ggPMX

Diagnostic Plots R-Package

Cheat Sheet version 1.2

Monolix Requirements

- Monolix versions 2016 and later
- Requiring to run the following Monolix tasks: POPULATION PARAMETERS, EBEs, STANDARD ERRORS and PLOTS
- Make sure to export charts data (in Monolix: Settings -> Preferences -> Export -> switch on the **Export charts data** button)
- Select at least the following plots to be displayed and saved: **individual fits** and **scatter plot of the residuals**

NONMEM Requirements

- NONMEM Version 7.2/7.3/7.4
- Preferred output tables according "sdtab, patab, cotab, catab" convention
- Simulation are required for creation of VPC (e.g. sdtab1sim)

nlmixr Requirements

- Fit objects need to be generated by nlmixr and have data attached.
- Standard errors are required (a successful covariance plot) for full diagnostic checks. Can use `boostrapFit()` to get standard error estimates if necessary

Controller Creation for Monolix

1. Define the location of your working directory
```
theophylline_path = file.path(system.file(
      package = 'ggPMX'), 'testdata', 'theophylline')
work_dir = file.path(theophylline_path, 'Monolix')
```
2. Define the location of your modeling dataset
```
input_data = file.path(theophylline_path, 'data_pk.csv')
```
3. Create controller using function **pmx_mlx()**
```
ctr = pmx_mlx(directory = work_dir, input = input_data,
              dv = 'Y', cats=c('SEX'),conts=c('AGE0','WT0'))
```

- Automatic parsing of .mlxtran files:
```
ctr = pmx_mlxtran(file_name, call, endpoint)
file_name: "path to .mlxtran file" (Monolix project)
```

Controller Creation for NONMEM

1. Define the location of your working directory
```
work_dir = file.path(system.file(
      package = 'ggPMX'), 'testdata', extdata')
```
2. Define the location of your modeling file
```
model_file = 'run001.lst'
```
3. Create controller using function **pmx_nm()** and model_file (e.g. .lst)
```
ctr = pmx_nm(directory = work_dir, file = model_file)
```
Using the run-number:
```
ctr = pmx_nm(directory = work_dir, runno = '001')
```
Using individual tables:
```
ctr = pmx_nm(directory = work_dir, table_names = 'sdtab')
```

Controller Creation for nlmixr

1. Generate a fit object with the nlmixr package
```
Check out github.com/nlmixrdevelopment/nlmixr
```
fit <- nlmixr(modelFunction, modelData, est="saem") # typically
2. Define the location of your modeling file
```
ctr = pmx_nlmixr(fit = fit, dv = 'Y', dvid = 'DVID',
   cats=c('SEX'),conts=c('AGE0','WT0'), vpc = TRUE)
```
Note: VPC takes time and does not need to be created if you do not plan to use the VPC plots. In that case, set *vpc* argument to FALSE.

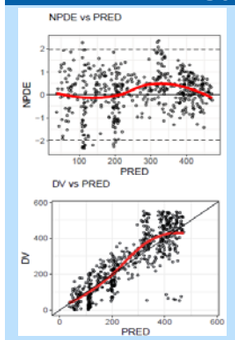
Key Features

Library of **diagnostic plots** for nonlinear mixed-effects model building and evaluation; **Consistent**, **reproducible** and **efficient** workflow; High quality graphics, ready-to-use in **submission documents and publications**; Advanced Features for **customization** and **stratification**; Automated generation of PDF, Word or PNG outputs for reporting purpose; **Controller is the user interface object of ggPMX**, it contains all underlying information to generate diagnostics plot

Default Plots

- Default plots visualized using the functions **pmx_plot_xx()**, where xx is a placeholder for the plot name
- Use of %>% operator on controller to generate diagnostics plot `ctr %>% pmx_plot_npde_pred()`
- List of plot names **xx** available by printing the controller

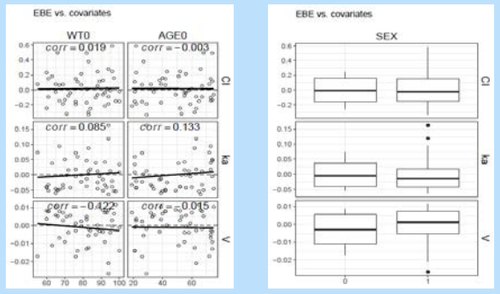
Scatter plots




```
pmx_plot_npde_pred()
pmx_plot_npde_time()
pmx_plot_iwres_ipred()
pmx_plot_iwres_time()
pmx_plot_abs_iwres_ipred()
pmx_plot_dv_pred()
pmx_plot_dv_ipred()
```

