. However, scaling such a project from a single user to multiple people is challenging. The chances of success would be optimized if the project became fully collaborative, in line with bazaar development. Thanks to its R-only structure and high modularity, every user can become a co-developer by checking, improving, or adding new features to Peccary. Being frequently updated, each modification or addition of a feature is registered in the “author section” in the Shiny app (authors listed by alphabetic order). Finally, any part of Peccary can be integrated in any other project without the authors’ initial authorization.

Fixing bugs
Due to the bazaar development approach of Peccary, there is a large heterogeneity in terms of Peccary function qualities. Some of them have been tested, used, improved and fixed several times until complete stability. Some others (the most recent or the less used ones) are more likely to contain errors or to be incomplete. Nevertheless, Peccary was designed to be useful even if imperfect and unvalidated. Despite the integration of several unit tests (e.g., for the peccary_nca() function, more information on GitHub), it is always the user’s responsibility to verify the results, as the purpose of Peccary is only to speed up the creation of R codes and have quick access to various results. Users can verify these outputs in two ways, 1) via the metaprogramming system and 2) by comparing the Peccary result to any other software making the same result. For the metaprogrammed functions, computational errors can occur either during the internal Peccary metaprogramming task or during the code evaluation. In the last case, it is still possible to extract the metaprogrammed code and manually fix it (most of the time requiring little effort and time). Above all, multiplying the numbers of co-developer would accelerate the detection and correction of these bugs, in accordance with Linus’s law “given enough eyeballs, all bugs are shallow”.

Adding new functionalities
There are also many features that need to be added. For example, the current version of Peccary does not support Time-To-Event data, mixture model, Bayesian analysis, and the Phoenix WinNonLin translation tool sets. . . . It should be noted that other types of ODE-based model types (QSP or PBPK modeling for instance) can be imported or created inside PeccaReverse. Being able to launch and program run parameter estimations through an R package (mlmix, saemix) or external software (NONMEM, Monolix. . . ) within Peccary can also be of great interest, requiring appropriate parallelization and computational server communications. Finally, any R package can potentially be metaprogrammed in Peccary to further merge more open-source tools within the same platform.
Conclusion
In summary, Peccary is an R package that increases the daily efficiency of a pharmacometrician. Although Peccary was initially developed as an individual project, it has now grown into a platform that aims to benefit the workflow process development for all pharmacometricians by integrating the different steps of the PMx into one tool. However, in line with the bazaar model, Peccary will only reach its full potential through a fully collaborative process.

Software availability

**Peccary source code:**
Software available using the following R code (automatically downloading all the subpackages): devtools::install_github("Peccary-PMX/Peccary")

Source code available from: https://github.com/Peccary-PMX/Peccary

Archived source code at time of publication: https://doi.org/10.5281/zenodo.684180

License: GNU General Public License v3.0.

**PeccAnalysis source code:**
Software available using the following R code (already downloaded with Peccary main package): devtools::install_github("Peccary-PMX/PeccAnalysis")

Source code available from: https://github.com/Peccary-PMX/PeccAnalysis

Archived source code at time of publication: https://doi.org/10.5281/zenodo.684182

License: GNU General Public License v3.0.

**PeccaReverse source code:**
Software available using the following R code (already downloaded with Peccary main package): devtools::install_github("Peccary-PMX/PeccaReverse")

Source code available from: https://github.com/Peccary-PMX/PeccaReverse

Archived source code at time of publication: https://doi.org/10.5281/zenodo.684182

License: GNU General Public License v3.0.

**PeccaResult source code:**
Software available using the following R code (already downloaded with Peccary main package): devtools::install_github("Peccary-PMX/PeccaResult")

Source code available from: https://github.com/Peccary-PMX/PeccaResult

Archived source code at time of publication: https://doi.org/10.5281/zenodo.684183

License: GNU General Public License v3.0.

Author contributions
TD initiated the project, designed the software architecture, implemented Peccary and wrote the original manuscript. SF supervised the project. SF and XD reviewed and edited the manuscript.

Acknowledgements
The authors would like to thank all the first Peccary users and testers, and in particular Blaise Pasquier and Christelle Rodrigues, in addition to Shruti Shah for her proofreading of the manuscript.
References

1. Barrett JS, Fossler MJ, David Cadiou K, et al.: Pharmacometrics: A multidisciplinary field to facilitate critical thinking in drug development and translational research settings. J Clin Pharmacol. 2006; 46(S): 632–649. PubMed Abstract | Publisher Full Text

2. van der Graaf PH: Regulatory modeling and simulation moves into the next gear in europe. CPT Pharmacometrics Syst Pharmacol. 2013; 2(2): e32. PubMed Abstract | Publisher Full Text | Free Full Text

