Session 5: xpose

Ashwin Karanam

Contents

1 The xpose package

Xpose is an R-based model building aid for population analysis using NONMEM. It facilitates data set checkout, exploration and visualization, model diagnostics, candidate covariate identification and model comparison created by Andrew Hooker, Mats O. Karlsson, Benjamin Guiastrennec and E. Niclas Jonsson from Uppsala University.

1.1 Install package

```
# Install the lastest release from the CRAN
install.packages('xpose')

# Or install the development version from GitHub
# install.packages('devtools')
devtools::install_github('UUPharmacometrics/xpose')
```

1.2 xpose requirements

To make full use of the functionality offered by `xpose` the following NONMEM output files should be available:

- `.lst/.out/.res`: used to collect information on the run (`template_titles`) as well as the output table names. Alternatively a model file (`.mod/.ctl`) can be used but some of the information in `template_titles` may not be available.
- `.ext`: used to collect final parameter estimates and residual standard error (RSE)
- `.phi`: used for the random effects and iOFV
- `.cov`: used for the covariance matrix
- `.cor`: used for the correlation matrix
- `.grd`: used for the estimation gradients
- `.shk`: used to compute random effect shrinkage `template_titles`
- output and simulation tables: for the actual data

When importing the files, xpose will return messages to the console and inform of any issue encountered during the import.

`xpose` is compatible with the `$TABLE FIRSTONLY` option of NONMEM. The option `FIRSTONLY` only output the first record for each `ID` and hence can be used to decrease the size of output tables having no time-varying columns. During tables import xpose will merge `FIRSTONLY` tables with regular tables allowing seamless use of columns from `FIRSTONLY` in plots.

1.3 Create xpose database

```
library(xpose)
library(tidyverse)
library(gridExtra)

#xpdb_ex_pk is an inbuilt example from xpose.
# Look at the ~/Documents/R/win-library/3.4/xpose/extdata
xpdb <- xpdb_ex_pk
```

If your run number is 001, all your NONMEM output files will end in 001. Create xpose database using this:

```
xpdb <- xpose_data(runno = '001')

Looking for nonmem output tables
Reading: sdtab001, catab001, cotab001, patab001 [$prob no.1]

Looking for nonmem output files
Reading: run001.cor, run001.cov, run001.ext, run001.grd, run001.phi
```

These messages can be silenced with the option `quiet = TRUE`.

2 xpose xpdb data access

A typical `xpdb` object contains 8 levels namely:

- `code`: the parsed model code
- `summary`: contains key information regarding the model. All the information contained in the summary can be used as part of the `template_titles`.
- `data`: contains all output and simulation tables as well as the column indexing
- `files`: contains all output files
- `special`: contains post-processed datasets used by functions like `vpc()`
- `gg_theme`: an attached ggplot2 theme
- `xp_theme`: an attached xpose theme
- `options`: attached global options

2.1 Glimpse at the xpdb

The files attached to an xpdb object can be displayed to the console simply by writing the xpdb name to the console or by using the `print()` function. Any of these files can be accessed from the xpdb using one of the functions listed below.

```
xpdb # or print(xpdb)
```

```
## run001.lst overview:
##  - Software: nonmem 7.3.0
##  - Attached files (memory usage 1.3 Mb):
##    + obs tabs: $prob no.1: catab001.csv, cotab001, patab001, sdtab001
##    + sim tabs: $prob no.2: simtab001.zip
##    + output files: run001.cor, run001.cov, run001.ext, run001.grd, run001.phi, run001.shk
##    + special: <none>
##  - gg_theme: theme_readable
##  - xp_theme: theme_xp_default
##  - Options: dir = analysis/models/pk/, quiet = FALSE, manual_import = NULL
```

2.2 Access model code

The `get_code()` function can be used to access the parsed model code from the xpdb. This code was used to create the summary and find table names. The parsed code can be used to get additional information about the run. If the argument `.problem` is specified a subset of the code can be returned based on `$PROBLEM`.

