Pmetrics User Manual

LAPKB

2022-05-22

Contents

Chapter 1

Preface

This is the manual for Pmetrics, a population modeling and simulation package for R.

Pmetrics is the result of years of labor from many people. It was created by the Laboratory of Applied Pharmacokinetics and Bioinformatics (LAPKB).

1.1 A brief history

LAPKB, established in 1973 as LAPK by Roger Jelliffe, MD, has been continually associated with the University of Southern California and the USC Keck School of Medicine. Since 2012, it has been housed under the Saban Research Institute at Children's Hospital Los Angeles (CHLA).

Since its inception, LAPKB has been a pharmacometric resource for optimal study and control of pharmacokinetic/pharmacodynamic systems and for individualized drug therapy and personalized medicine. It has been continually supported by grants, including from The National Institute for General Medical Studies (NIGMS), National Institute of Biomedical Imagining and Bioengineering (NIBIB), the Eunice Kennedy Shriver National Institute of Child Health and Human Development (NICHD), the US Food and Drug Administration (FDA), and by the Stella Slutzky Kunin Memorial Research Fund.

The laboratory has employed physicians, pharmacists, engineers, statisticians, and mathematicians. LAPKB has special strengths in nonparametric statistical methods, optimal stochastic control, optimal design of pharmacokinetic experiments and clinical trials, and practical application of tools for optimal clinical therapy.

The laboratory also seeks collaborative relationships to further the understanding and development of this field of Clinical Pharmacology. These collaborations may take the form of clinical trials and evaluations of therapeutic methods or of development and software implementation of new concepts. Educational opportunities in the form of workshops and visiting scholars are available to physicians, pharmacists, engineers, mathematicians, and other investigators.

1.2 People

This is not an exhaustive list by any means, but highlights some individuals who have made exceptional contributions to the lab.

- Roger Jellife, MD - founder and pioneer. Passed away June 22, 2002.
- Alan Schumitzky, PhD - Emeritus Professor of Mathematics at USC. Professor Schumitzky's research interests are focused on estimation and control theory, applied pharmacokinetics, complex analysis, and software development. He developed NPEM, co-developed NPAG, NPOD, and every other algorithm from the lab, and he continues to share his genius.
- Robert Leary, PhD - co-developer of NPAG and former consultant to the lab.
- David Bayard, PhD - consultant to the lab and expert in optimal control. He developed the Multiple Model algorithm powering BestDose, as well as the MMopt optimal sampling algorithm.
- Michael van Guilder PhD - consultant to the lab who turned all of the ideas into working, stable, reliable Fortran code.
- Walter Yamada, PhD - Current scientific programmer, modeler, and the person who knows the code the best and manages the lab on a daily basis. He developed the Poisson likelihood function and makes updates to the code every day.
- Julian Otalvaro - Scientfic programmer who has transformed Pmetrics by developing the R6 framework, moving it to Github, and just about every other aspect of the package. He is also one of the labs experts in machine learning.
- Rong Chen, PhD - post-doc and author of RPEM. He is our expert in Fortran and use of MPI to parallelize code.
- Alona Kryshchenko, PhD - consulant and co-developer of NPOD.
- Michael Neely, MD - current leader of the lab, physician and dabbler in statistics. He wrote all of the original Pmetrics package and continues to write to this day, in addition to all the other jobs associate with Principle Investigator and Chief of the Division of Infectious Diseases at CHLA.

1.3 Citing Pmetrics

Please help us maintain our funding to provide Pmetrics as a free research tool to the pharmacometric community. If you use Pmetrics in a publication, you can cite it as below.

```
citation("Pmetrics")
```

```
##
## To cite package 'Pmetrics' in publications use:
##
##   Neely MN, van Guilder MG, Yamada WM, Schumitzky A, Jelliffe RW.
##   Accurate detection of outliers and subpopulations with Pmetrics, a
##   nonparametric and parametric pharmacometric modeling and simulation
##   package for R. Therapeutic Drug Monitoring. 2012; 34(4): 467-476.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Accurate Detection of outliers and subpopulations with Pmetrics: a non-parametric
##     author = {Michael Neely and Michael {van Guilder} and Walter Yamada and Alan Schumitzky an
##     year = {2012},
##     journal = {Therapeutic Drug Monitoring},
##     volume = {34},
##     number = {4},
##     pages = {467-476},
##   }
```

1.4 Donate

If you appreciate the enormous work to develop and maintain Pmetrics, please consider a donation.

Chapter 2

Introduction

Thank you for your interest in Pmetrics! This guide provides instructions and examples to assist users of the Pmetrics R package, by the Laboratory of Applied Pharmacokinetics and Bioinformatics at the Saban Research Institute, Children's Hospital Los Angeles, and the Department of Pediatrics, Keck School of Medicine, University of Southern California. Please see our website at http://www.lapk.org for more information.

2.1 R6 architecture

As of v. 2.0, Pmetrics is shifting to an architecture less dependent on reading and writing files. Data files are unchanged and described later in this manual.

Model files can now be defined as an R object, instead of in a text file. Pmetrics can suppoort legacy runs with the old-style text file models, but users are encouraged to change to the new methods. Throughout this manual we will indicate the new style with the R6 designation to reflect the object-oriented R6 style of programming available in R that makes it more consistent with object-oriented languages such as Python. We will indicate old-style approaches with Legacy.

Here are some tips for using this guide.

- The table of contents to the left is expandable and navigable.
- Items that are hyperlinked can be selected to cross reference within this manual or link to external sites.
- `Items` correspond to inline examples of R code, which are not evaluted in this document, but serve as templates for what may be typed into your

R console or script. They may not necessarily be executable if typed verbatim.

2.2 Disclaimer

You, the user, assume all responsibility for acting on the results obtained from Pmetrics. The Laboratory of Applied Pharmacokinetics and Bioinformatics (LAPKB), members and consultants to LAPKB, and Children's Hospital Los Angeles and the University of Southern California and their employees assume no liability whatsoever. Your use of the package constitutes your agreement to this provision.

2.3 System Requirements and Installation

Pmetrics and all required components will run under Mac (Unix), Windows, and Linux. There are three *required* software components which must be installed on your system **in this order**:

1. The statistical programming language and environment "**R**"
    - After installing R, we highly recommended that you also install **Rstudio**, a user-friendly environment for R.
2. The **Pmetrics** package for R
3. The **gfortran** Fortran compiler.

All components have versions for Mac, Windows, and Linux environments, and 64- bit processors. Systems with 32-bit processors are no longer supported. All are free of charge.

2.3.0.1 R

R is a free software environment for statistical computing and graphics, which can be obtained from http://www.R-project.org. Pmetrics is a library for R.

2.3.0.2 Rstudio

We strongly recommend using Rstudio rather than any other R interface.

2.3.0.3 Pmetrics

If you are reading this manual, then you have likely visited our website at http://www.lapk.org, where you can select the software tab to access instructions. As of version 1.9, Pmetrics is distributed on github and is a self-contained package that will install gfortran with your permission, if it is not already installed on your computer. Installing from github will also install all packages upon which Pmetrics depends.

2.3.0.4 Gfortran

In order to run Pmetrics, a Fortran compiler compiler is required. Pmetrics is designed to work with gfortran, a free compiler. After you have installed Pmetrics, it will check your system for an active gfortran installation. If it doesn't find one, it will offer to download and install it. From there, installation should proceed automatically. This is by far the easiest and most reliable way to complete installation. Rest assured that no files are installed without your permission.

If you do not wish to do this, you will have to get components manually, and the first command to run is `PMbuild()`. You can get detailed instructions on how to obtain and install gfortran appropriate for your system on our LAPKB website.

2.4 What This Manual Is Not

We assume that the user has familiarity with population modeling and R, and thus this manual is not a tutorial for basic concepts and techniques in either domain. We have tried to make the R code simple, regular and well documented. A very good free online resource for learning the basics of R can be found at Stat Methods.

We recognize that initial use of a new software package can be complex, so please feel free to contact us at any time, preferably through the Pmetrics forum or directly by email.