3. Marshall S, Madabushi R, Malonis E, et al.: Model-informed drug discovery and development: Current industry good practice and regulatory expectations and future perspectives. CPT Pharmacometrics Syst Pharmacol. 2019; 8(2): 87–96. PubMed Abstract | Publisher Full Text | Free Full Text

4. Bucin T, Thoma Y, Wälder N, et al.: The steps to therapeutic drug monitoring: A structured approach illustrated with imatinib. Front Pharmacol. 2020; 11: 177. PubMed Abstract | Publisher Full Text | Free Full Text

5. Wilkins J, Chan P, Chard J, et al.: Thoughtflow: Standards and tools for provenance capture and workflow definition to support model-informed drug discovery and development. CPT Pharmacometrics Syst Pharmacol. 2017; 6(5): 285–292. PubMed Abstract | Publisher Full Text | Free Full Text

6. Grasela TH, Dement CW: Interoperability framework for r open-source packages. F1000Research 2022, 11:951 Last updated: 10 NOV 2022

7. Bauer Rj: NONMEM Tutorial Part I: Description of Commands and Options, With Simple Examples of Population Analysis. CPT Pharmacometrics Syst Pharmacol. 2019; 8(2): 525–537. Publisher Full Text

8. Monolix, Lixoft SAS, A Simulations Plus company. Reference Source

9. WinNonlin® version 5.3 (Certara L.P. (Pharsight), St. Louis, MO). Reference Source

10. Burches E, Burches M: Efficacy, effectiveness and efficiency in the health care: The need for an agreement to clarify its meaning. Int Arch Public Health Community Med. 2020; 4: 035. Publisher Full Text

11. Vlasakakis G, Cornets E, Keunecke A, et al.: White paper: Landscape on technical and conceptual requirements and competence framework in drug/disease modeling and simulation. CPT Pharmacometrics Syst Pharmacol. 2013; 2(5): e40. PubMed Abstract | Publisher Full Text | Free Full Text

12. Mentré F, Chenel M, Comets E, et al.: Current use and developments needed for optimal design in pharmacometrics: A study performed among DMORe's european federation of pharmaceutical industries and associations members. CPT Pharmacometrics Syst Pharmacol. 2013; 2(5): e46. PubMed Abstract | Publisher Full Text | Free Full Text

13. Hucka M, Finney A, Bornstein BJ, et al.: Evolving a lingua franca and associated software infrastructure for computational systems biology: the systems biology markup language (SBML) project. Syst Biol (Stevens). 2004; 1(1): 41–53. PubMed Abstract | Publisher Full Text

14. Martin J, Matthews L, Chard J, et al.: Drug and disease model resources: A consortium to create standards and tools to enhance model-based drug development. CPT Pharmacometrics Syst Pharmacol. 2013; 2(3): e54. PubMed Abstract | Publisher Full Text | Free Full Text

15. Bizzotto R, Cornets E, Smith G, et al.: PharmML in action: an interoperable language for modeling and simulation. PharmML in action. CPT Pharmacometrics Syst Pharmacol. 2017; 6(10): 651–665. PubMed Abstract | Publisher Full Text | Free Full Text

16. Smith MK, Moodie SL, Bizzotto R, et al.: Model description language (MDL): A standard for modeling and simulation. CPT Pharmacometrics Syst Pharmacol. 2017; 6(10): 647–650. PubMed Abstract | Publisher Full Text | Free Full Text

17. Interoperability framework. Reference Source

18. Pandoc - about pandoc. Reference Source

19. Chang W, Cheng J, Allaire J, et al.: shiny: Web application framework for r. Reference Source

20. The GNU general public license v3.0 - GNU project - free software foundation. Reference Source

21. The MIT license. open source initiative. Reference Source

22. Denney B: Introduction to PNKCA and usage instructions. Reference Source

23. Keizer Rj, Karlsson MO, Hooker A: Modeling and Simulation Workbench for NONMEM: Tutorial on Pirana, PeN, and Xpose. CPT Pharmacometrics Syst Pharmacol. 2013; 2(6): e50. PubMed Abstract | Publisher Full Text | Free Full Text

24. Comets E, Brentel K, Mentré F: Computing normalised prediction distribution errors to evaluate nonlinear mixed-effect models: the nped add-on package for r. Comput Methods Programs Biomed. 2008; 90(2): 154–166. PubMed Abstract | Publisher Full Text

25. Keizer R: vpcl: Create visual predictive checks. 2021. Reference Source

26. Bazoli C, Retout S, Mentré F: Design evaluation and optimisation in multiple response nonlinear mixed effect models: PFIM 3.0. Comput Methods Programs Biomed. 2010; 98(1): 55–63. PubMed Abstract | Publisher Full Text