Covariate plots

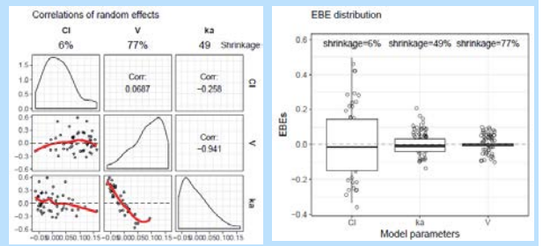
`pmx_plot_eta_conts()` `pmx_plot_eta_cats()`



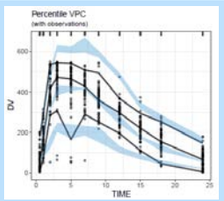
EBE-based plots[1]

[1]EBE: Empirical Bayes estimate

`pmx_plot_eta_matrix()` `pmx_plot_eta_box()`

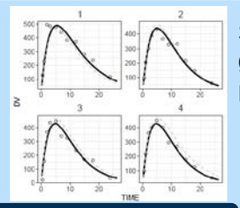


VPC



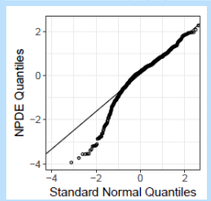

```
pmx_plot_vpc()

[Arguments]
ctr, type, idv, obs, pi, ci,
rug, bin, is.legend, dname
```

Individual plots



Several pages of plots are generated:
```
pmx_plot_individual(
  npage = 1,
  facets=list(
    nrow=3, ncol=2))
```

QQ-plots and density plots[2]




```
pmx_plot_eta_hist()
pmx_plot_eta_qq()
pmx_plot_npde_qq()
pmx_plot_iwres_qq()
pmx_plot_iwres_dens()
```

Automated Diagnostics Report

`pmx_report()` - [Arguments]: *ctr, name, save_dir, format, template, footnote, edit, title,...*

Three types of format (*format argument)* :

- "**report**": produces *name.pdf* and *name.docx* reports, located in *save_dir* with default diagnostic plots
- "**plots**": produces a folder named *ggpmx_GOF* located in *save_dir* containing all default diagnostic plots (pdf and .png format)
- "**both**": is a combination of both options above

Report Customization: Create Custom Template

1. Start from a generated R Markdown (.Rmd):
```
ctr %>% pmx_report(name = "DiagnosticPlots1",
                   save_dir = work_dir, format = "both")
```
2. Modify resulting R Markdown (*Diagnostic_plots1.Rmd)* as desired (e.g. change the size, settings of some figures) into new template
3. Create a report using the customized template:
```
ctr %>% pmx_report(name = "DiagnosticPlots2", save_dir =
           work_dir, format = "report", template =
           file.path(work_dir, "DiagnosticPlots2.Rmd"))
```

Visual Predictive Check (VPC) for NONMEM/nlmixr users

NONMEM: Simulation tables are required for creation of VPC (e.g. sdtab1sim). They are loaded automatically if detected and the sim object is created automatically. For post-hoc simulation, you can also specify:

```
ctr = pmx_nm(directory = work_dir, file = model_file,
             simfile = "my_simulation.ctl")
```

nlmixr: VPC will be autmatically enabled with controller creation

```
ctr = pmx_nlmixr(fit = fit, vpc = TRUE)
```

Multiple-endpoint models (PKPD models)

- ggPMX produces one set of diagnostics plots per endpoint
- Select the endpoint (if more than one) at the level of the Controller -> create ***endpoint*** object with *pmx_endpoint()* and pass it to Controller.