This manual is also not intended to be a theoretical treatise on the algorithms used in IT2B or NPAG. For that, the user is directed to our website.

2.5 Getting Help and Updates

Within R, you can use `help("command")` or `?command` in the R console to see detailed help files for any Pmetrics command. Many commands have examples included in this documentation and you can execute the examples with `example(command)`.

Pmetrics will check for updates automatically every time you load it with `library(Pmetrics)` and you are connected to the internet. If an update is available, it will provide a brief message to inform you. You can then reinstall the package from github.

2.6 Customizing Pmetrics Options

You can change global options in Pmetrics with `setPMoptions(sep, dec,server\_address)`.

Currently you can change three options: `sep` and `dec` will allow Pmetrics to read data files whose field separators are semicolons and decimal separators are commas, e.g. `setPMoptions(sep=";", dec=",")`. These options will persist from session to session until changed. The third option, `server\_address`, allows you to specify the address of a remote server with Pmetrics installed, to allow remote runs.

`getPMoptions()` will return the current options.

Chapter 3

Pmetrics Overview

3.1 Software engines

There are three main software engines that Pmetrics controls.

- **IT2B** is the ITerative 2-stage Bayesian parametric population PK modeling program. It is generally used to estimate parameter ranges to pass to NPAG. It will estimate values for population model parameters under the assumption that the underlying distributions of those values are normal or transformed to normal.

- **NPAG** is the Non-parametric Adaptive Grid software. It will create a non-parametric population model consisting of discrete support points, each with a set of estimates for all parameters in the model plus an associated probability (weight) of that set of estimates. There can be at most one point for each subject in the study population. There is no need for any assumption about the underlying distribution of model parameter values.

- The **Simulator** is a semi-parametric Monte Carlo simulation software program that can use the output of IT2B or NPAG to build randomly generated response profiles (e.g. time-concentration curves) for a given population model, parameter estimates, and data input. Simulation from a non-parametric joint density model, i.e. NPAG output, is possible, with each point serving as the mean of a multivariate normal distribution, weighted according to the weight of the point. The covariance matrix of the entire set of support points is divided equally among the points for the purposes of simulation.

3.2 Pmetrics control functions

R6

Pmetrics uses`PM_data` to create data objects, `PM_model` to create model objects, and `PM_fit` to create objects that combine the model with the data, ready to be run (fitted), generating probability distributions for primary model parameters. These are extensively documented within R by using the `help(command)` or `?command syntax`.

These functions replace the following Legacy functions: `ITrun`, `ERRrun`, `NPrun`.

Invoking the simulator in R6 becomes a method attached to `PM_result` objects.

Legacy

Pmetrics has groups of R functions named logically to run each of these programs and to extract the output. Again, these are extensively documented within R by using the `help(command)` or `?command syntax`.

- `ITrun`, `ITparse`, `ERRrun`
- `NPrun`, `NPparse`
- `PMload`, `PMsave`, `PMreport`
- `SIMrun`, `SIMparse`

3.2.1 Run functions

R6

Once a `PM_fit` object is created, which combines a model with a data file, it can be run by using the syntax `$run()` to access the appropriate function defined for the `PM_fit` object.

```
fit1 <- PM_fit$new(model, data)
fit1$run(options)
```

R6 Legacy

For IT2B and NPAG, the "run" functions generate batch files, which when executed, launch the software programs to do the analysis. `$run(engine="err")` or `ERRrun()` is a special implementation of IT2B designed to estimate the assay error polynomial coefficients from the data, when they cannot be calculated from assay validation data (using `makeErrorPoly()`) supplied by the analytical laboratory. The batch files contain all the information necessary to complete a run, tidy the output into a date/time stamped directory with meaningful subdirectories, extract the information, generate a report, and a saved Rdata file of parsed output which can be quickly and easily loaded into R. On Mac (Unix) and Linux systems, the batch file automatically launches in a Terminal window. Prior to v1.9, on Windows systems, the batch file was launched manually, but as of v1.9, this manual step is no longer necessary. The execution of the program to do the actual model parameter estimation is independent of R, so that the user is free to use R for other purposes.

R6

For the Simulator, the `$sim` method for a `PM_result` object will execute the program directly within R and return a `PM_sim` object. An equivalent method is to construct a new `PM_sim` directly.

s

Legacy

To run the simulator in legacy mode, use `SIMrun`. Results are saved to the hard drive in the working directory, and can be read with `SIMparse` as described below.

```
SIMrun(...)
```

3.2.2 Parse functions

R6 Legacy

For all programs, the "parse" functions will extract the primary output from the program into meaningful R data objects. For IT2B and NPAG in either R6 or legacy modes, this is done automatically at the end of a successful run, and the objects are saved in the output subdirectory as IT2Bout.Rdata or NPAGout.Rdata, respectively. The parse functions run automatically and are not generally necessary for the user to access.

R6

For the simulator the results are returned automatically to the object assigned to contain the output of the simulator, e.g. `sim1` below.

```
sim1 <- PM_result$sim(...)
```

There is no longer any need to use the `SIMparse` legacy function described below, because the `SIMrun` and `SIMparse` legacy functions are combined within the `$sim()` method for `PM_result` objects or the `$new()` method for `PM_sim` objects.

Legacy

After running the simulator with `SIMrun`, the results are saved to files in the working directory. They can be retrieved using `SIMparse`.

```
simdata <- SIMparse("simout1.txt")
```

3.2.3 Saving functions

R6

The Pmetrics R6 objects `PM_fit`, `PM_result`, `PM_sim`, `PM_valid`, and `PM_pta` all have a `$save()` method. This method saves the object to the hard drive in the current working directory by default. The format is .rds which is a binary format used by R to save individual objects. The purpose of the `$save()` method is to enable retrieval of the object at a later time.

Legacy

Objects created that are attached to a particular run can be identified by using the same suffix number as the run. `PMsave` can add these attached objects to the .Rdata file for that run, which is loaded with `PMload`.

```
sim.2 <- SIMparse("simout1.txt")
PMsave(2)
```

In the above example, `sim.2` will be added to the .Rdata file for future retrieval with `PMload(2)`.

3.2.4 Loading functions

R6

After a successful IT2B or NPAG run, `PM_load` creates a `PM_result` object rather than loading run results into the current environment and suffixed with the run number as for legacy mode.

```
res1 <- PM_load(1)
res1$op$plot()
```

`PM_result`, `PM_fit`, `PM_sim`, `PM_valid`, and `PM_pta` all have a `$load()` method that takes as its only argument the full or relative path (if not in the working directory) and name of an .rds file created by the corresponding `$save()` method.

```
sim1 <- PM_sim$load("sim.rds")
```

Legacy

For IT2B and NPAG, the `PMload` function can be used to load the .Rdata file after a successful run. Objects will be loaded into the current environment in R and suffixed with ".run", where "run" is the run number.

```
PMload(1)
plot(op.1)
```

For simulations, `SIMparse`, described above, combines the parsing and loading functions.

```
sim.2 <- SIMparse("simout1.txt")
```

3.2.5 Report generation

R6 Legacy

The `PMreport` function is automatically run at the end of a successful NPAG and IT2B run, and it will generate an HTML page with summaries of the run, as well as the .Rdata files and other objects. The default browser will be automatically launched for viewing of the HTML report page. See the Pmetrics Outputs section.

3.3 Other functions

R6 Legacy

Within Pmetrics there are also functions to manipulate data .csv files and process and plot extracted data.