27. Foracchia M, Hooker A, Vicini P, et al.: POPED, a software for optimal experimental design in population kinetics. Comput Methods Programs Biomed. 2004; 74(1): 29–46. PubMed Abstract | Publisher Full Text

28. Simulx 2021R2, Lixoft SAS, A Simulations Plus company. Reference Source

29. Fidler M, Wilkins Jj, Houijmajaras R, et al.: Nonlinear mixed-effects model development and simulation using nlmixr and related r open-source packages. CPT Pharmacometrics Syst Pharmacol. 2019; 8(9): 621–633. PubMed Abstract | Publisher Full Text | Free Full Text

30. Comets E, Lavenu A, Lavieille M: Parameter estimation in nonlinear mixed effect models using saemix, an r implementation of the SAEM algorithm. J Stat Softw. 2017; 80(3): 1–41. Publisher Full Text

31. Nolain P, Combet R, Marchionni D, et al.: PopkinR: a suite of shiny applications focused on the pharmacometrics workflow. 1. Reference Source

32. Lixoft: R-package installation and initialization. Reference Source

33. Raymond ES: The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary. First edition. Beijing: Cambridge, [Mass.]: O'Reilly, 1999. Reference Source

34. Wickham H: Tidy data. J Stat Softw. 2014; 59(10): 1–23. Publisher Full Text

35. Wickham H: Advanced r - metaprogramming. Reference Source

36. Wickham H: Advanced r - functional programming. Reference Source

37. Rich B: table1: Tables of descriptive statistics in HTML. 2021. Reference Source

38. Kassambara A: ggpubr: ‘ggplot2’ based publication ready plots. Reference Source

39. Wickham H: Advanced r - s4. Reference Source

40. Wickham H: Advanced r - functional programming. Reference Source

41. Mouksissi S, Attali D: coveffectplot: Produce forest plots to visualize covariate effects. 2022. Reference Source

42. Peccary-PMX, Derippe T: Peccary-PMX/Peccary: F1000_version. https://zenodo.org/record/6841801

43. Derippe T, Peccary-PMX: Peccary-PMX/PeccAnalysis: F1000research submission. https://zenodo.org/record/6841823

44. Derippe T, Peccary-PMX: Peccary-PMX/PeccResult: F1000research submission. https://zenodo.org/record/6841833
Open Peer Review

Current Peer Review Status: ?

Version 1

Reviewer Report 08 November 2022

https://doi.org/10.5256/f1000research.136060.r153791

© 2022 Tosca E. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Elena Maria Tosca
Dipartimento di Ingegneria Industriale e dell'informazione, Università degli Studi di Pavia, Pavia, Italy

This work proposes a new R package, Peccary, integrated with its Shiny App, to support pharmacometric modeling activities. Peccary comprises three sub-packages: PeccAnalysis, dealing with pre-modeling analysis, PeccaReverse, relating with model building and simulation, and PeccaResult for model diagnostics. The goal of Peccary is to improve the efficiency of real-life pharmacometrics activities accelerating the use of previously released R packages and creating links between independent tools through various translation and metaprogramming processes. The work is of great interest and Peccary could potentially provide an extremely useful support to modeling activities standardly performed during a pharmacometrics project. However, the actual potentialities of Peccary to improve work of pharmacometricians have to be better demonstrated in the paper.

For example, the description of PeccaReverse is mainly focused on the implementation and simulation of a model through deSolve and/or RxODE. However, a pharmacometric model is typically developed in NONMEM or Monolix and it wouldn't be efficient to re-code the model, the parameters and the study design to make simulations through PeccaReverse functions. I think that the possibility of importing NONMEM and Monolix projects could be of great value and could increase the usability of PeccaReverse package. In the paper, only a short paragraph was included on this topic and any example was reported. I strongly suggest to extend this point, also providing examples. In addition, the authors reported that only simplest study designs can be imported and translated for now. Please provide a more complete list of supported features.

As also reported by the authors in the paper, the use of multiple software is a common practice during a pharmacometrics project. For example, a very common situation is to develop and identify a model using a software (NONMEM) and, subsequently, to perform simulations through another tool (SimulX). The possibility of switch from one tool to another without re-coding the model and the entire set-up would allow to strongly improve efficiency. I would like to see an example in which a model (and the entire set-up) is developed and identified in a tool, it is imported in Peccary and then translated into another syntax to perform simulations through a
different software. Is it currently possible in Peccary?

Minor: Most Peccary functions have been metaprogrammed. This is a useful option that allows to learn what a function does and to check and personalize the R code. However, always printing the R code in the R console before the outputs could prevent an easy reading of the results. For this reason, I strongly suggest to set the code printing as an function option that the used can select if needed.

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Partly

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Partly

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Mathematical modelling in pharmacometrics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.
The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com