Only for Monolix :

- Endpoint index (*code*) in modeling dataset (usually dvid) should be mapped to Monolix output index X (*file.code*), i.e. predictionsX.txt and finegridX.txt, where X=1,2,...

```
ep  = pmx_endpoint( code = "4", label = "some_label",
    unit = "some_unit", file.code = "2", trans="log10")

ctr = pmx_mlx(..., dvid = "YTYPE", endpoint = ep)
```

For Monolix,NONMEM and nlmixr :

- If *code* and *file.code* index match, use simplified syntax (no endpoint object): `ctr = pmx_* (..., dvid = "YTYPE", endpoint = 1)`

**can be pmx_mlx/pmx_nm or pmx_nlmixr*

Visual Predictive Check (VPC) for Monolix users

- To produce the VPC, a simulation file is required (e.g. *sim.csv*), with (at least) the following columns: ID (individual identifiers), REP (simulation replicate number), TIME, DV (dependent variable)
- Create a simulation object to use as argument in Controller creation:
```
sim = pmx_sim(file = "sim.csv", irun ="rep", idv="TIME")
```
- Change bins:
```
pmx_plot_vpc(bin = pmx_vpc_bin(style="equal", n=10))
```
- Change prediction and confidence intervals, data points properties:
```
pmx_plot_vpc(type = "percentile",
      pi  = pmx_vpc_pi(interval = c(0.1,0.9),
      ci  = pmx_vpc_ci(interval = c(0.1,0.9),
      obs = pmx_vpc_obs(color="blue", shape=18, size=2))
```


NOVARTIS

Plot Customization

Any plot can be customized in two ways:

1. Individually, by specifying options:

```
pmx_plot_npde_time(ctr, smooth = list(color = "blue"),
   point  = list(shape = 4), is.draft = TRUE,
   labels = list(x = "TIME after first dose (days)",
                 y = "Normalized PDE"))
```

2. Globally, at the level of Controller creation, using `pmx_settings`, e.g. adding a DRAFT label to all plots, modify labels using *cats.labels*:

```
mySet = pmx_settings(is.draft=TRUE, use.labels=TRUE,
     cats.labels=list(SEX=c("0"="Male","1"="Female")))
ctr = pmx_mlx(config = "standing",...,settings = mySet)
```

Stratification and Filtering

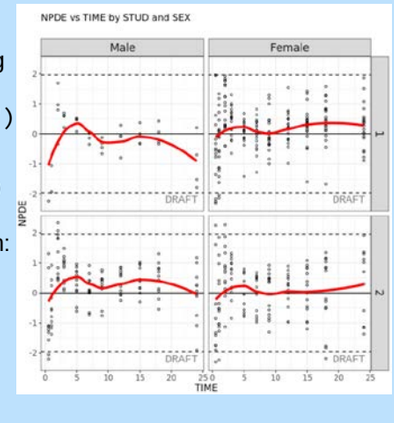
Two ways of **stratifying** a plot:

1. using the stratification function by categorical covariates and providing the plot name in argument:

```
pmx_plot_cats(ctr, "npde_time")
```

2. using arguments *strat.facet* (discrete) or *strat.color* (continuous) in *pmx_plot_xx()*.
   Allows two-dimentional stratification:

```
pmx_plot_npde_time(ctr,
       strat.facet = STUD~SEX)
```



Two ways of **filtering** the data:

1. Locally, at the level of a plot:

```
ctr %>% pmx_plot_dv_pred(
          filter = DV > mean(DV) & PRED < median(PRED))
```

2. Globally, at the level of the Controller, affecting all subsequent plots:

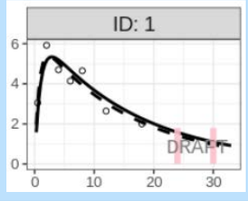
```
ctr %>% pmx_filter(
       data_set = "prediction",
       ID == 5 & TIME < 2)
```

Visualization of Censored data

Create controller with BLOQ (below limit of quantification) data (BLQ column in input dataset):

```
ctr %>% pmx_mlxtran(file_name = mlx_file,
                    bloq = pmx_bloq(cens = "BLQ",
                    limit="LIMIT"))

ctr %>% pmx_plot_individual()
```

The above applies also to ULOQ (upper limit of quantification, or right-censored data)



Parameter Table

Create a table with parameter estimates, either use:

```
param_tab <- ctr %>% get_data("estimates")
```

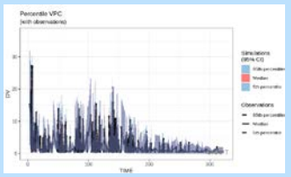
Or use:

```
param_tab <- ctr %>% param_table()
```

VPC Plot Customization Help

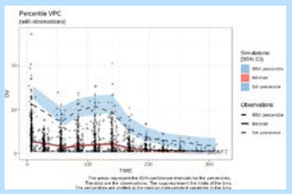
Sometimes, calling the standard vpc function will not result in a usable vpc-plot.

```
ctr %>% pmx_plot_vpc()
```