3.3.1 Data manipulation

Function	R6	Legacy
Read data file	PM_data\$new()	PMreadMatrix()
Check data file	Embedded in PM_data\$new()	PMcheck()
Write data file	PM_data\$write()	PMwriteMatrix()
Convert calendar dates and clock times	Embedded in PM_data\$new()	PMmatrixReltime()
Convert from old USC*PACK .wrk format	PMwrk2csv()	PMwrk2csv()
Convert from NONMEM	NM2PM()	NM2PM()
Convert from old USC*PACK .mb format	PMmb2csv()	PMmb2csv()

3.3.2 Model selection and diagnostics

Function	R6	Legacy
Compare models	PM_compare(PM_result1, PMresult2,...)	PMcompare(1, 2, ...)
Plot residuals	PM_result$op$plot(resid = T,...)	plot(op, resid = T)
Construct VPC, pcVPC, NPDE	PM_result\$valid() or PM_valid\$new()	makeValid()
Plot VPC, pcVPC, NPDE	PM_valid$plot()	plot(PMvalid)
Stepwise covariate regression	PM_result$step()	PMstep()

3.3.3 Other functions

```
## Warning in read.table(file = file, header = header, sep = sep, quote = quote, :
## incomplete final line found by readTableHeader on 'Data/RLcomp_other.csv'
```

Function	R6	L
Calculate AUC	\$auc() method for PM_result\$op/\$post/\$pop, or PM_sim	m
Simulate	PM_result\$sim() or PM_sim\$new()	S
Probability of target attainment	PM_sim\$pta() or PM_pta\$new()	m

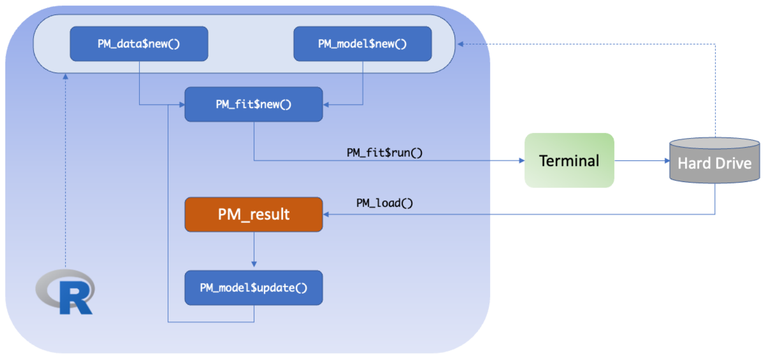
Again, all functions have extensive help files and examples which can be examined in R by using the `help(command)` or `?command` syntax.

Chapter 4

General Workflow

R6

The general Pmetrics workflow in R6 for IT2B and NPAG is shown in the following diagram.



The user creates the data and model objects at the top. These can come from the hard drive or from within R (dashed arrows). The model file is created in R using the `PM_model$new()` function and the data file by `PM_data$new()`. When combined using `PM_fit$new()` and the `$run()` function on the resulting object, the analysis is executed. At the end of the run, the hard drive will contain a new numerically named folder, e.g., 1, 2, 3, …, that contains the files which can be loaded into R subsequently using `PM_load(x)`, replacing `x` with the folder number. `PM_load()` is an alias for `PM_result$new()` because it creates a new

`PM_result` object which contains all the results of a run, and has many assciated methods attached to it for plotting, simulating, etc.

To change model parameters, the model object is updated via `PM_model$update()`, `PM_fit$new()` is called and then the run is executed with `$run()`. If continuing a previous run that did not end, simply use the `$run()` and specify the run number you wish to continue as the `prior` argument to `$run()`. These scenarios are illustrated below.

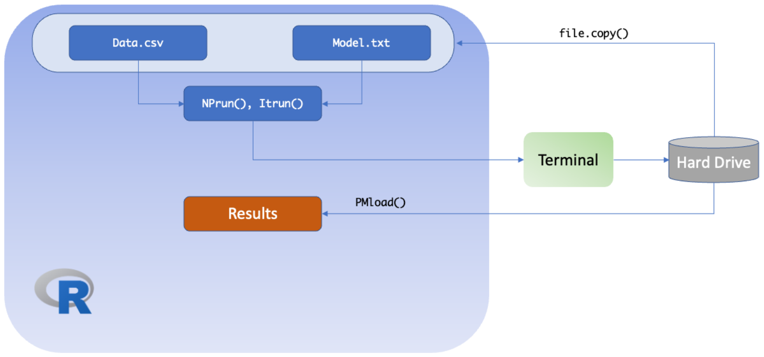
```
#Run 1 - ensure the data, model files are in the working directory
data1 <- PM_data$new("data.csv")
mod1 <- PM_model$new("model.txt")
fit1 <- PM_fit$new(data1, mod1)
fit1$run()
res1 <- PM_load(1) #PM_load is an alias for PM_result$new()

#Run 2 - update Ke range
mod2 <- mod1$update(list(pri = list(Ke = range(0.5, 3))))
fit2 <- PM_fit$new(data1, mod2)
fit2$run()

#Run 3 - continue run 2
fit2$run(prior = 2)
```

The great advantage of R6 over Legacy is that in R6, you no longer need to spend time copying files from prior run folders, modifying them, and ensuring that they are in the working directory. After the initial creation of the data and model objects, everything can be done in R from memory, although results are still saved to hard drive for later retrieval.

Legacy

The general Pmetrics workflow in Legacy for IT2B and NPAG is shown in the following diagram. The major differences compared to R6 are:

- *Data and model* The data is always a .csv file, and the model is always a text file which must be edited to make changes. Both must be copied from a prior folder to the current working directory for every run.
- *Run.* Separate commands for NPAG and IT2B must be used, i.e. `NPrun()` and `ITrun()`.
- *Loading.* Results are loaded with `PMload()` not `PM_load()` because the former returns a collection of results all suffixed with the run number, e.g. `op.1` or `final.1` for run 1. This can lead to confusion if working on two projects at once, or uncertainty about what is actually loaded.



The user supplies the names of the data and model files as arguments to the run functions. R is used to specify the working directory containing these files. The analysis is executed in a terminal window by calling `NPrun`, `ITrun` or `ERRrun` functions. The results can be loaded into R after the run completes using `PMload`. The analgous Legacy code to the R6 code above is shown here.

```
#Run 1 - ensure the data, model files are in the working directory
NPrun("data.csv", "model.txt")
PMload(1)

#Run 2 - update Ke range - edit file outside R, copy to working directory
#use same data as for run 1
NPrun(data = 1, "model2.txt")

#Run 3 - continue run 2
NPrun(data = 1, model = 1, prior = 2)
```

While there are fewer lines of R code than for R6, the edits made to create model2.txt are not recorded anywhere in R, and must be made outside R, with cumbersome file copying to ensure everything is in the working directory.

Chapter 5

Pmetrics Input Files

5.1 Data.csv Files

Pmetrics accepts input as a spreadsheet “matrix” format. It is designed for input of multiple records in a concise way. **Please keep the number of characters in the file name 8.**

In R6 use `PM_data$new("filename")` to create a `PM_data` object by reading the file.

```r
#ensure that data.csv is in the working directory
data1 <- PM_data$new("data.csv")
```

You can also build an appropriate data frame in R and provide that as an argument to `PM_data$new()`.

```r
#assume df is data frame with at least these columns:
#id, time, dose, out
data1 <- PM_data$new(df)
```

Once you have created the `PM_data` object, you never need to create it again during your R session. You also don’t have to bother copying the data file to the Runs folder each time you run the model.

In Legacy you must always have the the data file in the current working directory. You can manually copy it there from a previous run or some other folder or use the shortcut of providing a prior run number as an argument to `NPrun` or `ITrun`.

```
#Run 1 - ensure that data.csv is in the working directory
NPrun("data.csv", "model.txt")

#run 2 - use the data from run 1 in this run
#note that the file model.txt still has to be copied
# into the working directory in this example
NPrun(data = 1, "model.txt")
```

5.1.1 Data file format

If you create your data as a file, which is the most common scenario, the file should be in comma-separated-values (.csv) format. It is possible to use other separators, like the semicolon, by setting the appropriate argument with `setPMoptions(sep = ";")`.

Examples of programs that can save .csv files are any text editor (e.g. TextEdit on Mac, Notepad on Windows) or spreadsheet program (e.g. Excel).

The file or data frame format is much more flexible in R6. The only required columns are id, time, dose, and out. You may also specify time as clock time if you include a date column. The default format of the date column is YYYY-MM-DD and HH:MM for the time column by default, but other formats can be specified. See `?PM_data` for more details. There is no header required, the column order can be anything you wish, but the names should be the same as in the legacy format below. Ultimately, `PM_data$new()` converts all valid data into the format used in legacy Pmetrics.

Legacy

All .csv files must be formatted as below. As for R6, other separators are possible by using `setPMoptions()`.

IMPORTANT: The order, capitalization and names of the header and the first 12 columns are fixed. All entries must be numeric, with the exception of ID and "." for non-required placeholder entries.

POPDATA DEC_11

#ID	EVID	TIME	DUR	DOSE	ADDL	II	INPUT	OUT	OUTEQ	C0	C1	C2	C3
GH	1	0.00	0.0	400	.	.	1	.	.	.	.	.	.
GH	0	0.50	.	.	.	.	.	0.42	1	.	.	.	.
GH	0	1.00	.	.	.	.	.	0.46	1	.	.	.	.
GH	0	2.00	.	.	.	.	.	2.47	1	.	.	.	.
GH	4	0.00	0.0	150	.	.	1	.	.	.	.	.	.
GH	1	3.50	0.5	150	.	.	1	.	.	.	.	.	.
GH	0	5.12	.	.	.	.	.	0.55	1	.	.	.	.
GH	0	24.00	.	.	.	.	.	0.52	1	.	.	.	.
1423	1	0.00	1.0	400	-1	12	1	.	.	.	.	.	.
1423	1	0.10	0.0	100	.	.	2	.	.	.	.	.	.
1423	0	1.00	.	.	.	.	.	-99	1	0.01	0.1	0.00	0.000
1423	0	2.00	.	.	.	.	.	0.38	1	0.01	0.1	0.00	0.000
1423	0	2.00	.	.	.	.	.	1.6	2	0.05	0.2	-0.11	0.002

- ***POPDATA DEC_11*** This is the fixed header for the file and must be in the first line. It identifies the version. It is not the date of your data file.

- ***#ID*** This field must be preceded by the "#" symbol to confirm that this is the header row. It can be numeric or character and identifies each individual. All rows must contain an ID, and all records from one individual must be contiguous. Any subsequent row that begins with "#" will be ignored, which is helpful if you want to exclude data from the analysis, but preserve the integrity of the original dataset, or to add comment lines. IDs should be 11 characters or less but may be any alphanumeric combination. **There can be at most 800 subjects per run.**

- ***EVID*** This is the event ID field. It can be 0, 1, or 4. Every row must have an entry.

  - 0 = observation
  - 1 = input (e.g. dose)
  - 2, 3 are currently unused
  - 4 = reset, where all compartment values are set to 0 and the time counter is reset to 0. This is useful when an individual has multiple sampling episodes that are widely spaced in time with no new information gathered. This is a dose event, so dose information needs to be complete.

- ***TIME*** This is the elapsed time in decimal hours since the first event. It is not clock time (e.g. 09:30), although the `PMmatrixRelTime` function can convert dates and clock times to decimal hours. Every row must have an entry, and within a given ID, rows must be sorted chronologically, earliest to latest.

- ***DUR*** This is the duration of an infusion in hours. If EVID=1, there must be an entry, otherwise it is ignored. For a bolus (i.e. an oral dose), set the value equal to 0.
- ***DOSE*** This is the dose amount. If EVID=1, there must be an entry, otherwise it is ignored.
- ***ADDL*** This specifies the number of additional doses to give at interval II. It may be missing for dose events (EVID=1 or 4), in which case it is assumed to be 0. It is ignored for observation (EVID=0) events. Be sure to adjust the time entry for the subsequent row, if necessary, to account for the extra doses. If set to -1, the dose is assumed to be given under steady-state conditions. ADDL=-1 can only be used for the first dose event for a given subject, or an EVID=4 event, as you cannot suddenly be at steady state in the middle of dosing record, unless all compartments/times are reset to 0 (as for an EVID=4 event). To clarify further, when ADDL=-1, all compartments in the model will contain the predicted amounts of drug at the end of the 100th II interval.
- ***II*** This is the interdose interval and is only relevant if ADDL is not equal to 0, in which case it cannot be missing. If ADDL=0 or is missing, II is ignored.
- ***INPUT*** This defines which input (i.e. drug) the DOSE corresponds to. Inputs are defined in the model file.
- ***OUT*** This is the observation, or output value. If EVID=0, there must be an entry; if missing, this must be coded as -99. It will be ignored for any other EVID and therefore can be ".". There can be at most 150 observations for a given subject.
- ***OUTEQ*** This is the output equation number that corresponds to the OUT value. Output equations are defined in the model file.
- ***C0, C1, C2, C3*** These are the coefficients for the assay error polynomial for that observation. Each subject may have up to one set of coefficients per output equation. If more than one set is detected for a given subject and output equation, the last set will be used. If there are no available coefficients, these cells may be left blank or filled with "." as a placeholder.
- ***COV***... Any column after the assay error coefficients is assumed to be a covariate, one column per covariate. The first row for any subject must have a value for all covariates, since the first row is always a dose. Covariate values are applied at the time of doses.

5.1.2 Manipulation of CSV files

- As we have seen, in R6 `PM_data$new("filename")` will read an appropriate data file in the current working directory to create a new `PM_data`

object. In Legacy
`PMreadMatrix("filename", ...)` reads *filename* and creates a PMmatrix object in R which can be plotted (see `?plot.PMmatrix`) or otherwise analyzed. However, unlike R6, it cannot be used to run a model. For that, you need to copy the file into the working directory each time, either yourself or by using the `NPrun(data = 1, ...)` shortcut, for example.

- R6 `PM_data$write("filename")` will write the `PM_data` object to a file called "filename". This can be useful if you have loaded or created a data file and then changed it in R. Legacy `PMwriteMatrix(data.frame, "filename", ...)` writes an appropriate data frame as a new .csv file. It will first check the data.frame for errors via the `PMcheck()` function below, and writing will fail if errors are detected. This can be overridden with `override=T`.

- R6 `PM_data$new()` automatically calls `PMcheck(filename / PMmatrix, model,...)`, which is also available as an indepedent function in either R6 or Legacy. This function will check a .csv file named *filename*, a `PM_data` object in R6 or a `PMmatrix` data frame in Legacy containing a previously loaded .csv file (the output of `PMreadMatrix`) for errors which would cause the analysis to fail. If a model file is provided, and the data file has no errors, it will also check the model file for errors. If it finds errors, it will generate a new errors.xlsx file with all errors highlighted and commented so that you can find and correct them easily. See `?PMcheck` for details in R.

- R6 accepts data files with calendar dates and clock times by calling the Legacy function `PMmatrixRelTime()` any time `PM_data$new()` is invoked. This function converts dates and clock times of specified formats into relative times for use in the NPAG, IT2B and Simulator engines. This means Pmetrics in R6 does not require any action by the user to handle calendar dates and clock times. In contrast, for Legacy `PMmatrixRelTime()` must be called by the user and the output used to create a data frame with relative times that can be saves as a new .csv file with `PMwriteMatrix()`, which in turn serves as input to a run. See `?PMmatrixRelTime` for details.

The following functions are the same in either R6 or Legacy.

- `PMwrk2csv()` This function will convert old-style, single-drug USC*PACK .wrk formatted files into Pmetrics data .csv files. Details are available with `?PMwrk2csv` in R.

- `PMmb2csv()` This function will convert USC*PACK .mb files into Pmetrics data .csv files. Details are available with `?PMmb2csv` in R.

- `NM2PM()` Although the structure of Pmetrics data files are similar to NONMEM, there are some differences. This function attempts to automatically convert to Pmetrics format. It has been tested on several examples, but there are probably NONMEM files which will cause it to crash. Running `PMcheck()` afterwards is a good idea. Details can be found with `?NM2PM` in R.

5.2 Specifying Models in R6

R6

In R6 Pmetrics, use the `PM_model` function to create models directly in R. See `?PM_model` for help on this object class. Blocks in the legacy model.txt files which were delimited with the "#" character become listsin R6.