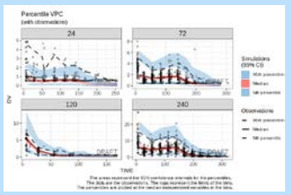
Using different style of binning might result in better plots. Many styles are available: "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust" or "jenks".

```
ctr %>% pmx_plot_vpc(bin=pmx_vpc_bin(style = "equal"))
```



Sometimes, it is helpful to stratify according to a covariate. Furthermore, it is possible to set the scales on the y- and x-axis free.

```
ctr %>% pmx_plot_vpc(bin=pmx_vpc_bin(style = "equal"),
strat.facet = "DOSE", facets = list(scales = "free"))
```



There are more options to set e.g. scale_y_log10. Use ?pmx_plot_vpc for further help.

Simulation with simulx for Monolix

Create simulated object using simulx

```
mysim <- simulx(project=myprojfilename, nrep=100)
```

Retrieve simulated dataset (assumed to be in variable y1):

```
simdata <- mysim$y1
```

Need to revert the original IDs as in modeling dataset for ggPMX. Rename ID column to same name as in modeling dataset, e.g. "id" in the example below:

```
simdata <- simdata %>%
           left_join(., mysim$originalId) %>%
           mutate(id = oriId) %>%
           select(-oriId, -newId)
```

Save the simulated dataset in a csv file:

```
mysimfilename <- "my_VPC.csv"
write.csv(simdata, file = mysimfilename, quote = FALSE,
row.names = FALSE)
```

Creation of an independent copy of the controller

The controller is an 'R6' object, it behaves like a reference object. Some functions (methods) can have a side effect on the controller and modify it internally. Technically speaking we talk about chaining not piping here. However , using pmx_copy user can work on a copy of the controller.

```
ctr_copy = pmx_copy(ctr, keep_globals=FALSE)
```

Labeling of axes

There are different ways to modify the labels of axes in ggPMX:

Modifiying labels globally

Using customized labels for all plots by modifiyng controller

```
ctr %>% set_abbrev(
            IPRED = "Individual Prediction (ug/ml)",
            PRED  = "Population Prediction (ug/ml)",
            DV    = "Observed Drug Concentration (ug/ml)")
```

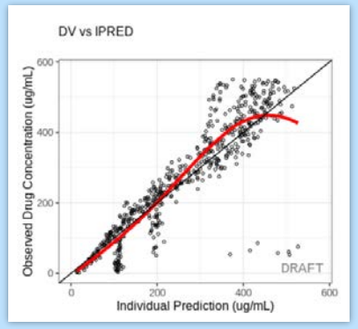
Changes can be applied using the usper-spcified label names for all plots by modifying the controller, use.abbrev set to TRUE by default for all ggPMX plots

```
ctr %>% pmx_mlxtran(file_name = mlxtran_path,
            settings  = pmx_settings(use.abbrev = TRUE))

# use.abbrev = TRUE is now default
ctr %>% pmx_plot_dv_ipred()
```

Modifiying labels locally

Individual specification of label-names within the plot:

```
ctr %>% pmx_plot_dv_ipred(
            labels = list(
            x      = "Individual Prediction (ug/ml)",
            y      = "Observed Drug Concentration (ug/ml)"))
```



Using simulated censored data (Monolix)

If you want to use simulated BLOQ (below limit of quantification) in ggPMX you can specify sim_blq within the controller.

Using sim_blq at the controller level:

```
ctr = pmx_mlx(directory = work_dir, input = input_data,
              sim_blq = TRUE)

ctr %>% pmx_plot_iwres_time() #will plot simulated BLOQs
ctr %>% pmx_plot_iwres_time(sim_blq = FALSE) # will plot LOQs
```

If a default controller is created, sim_blq = TRUE can also be used at the individual plot level:

```
ctr = pmx_mlx(directory = work_dir, input = input_data)

# Will plot simulated BLOQs:
ctr %>% pmx_plot_iwres_time(sim_blq = TRUE)
```


Authors: Irina Baltcheva, Christian Bartels, Souvik Bhattacharya, Seid Hamzic, Thomas Dumortier, Didier Renard, Amine Gassem, Bruno Bieth

Resource: https://github.com/ggPMXdevelopment/ggPMX

NOVARTIS