The R6 model components are:

- PRImary
- COVariate
- SECondary
- BOLus
- INItial conditions
- F (bioavailability)
- LAG time
- DIFferential equations
- OUTputs

5.2.1 PRImary variables

Primary variables are the model parameters that are to be estimated by Pmetrics or are designated as fixed parameters with user specified values. It should be a list of variable names, one name to a line. Variable names should be 11 characters or fewer. Some variable names are reserved for use by Pmetrics and cannot be used as primary variable names. **The number of primary variables must be between 2 and 32, with at most 30 random or 20 fixed.**

Each variable can be specified by `range` or `msd`. The first defines the absolute search space for that parameter for NPAG/NPOD. For IT2B/RPEM, it defines the mean of the piror as the midpoint of the range, and the range covers 6 standard deviations, e.g. ± 3 SD above and below the mean, or 99.7% of the piror distribution. `msd` is the companion function that specifies a mean and SD in IT2B and RPEM. For NPAG/NPOD, it will be converted in to a range in the reverse fashion as described for `range`. For both specifying functions, `gtz` is an argument to force the parameter value to be positive, i.e. `gtz=T`, which is the default. To allow negative parameters, set `gtz=F`.

```
mod <- PM_model$new(list(
  pri = list(
    Ke = range(0,5),
    V = msd(100,20),
    eff = range(-2,2,gtz=F)
  )
))
```

5.2.2 COVariates

Covariates are subject specific data, such as body weight, contained in the data .csv file. The covariate names, which are the column names in the data file, must be declared, even if not used in the model object. Once declared, they can be used in secondary variable and differential equations. The order and names should be the same as in the data file.

Covariates are applied at each dose event. The first dose event for each subject must have a value for every covariate in the data file.

Update By default, missing covariate values for subsequent dose events are linearly interpolated between existing values, or carried forward if the first value is the only non-missing entry.

To specify a new covariate value at a time other than a dose, enter a dose event in the data file with 0 dose amount and the new covariate value.

```
mod <- PM_model$new(list(
  pri = list(...),
  cov = list("wt","age")
))
```

5.2.3 SECondary variables

Secondary variables are those that are defined by equations that are combinations of primary, covariates, and other secondary variables. If using other secondary variables, define them first within this block. Equation syntax must be Fortran. Specify each variable equation as a character vector. It is permissible to have conditional statements, but because expressions in this block are translated into variable declarations in Fortran, expressions other than of the form ”X = function(Y)” must be on a new line, prefixed by ”&” and contain only variables which have been previously defined in the Primary, Covariate, or Secondary blocks.

In the example below, V0 is the primary parameter which will be estimated, but internally, the model uses V as V0*wt, unless age is >18, in which case weight is capped at 75 kg. It's the same for CL0. Note that the conditional statement is not named.

```
mod <- PM_model$new(list(
  pri = pri = list(
    CL0 = range(0,5),
    V0 = msd(10,3),
    eff = range(-2,2,gtz=F)
  ),
  cov = list("wt","age"),
  sec = list(
    V = "V0*wt",
    "&IF(age >18) V = V0 * 75",
    CL = "CL0 * wt"
  )
))
```

5.2.4 BOLus inputs

By default, inputs with DUR (duration) of 0 in the data .csv file are ”delivered” instantaneously to the model compartment equal to the input number, i.e. input 1 goes to compartment 1, input 2 goes to compartment 2, etc. This can be overridden with NBOLUS(input number) = compartment number.

```
mod <- PM_model$new(list(
  bol = list("NBCOMP(1) = 2")
))
```

5.2.5 INItial conditions

By default, all model compartments have zero amounts at time 0. This can be changed by specifying the compartment amount as X(.) = expression, where ”.” is the compartment number. Primary and secondary variables and covariates may be used in the expression, as can conditional statements in Fortran code. An ”&” continuation prefix is not necessary in this block for any statement, although if present, will be ignored.

```
mod <- PM_model$new(list(
  pri = pri = list(
    Ke = range(0,5),
    V = msd(100,30),
    IC3 = range(0,1000)
  ),
  cov = list("wt","age","IC2"),
  ini = list(
    X2 = "IC2*V",
    X3 = "IC3"
  )
))
```

In the example above, IC is a covariate with the measured trough concentration prior to an observed dose and IC3 is a fitted primary parameter specifying an initial amount in unobserved compartment 3.

In the first case, the initial condition for compartment 2 becomes the value of the IC covariate (defined in `cov` list) multiplied by the current estimate of V during each iteration. This is useful when a subject has been taking a drug as an outpatient, and comes in to the lab for PK sampling, with measurement of a concentration immediately prior to a witnessed dose, which is in turn followed by more sampling. In this case, IC or any other covariate can be set to the initial measured concentration, and if V is the volume of compartment 2, the initial condition (amount) in compartment 2 will now be set to the measured concentration of drug multiplied by the estimated volume for each iteration until convergence.

In the second case, the initial condition for compartment 3 becomes another variable, IC3 defined in the `pri` list, to fit in the model, given the observed data.

5.2.6 FA (bioavailability)

Specify the bioavailability term, if present. Use the form FA(.) = expression, where ”.” is the input number. Primary and secondary variables and covariates may be used in the expression, as can conditional statements in Fortran code. An ”&” continuation prefix is not necessary in this block for any statement, although if present, will be ignored.

```
mod <- PM_model$new(list(
  pri = pri = list(
    Ke = range(0,5),
    V = msd(100,30),
    FA1 = range(0,1)
  ),
  fa = list(
    fa1 = "FA1"
  )
)
```

5.2.7 LAG time

Specify the lag term, if present, which is the delay after an absorbed dose before observed concentrations. Use the form TLAG(.) = expression, where ”.” is the input number. Primary and secondary variables and covariates may be used in the expression, as can conditional statements in Fortran code. An ”&” continuation prefix is not necessary in this block for any statement, although if present, will be ignored.

```
mod <- PM_model$new(list(
  pri = pri = list(
    Ke = range(0,5),
    V = msd(100,30),
    lag1 = range(0,4)
  ),
  lag = list(tlag1 = "lag1")
  )
)
```

5.2.7.1 Differential equations

Specify a model in terms of ordinary differential equations, in Fortran format. XP(.) is the notation for dX(.)/dt, where ”.” is the compartment number. X(.) is the amount in the compartment. **There can be a maximum of 20 such equations.**

Specify equations as elements in a list, with XP(1) replaced by XP1, for example, to name the list, and the list value a character vector in Fortran.

```
mod <- PM_model$new(list(
  pri = pri = list(
    Ka = range(0,5),
    Ke = range(0,5),
    V = msd(100,30),
    Kcp = range(0,5),
    Kpc = range(0,5)
  ),
  diff = list(
    xp1 = "-Ka * X(1)",
    xp2 = "RATEIV(1) + Ka * X(1) - (Ke + Kcp) * X(2) + Kpc * X(3)",
    xp3 = "Kcp * X(2) - Kpc * X(3)"
  )
))
```

RATEIV(1) is the notation to indicate an infusion of input 1 (typically drug 1). The duration of the infusion and total dose is defined in the data.csv file.

Up to 7 inputs are currently allowed. These can be used in the model file as RATEIV(1), RATEIV(2), etc. The compartments for receiving the inputs of oral (bolus) doses are defined in the `bol` list, but can be accessed by using the B(1), B(2), etc notation in equations.

5.2.8 OUTputs

Output equations are in Fortran format. Outputs are of the form Y(.) = expression, where "." is the output equation number. Primary and secondary variables and covariates may be used in the expression, as can conditional statements in Fortran code. An "&" continuation prefix is not necessary in this block for any statement, although if present, will be ignored. **There can be a maximum of 6 outputs.**

They are referred to as Y(1), Y(2), etc. These equations may also define a model explicitly as a function of primary and secondary variables and covariates.

The `out` list is a series of nested lists. The outer list defines all the outputs. The next level defines each output equation. Within the equation list is the equation and the error model. Within the error model for that output, is the last list comprising model and assay error specifications.

- To name output equations, Y(1) is replaced by Y1

```
out = list(
  Y1 = list(...)
)
```

- The output equation is a character vector followed by an error list.

```
out = list(
  Y1 = list(
    "X(1)/V",
    err = list(...)
  )
)
```

- The error model for an output equation has two elements. The first is the `model` error, which can be one of three functions: `proportional`, `additive`, or `combination`. The arguments to these functions are a number and optionally `fixed`, which defaults to `FALSE`. If `fixed` is `FALSE`, the number serves as the starting estimate for the model error. If `fixed` is `TRUE`, the number serves as the model error, with no estimation. Note that you can only fix λ currently to zero.

The second element is the `assay` error model. It is a vector of 4 numbers that define a polynomial equation to permit calculation of the standard deviation of an observation, based on the noise of the asaay. The four terms estimate SD according to the folowing equation: $C0 + C1 * [obs] + C2 * [obs]^2 + C3 * [obs]^3$ and $[obs]$ is the observation. The values for the coefficients should ideally come from the analytic lab in the form of inter-run standard deviations or coefficients of variation at standard concentrations. You can use the Pmetrics function `makeErrorPoly` to choose the best set of coefficients that fit the data from the laboratory. Alternatively, if you have no information about the assay, you can use the Pmetrics ERRrun engine as an argument to the `run` function for `PM_fit` objects (i.e., `$run(engine="err")`) to estimate the coefficients from the data. Finally, you can use a generic set of coefficients. We recommend that as a start, $C0$ be set to half of the lowest concentration in the dataset and $C1$ be set to 0.1. $C2$ and $C3$ can be 0.

The proportional model weights each observation by $1/(\gamma * SD)^2$, where γ is either fixed or estimated. The additive model weights each observation by $1/(\lambda + SD)^2$, where λ is either fixed or estimated. The combination model uses $1/((\gamma * SD)^2 + (\lambda + SD)^2)$.

In the proportional model, γ is a scalar on assay SD. In general, well-designed and executed studies and models with low mis-specification will have data with γ values approaching 1. Values <1 suggest over inflated assay noise. Poor quality, noisy data will result in γ of 5 or more. A good starting value for γ is usually 5, and sometimes 10 if data are particularly complex or noisy.

In the additive model, λ is additive to assay SD. In general, well-designed and executed studies and models with low mis-specification will have data with λ values approaching 0. Values of 0 may suggest over inflated assay noise. Poor quality, noisy data will result in λ of $5 * C0$ or more. A good starting value for λ is usually $3 * C0$. Note, that $C0$ should generally not be 0, as it represents machine noise (e.g. HPLC or mass spectrometer) that is always present.

```r
out = list(
  Y1 = list(
    "X(1)/V",
    err = list(
      model = proportional(1),
      assay = c(0.15, 0.1, 0, 0)
    )
  )
)
```

```r
out = list(
  Y1 = list(
    "X(1)/V",
    err = list(
      model = additive(1, fixed = TRUE)
      assay = c(0.05, 0.1, 0, 0)
    )
)
```

More complete examples.

```
mod <- PM_model$new(list(
  pri = pri = list(
    Ke = range(0,5),
    V = msd(100,30),
  ),
  out = list(
    y1 = list(
      "X(1)/V",
      err = list(
        model = proportional(5),
        assay = c(0.05, 0.1, 0, 0)
      )
    )
  )
))

mod2 <- PM_model$new(list(
  pri = pri = list(
    kin = range(0,5),
    kout = range(0,5),
    tpd = range(0,5),
    V = msd(100,30),
  ),
  sec = list("RES = B(1) * KIN/(KIN-KOUT) * (EXP(-KOUT*TPD)-EXP(-KIN*TPD))"),
  out = list(
    y1 = list(
      "RES/V",
      err = list(
        model = combination(0.4,3) #additive, proportional
        assay = c(0.3, 0.15, 0, 0)
      )
    )
  )
))
```

This last example is known as the Bateman equation for a model with linear absorption (KIN) into and elimination (KOUT) from a central compartment, and a time post-dose (TPD) or lag time. Here B(1) is the oral bolus dosing vector for drug 1, and V is the volume of the central compartment.

5.3 Specifying Models in Legacy

Legacy

In legacy Pmetrics, models are text files that are ultimately translated into Fortran text files with a header version of TSMULT...

However, after Pmetrics version 0.30, we adopted a very simple user format that Pmetrics will use to generate the Fortran code automatically for you. Version 0.4 additionally eliminates the previously separate instruction file. A model library is available on our website at http://www.lapk.org/pmetrics.php.

Naming your model files. The default model file name is "model.txt," but you can call them whatever you wish. However, **please keep the number of characters in the model file name 8**. When you use a model file in NPrun(), ITrun(), ERRrun(), or SIMrun(), Pmetrics will make a Fortran model file of the same name, temporarily renaming your file. At the end of the run, your original model file will be in the /inputs subfolder of the run folder, and the generated Fortran model file will be called "model.for" and moved to the /etc subfolder of the run folder. If your model is called "mymodel.txt", then the Fortran file will be "mymodel.for".

You can still use appropriate Fortran model files directly, but we suggest you keep the .for extension for all Fortran files to avoid confusion with the new format. If you use a .for file as your model, you will have to specify its name explicitly in the NPrun(), ITrun, ERRrun(), or SIMrun() command, since the default model name again is "model.txt." If you use a .for file directly, it will be in the /inputs subfolder of the run folder, not in /etc, since you did not use the simpler template as your model file.

Structure of model files. The new model file is a text file with 11 blocks, each marked by "#" followed by a header tag. Only details which are different than the R6 documentation are included below.

- #PRImary variables
- #COVariates
- #SECcondary variables
- #BOLus inputs
- #INItial conditions
- #Fa (bioavailability)
- #LAG time
- #DIFferential equations
- #OUTputs

- #ERRor
- #EXTra

For each header, only the capital letters are required for recognition by Pmetrics. The blocks can be in any order, and header names are case-insensitive (i.e. the capitalization here is just to show which letters are required). Fortran is also case-insensitive, so in variable names and expressions case is ignored. Details of each block are next, followed by a complete example.

Important: Sometimes it is important to preserve spacing and formatting in Fortran code that you might insert into blocks, particularly the #EXTRA block. If you wish to do this, insert [format] and [/format] before and after any code that you wish to reproduce verbatim with spacing in the fortran model file.

Comments: You can insert comments into your model text file by starting a line with a capital "C" followed by a space. These lines will be removed/ignored in the final fortran code.

5.3.1 Primary variables

Primary variables are the model parameters that are to be estimated by Pmetrics or are designated as fixed parameters with user specified values. It should be a list of variable names, one name to a line. Variable names should be 11 characters or fewer. Some variable names are reserved for use by Pmetrics and cannot be used as primary variable names. **The number of primary variables must be between 2 and 32, with at most 30 random or 20 fixed.**

On each row, following the variable name, include the range for the parameter that defines the search space. These ranges behave slightly differently for NPAG, IT2B, and the simulator.

- For all engines, the format of the limits is *min, max*. A single value will indicate that the parameter is to be fixed but unknown in the population, i.e. the value is taken as the starting point for the optimization, but the final value will depend on the model and data and will be the same across the population. A single value followed by an "!" will indicate that this value is to be held constant (i.e. fixed and known) across the population, and not to be estimated.

- For **NPAG**, when *min/max* limits are specified, they are absolute, i.e. the algorithm will not search outside this range.

- For **IT2B**, the range defines the Bayesian prior distribution of the parameter values for cycle 1. For each parameter, the mean of the Bayesian prior distribution is taken as the middle of the range, and the standard deviation is *xsig**range (see IT2B runs). Adding a plus sign (+) to a line will prevent that parameter from being assigned negative values. NPAG and the simulator will ignore the pluses as the ranges are absolute for these engines. Note that prior to version 1.5.0, this used to be an exclamation point (!) but to be consistent throughout the model file, the exclamation point is now used when fixed values are desired.

- The **simulator** will ignore the ranges with the default value of NULL for the *limits* argument. If the simulator *limits* argument is set to NA, which will mean that these ranges will be used as the limits to truncate the simulation (see Simulator Runs).

Example:

```
#Pri
KE, +0, 5
V, 0.01, 100
KA, 0, 5
KCP, 5
KPC, 0 , 5
Tlag1, 0, 2
IC3, 0, 10000
FA1, 0.5!
```

5.3.2 Covariates

By default, missing covariate values for subsequent dose events are linearly interpolated between existing values, or carried forward if the first value is the only non-missing entry. To suppress interpolation and carry forward the previous value in a piece-wise constant fashion, include an exclamation point (!) in any declaration line.

Note that any covariate relationship to any parameter may be described as the user wishes by mathematical equations and Fortran code, allowing for exploration of complex, non-linear, time-dependent, and/or conditional relationships. This is accomplished in the #Sec block.

Example:

```
#Cov
wt
cyp
IC!
```

where IC will be piece-wise constant and the other two will be linearly interpolated for missing values.

5.3.3 Secondary variables

Equation syntax must be Fortran. It is permissible to have conditional statements, but because expressions in this block are translated into variable declarations in Fortran, expressions other than of the form ”X = function(Y)” must be prefixed by a ”&” and contain only variables which have been previously defined in the Primary, Covariate, or Secondary blocks. Note that prior to version 1.5.0, the continuation symbol was “+” before each line, but to avoid confusion with the use of “+” in the Primary block for IT2B models, and to be consistent with Fortran notation, the “&” is used henceforth.

Example:

```
#Sec
CL = Ke * V * wt**0.75
& IF(cyp .GT. 1) CL = CL * cyp
```

5.3.4 Bolus inputs

Example:

```
#Bol
NBCOMP(1) = 2
```

5.3.5 Initial conditions

Example:

```
#Ini
X(2) = IC*V
X(3) = IC3
```

In the first case, the initial condition for compartment 2 becomes the value of the IC covariate (defined in #Covariate block) multiplied by the current estimate of V during each iteration. This is useful when a subject has been taking a drug as an outpatient, and comes in to the lab for PK sampling, with measurement of a concentration immediately prior to a witnessed dose, which is in turn followed by more sampling. In this case, IC or any other covariate can be set to the initial measured concentration, and if V is the volume of compartment 2, the initial condition (amount) in compartment 2 will now be set to the measured concentration of drug multiplied by the estimated volume for each iteration until convergence.

In the second case, the initial condition for compartment 3 becomes another variable, IC3 defined in the #Primary block, to fit in the model, given the observed data.

5.3.6 Fa (bioavailability)

Specify the bioavailability term, if present. Use the form FA(.) = expression, where "." is the input number. Primary and secondary variables and covariates may be used in the expression, as can conditional statements in Fortran code. An "&" continuation prefix is not necessary in this block for any statement, although if present, will be ignored.

Example:

```
#Fa
FA(1) = FA1
```

5.3.7 Lag time

Example:

```
#Lag
TLAG(1) = Tlag1
```

5.3.8 Differential equations

Example:

```
#Dif
XP(1) = -KA*X(1)
XP(2) = RATEIV(1) + KA*X(1) - (KE+KCP)*X(2) + KPC*X(3)
XP(3) = KCP*X(2) - KPC*X(3)
```

5.3.9 Outputs

Output equations, in Fortran format. Outputs are of the form Y(.) = expression, where "." is the output equation number. Primary and secondary variables and covariates may be used in the expression, as can conditional statements in Fortran code. An "&" continuation prefix is not necessary in this block for any statement, although if present, will be ignored. **There can be a maximum of 6 outputs.** They are referred to as Y(1), Y(2), etc. These equations may also define a model explicitly as a function of primary and secondary variables and covariates.

Examples:

```
#Out
Y(1) = X(2)/V
```

```
#OUT
```

RES = B(1) * KIN/(KIN-KOUT) * (EXP(-KOUT*TPD)-EXP(-KIN*TPD))

Y(1) = RES/VD

5.3.10 Error

Unlike the R6, the error block is separate from the output block. In Legacy Pmetrics, this block contains all the information Pmetrics requires for the structure of the error model.

To specify the model in this block, the first line needs to be either L=number or G=number for a λ or γ error model. The number is the starting value for λ or γ. If you include an exclamation point (!) in the declaration, then λ or γ will be fixed and not estimated. Recall that you can only fix λ currently to zero.

The next line(s) contain the values for $C0$, $C1$, $C2$, and $C3$, separated by commas. There should be one line of coefficients for each output equation. By default Pmetrics will use values for these coefficients found in the data file. If none are present or if the model declaration line contains an exclamation point (!) the values here will be used.

Example 1: estimated λ, starting at 0.4, one output, use data file coefficients but if missing, use 0.1,0.1,0,0.

```
#Err
L=0.4
0.1,0.1,0,0
```

Example 2: fixed γ of 2, two outputs, use data file coefficients but if missing, use 0.1,0.1,0,0 for the first output, but use 0.3, 0.1, 0, 0 for output 2 regardless of what is in the data file.

```
#Err
G=2!
0.1,0.1,0,0
0.3,0.1,0,0!
```

5.3.11 Extra

This block is for advanced Fortran programmers only. <span class=“update>UpdateIt is not yet implemented in R6. Occasionally, for very complex models, additional Fortran subroutines are required. They can be placed here. The code must specify complete Fortran subroutines which can be called from other blocks with appropriate call functions. As stated earlier, sometimes it is important to preserve spacing and formatting in Fortran code that you might insert into blocks, particularly the #EXTRA block. If you wish to do this, insert [format] and [/format] in the fortran model file around the affected code.

5.3.12 Reserved Names

The following cannot be used as primary, covariate, or secondary variable names. They can be used in equations, however.

Reserved Variable	Function in Pmetrics
ndim	internal
t	time
x	array of compartment amounts
xp	array of first derivative of compartment amounts
rpar	internal
ipar	internal
p	array of primary parameters
r	input rates
b	input boluses
npl	internal
numeqt	output equation number
ndrug	input number
nadd	covariate number
rateiv	intravenous input for inputs when DUR>0 in data files
cv	covariate values array
n	number of compartments
nd	internal
ni	internal
nup	internal
nuic	internal
np	number of primary parameters
nbcomp	bolus compartment array
psym	names of primary parameters
fa	biovailability
tlag	lag time
tin	internal
tout	internal

5.3.13 Complete Example

Here is a complete example of a model file, as of Pmetrics version 0.40 and higher:

```
#Pri
KE, 0, 5
```

```
V0, 0.1, 100
KA, 0, 5
Tlag1, 0, 3


#Cov
wt
C this weight is in kg


#Sec
V = V0*wt


#Lag
TLAG(1) = Tlag1


#Out
Y(1) = X(2)/V


#Err
L=0.4
0.1,0.1,0,0
```

Notes:

By omitting a #Diffeq block with ODEs, Pmetrics understands that you are specifying the model to be solved algebraically. In this case, at least KE and V must be in the Primary or Secondary variables. KA, KCP, and KPC are optional and specify absorption, and transfer to and from the central to a peripheral compartment, respectively.

The comment line "C this weight is in kg" will be ignored.

5.3.14 Brief Fortran Tutorial

Much more detailed help is available from http://www.cs.mtu.edu//~shene/COURSES/cs201/NOTES/fortran.html.

Arithmetic Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Relational Operator	**Alternative**	**Meaning**
<	.LT.	less than
<=	.LE.	less than or equal
>	.GT.	greater than
>=	.GE.	greater than or equal
==	.EQ.	equal
/=	.NE.	not equal

Selective Execution	**Example**
IF (logical-expression) one-statement	IF (T >= 100) CL = 10
IF (logical-expression) THEN	IF (T >= 100) THEN
statements	CL = 10
END IF	V = 10
	END IF
IF (logical-expression) THEN	IF (T >= 100) THEN
statements-1	CL = 10
ELSE	ELSE
statements-2	CL = CL
END IF	END IF

Chapter 6

How to use R and Pmetrics

In this section, we suggest a workflow to help you maintain organized modeling projects.

6.1 Setting up a Pmetrics project

When beginning a new modeling project, it is convenient to use the command `PMtree`. This command will set up a new directory in the current working directory named whatever you have included as the "project name".

```
PMtree("DrugX")
```

In the above example, a directory called "DrugX" will be created in the current working directory in R, which you can check with the `getwd` function. Beneath the new DrugX directory, several subdirectories will be also created.

- **Rscript** contains a skeleton R script to begin Pmetrics runs in the new project.
- **Runs** should contain all files required for a run (described next) and it will also contain the resulting numerically ordered run directories created after each Pmetrics NPAG or IT2B run.
- **Sim** can contain any files related to simulations
- **src** is a repository for original and manipulated source data files

You are free to edit this directory tree structure as you please, or make your own entirely.

6.2 Getting the required inputs to run Pmetrics

6.2.1 R6

R6

When you wish to execute a Pmetrics run, you must ensure that an appropriate Pmetrics data.csv file is in the working directory, i.e. the Runs subdirectory of the project directory. R can be used to help prepare the data.csv file by importing and manipulating spreadsheets (e.g. `read.csv`). The Pmetrics function `PMcheck` can be used to check a .csv file or an R dataframe that is to be saved as a Pmetrics data.csv file for errors. It can also check a model file for errors in the context of a datafile, e.g. covariates that do not match. `PMcheck(...,fix=T)` attempts to automatically rid data files of errors. The function `PMwriteMatrix` can be used to write the R data object in the correct format for use by IT2B, NPAG, or the Simulator.

We have discussed the creation of model objects with `PM_model`.

To bring these together, use the `PM_fit` object creator. It only needs two arguments: the name of the data file in the working directory or Updatein memory loaded via `PMreadMatrix`and a model object. `PM_fit` will accept a model object created by `PM_model` or the name of a model file in Legacy format and in the working directory.

```r
#Example 1 - data file and PM_model object
fit1 <- PM_fit$new(data = "data.csv", model = mod1)

#Example 2 - data object and model file
PMdata <- PMreadMatrix("data.csv")
fit2 <- PM_fit$new(data = PMdata, model = "model.txt")

#Example 3 - data and model objects
fit3 <- PM_fit$new(PMdata, mod1)
```

Once the `PM_fit` object is created it has only one function defined for it: `$run()`. Arguments for this function can be found in the help for `PM_fit` and later in this manual.

```r
#default run parameters
fit3$run()

#change the cycle number from default 100
fit3$run(cycles=500)

#change the engine from default NPAG
fit3$run(engine = "IT2B")
```

See the sections on running NPAG and IT2B later in the manual for more details on arguments available to modify run behavior.

6.2.2 Legacy

Legacy

When you wish to execute a Pmetrics run, you must ensure that both of the appropriate Pmetrics data.csv and model.txt files are in the working directory, i.e. the Runs subdirectory of the project directory. The names are supplied as arguments to `NPrun`, `ITrun`, and `ERRrun`. A shorthand notation is to supply the number of a previous run for either the data, model or both files so that you do not have to manually copy them into the working directory.

```
#Using default names data.csv and model.txt
NPrun()

#Using custom names
ITrun(model = "model1.txt", data = "mydata.csv")

#Grab data from run 1 and use default model.txt
NPrun(data=1)

#Use model and data from run 2 and continue where run 2 ended
NPrun(data=2, model=2, prior=2, cycles=1000)
```

See the sections on runing NPAG and IT2B later in the manual for more details on arguments available to modify run behavior.

You can also download sample data and scripts from the Pmetrics downloads section of our website. Edit prior versions of model files to make new model files.

6.3 Using scripts to control Pmetrics

As you will see in the skeleton R script made by `PMtree` and placed in the Rscript subdirectory, if this is a first-time run, the R commands to run IT2B or NPAG are as follows. Recall that the "#" character is a comment character.

```
library(Pmetrics)
#Run 1 - add your run description here
setwd("working directory")
NPrun() #for NPAG or ITrun() for IT2B
```

The first line will load the Pmetrics library of functions. The second line sets the working directory to the specified path. The third line generates the batch file to run NPAG or IT2B and saves it to the working directory.

NOTE: On Mac systems, the batch file will be automatically launched in a Terminal window. On Windows systems prior to version 1.9, the batch file must be launched manually by double clicking the *npscript.bat* or *itscript.bat* file in the working directory. As of version 1.9, Windows users no longer need to do this.

`ITrun` and `NPrun` are described in full detail via their help commands in R and later in this manual. At minimum, they require a data file and a model file. If the default names of "data.csv" and "model.txt" are used, they may be called with no arguments. Again, the data and model files must be in the current working directory, usually the Runs folder.

Both functions return the full path of the output directory to the clipboard. By default, runs are placed in folders numbered sequentially, beginning with "1".

6.4 Loading results after a completed run

Now the output of IT2B or NPAG needs to be loaded into R, so the next command does this.

```
PMload(run_number)
```

Details of these commands and what is loaded are described in the R documentation (?PMload) and in the following section. The run_number should be included within the parentheses to be appended to the names of loaded R objects, allowing for comparison between runs, e.g. `PMload(1)`. Finally, at this point other Pmetrics commands can be added to the script to process the data, such as the following.

plot(final.1)
plot(cycle.1)
plot(op.1,type="pop") or plot(op.1$pop1)
plot(op.1) #default is to plot posterior predictions for output 1
plot(op.1,type="pop",resid=T)

Of course, the full power of R can be used in scripts to analyze data, but these simple statements serve as examples.

If you do not use the `PMtree` structure, we suggest that the R script for a particular project be saved into a folder called "Rscript" or some other meaningful name in the working directory. Folders are not be moved by the batch file. Within the script, number runs sequentially and use comments liberally to distinguish runs, as shown below.

```
library(Pmetrics)
#Run 1 - Ka, Kel, V, all subjects
setwd(“working directory”)
NPrun() #assumes model=“model.txt” and data=“data.csv”
PMload(1) ...
```

Remember in R that the command `example(function)` will provide examples for the specified function. Most Pmetrics functions have examples.

Chapter 7

Pmetrics Data Objects

After a successful IT2B or NPAG run, an R datafile is saved in the output subdirectory of the newly created numerically ordered folder in the working directory. After IT2B, this file is called “IT2Bout.Rdata”, and after NPAG it is called “NPAGout.Rdata”. As mentioned in the previous section, these data files can be loaded by ensuring that the Runs folder is set as the working directory, and then using the Pmetrics commands PMload(run_num).

There are several Pmetrics data objects contained within the Rdata files which are loaded with PMload(), making these objects available for plotting and other analysis.

Objects loaded by PMload(run_num)

Objects	Variables
op (class: PMop, list)	\$id
	\$time
	\$obs
	\$pred
	\$pred.type
	\$icen
	\$outeq
	\$block
	\$obsSD
	\$d
	\$ds
	\$wd
	\$wds
final (class: PMfinal, list)	\$popPoints
	\$popMean
	\$popSD
	\$popCV
	\$popVar
	\$popCov
	\$popCor
	\$popMedian
	\$popRanFix
	\$postPoints
	\$postMean
	\$postSD
	\$postVar
	\$postCov
	\$postCor
	\$gridpts
	\$ab
cycle (class: PMcycle, list)	\$names
	\$ll
	\$gamlam
	\$mean
	\$sd
	\$median
	\$aic
	\$bic
cov (class: PMcov, data.frame)	\$id
	\$time
	covariates...
	parameters...
	\$icen
pop (class: PMpop, data.frame)	\$id
post (class: PMpost, data.frame)	
NPAG only	

Since R is an object oriented language, to access the observations in a **PMop** object, for example, use the following syntax: op$post1$obs.

Note that you will place an integer corresponding to the run number within the parentheses of the loading functions, e.g. PMload(1), which will suffix all the above objects with that integer, e.g. op.1, final.1, NPdata.1. This allows several models to be loaded into R simultaneously, each with a unique suffix, and which can be compared with the PMcompare() command (see Model Diagnostics below).