Note that general code warnings and PsN outputs appended are listed as problem 0.

```
code <- get_code(xpdb)
code
```

```
## # A tibble: 764 x 5
##    problem level subroutine code                          comment
##  *   <int> <int> <chr>      <chr>                         <chr>
##  1       0     0 oth        Mon Oct 16 13:34:28 CEST 20~ ""
##  2       0     0 oth        ""                           ; 1. Based on: 0~
##  3       0     0 oth        ""                           "; 2. Descriptio~
##  4       0     0 oth        ""                           ; NONMEM PK exam~
##  5       1     1 pro        Parameter estimation         ""
##  6       1     2 inp        ID DOSE DV SCR AGE SEX CLAS~ ""
##  7       1     2 inp        " CLCR AMT SS II EVID"       ""
##  8       1     3 dat        ../../mx19_2.csv IGNORE=@    ""
##  9       1     4 abb        DERIV2=NO                    ""
## 10       1     5 sub        ADVAN2 TRANS1                ""
## # ... with 754 more rows
```

2.3 Access the output data

The `get_data()` function can be used to access the imported table files. Tables can be accessed by `table` name or by `.problem`. In the latter a single dataset containing all aggregated tables is returned. If more than one `table` name or `.problem` number is provided a named list is returned.

Note when providing a table name it is not guaranteed that the table will be identical to its file (i.e. the order of the columns may have been changed and tables with `FIRSTONLY` will no longer be deduplicated).

```
data <- get_data(xpdb, table = 'patab001')
data
```

```
## # A tibble: 550 x 8
##    ID       KA    CL     V ALAG1    ETA1     ETA2    ETA3
##    <fct> <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>   <dbl>
##  1 110   0.496  25.5   141 0.208 -0.0370 -0.00596 -2.14
##  2 110   0.496  25.5   141 0.208 -0.0370 -0.00596 -2.14
##  3 110   0.496  25.5   141 0.208 -0.0370 -0.00596 -2.14
##  4 110   0.496  25.5   141 0.208 -0.0370 -0.00596 -2.14
##  5 110   0.496  25.5   141 0.208 -0.0370 -0.00596 -2.14
##  6 110   0.496  25.5   141 0.208 -0.0370 -0.00596 -2.14
##  7 110   0.496  25.5   141 0.208 -0.0370 -0.00596 -2.14
##  8 112   4.11   21.8   122 0.208 -0.0495  0.122   -0.0235
##  9 112   4.11   21.8   122 0.208 -0.0495  0.122   -0.0235
## 10 112   4.11   21.8   122 0.208 -0.0495  0.122   -0.0235
## # ... with 540 more rows
```

2.4 Access the run files

The `get_file()` function can be used to access the imported output files. Files can be accessed by `file` name, by `.problem`, `.subprob` and/or `.method`. If more than one `file` name, `.problem`, `.subprob`, or `.method` is provided a named list is returned.

```
file <- get_file(xpdb, file = 'run001.ext')
file
```

```
## # A tibble: 28 x 16
##    ITERATION THETA1 THETA2 THETA3 THETA4 THETA5  THETA6  THETA7
##        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>
##  1      0      25.4   1.47   7.45  0.214  0.200 0.00983 0.00601
##  2      1.00   26.3   1.26   7.35  0.219  0.217 0.00989 0.00602
##  3      2.00   25.6   1.47   7.29  0.216  0.212 0.00987 0.00603
##  4      3.00   26.8   1.49   5.76  0.213  0.213 0.00979 0.00628
##  5      4.00   26.7   1.49   5.69  0.213  0.212 0.00979 0.00629
##  6      5.00   26.7   1.49   5.66  0.213  0.212 0.00979 0.00630
##  7      6.00   26.6   1.49   5.03  0.210  0.217 0.0100  0.00652
##  8      7.00   26.6   1.49   4.93  0.205  0.217 0.0100  0.00658
##  9      8.00   26.6   1.48   4.62  0.211  0.217 0.00951 0.00735
## 10      9.00   26.6   1.46   4.41  0.209  0.217 0.00903 0.00874
## # ... with 18 more rows, and 8 more variables: `SIGMA(1,1)` <dbl>,
## #   `OMEGA(1,1)` <dbl>, `OMEGA(2,1)` <dbl>, `OMEGA(2,2)` <dbl>,
## #   `OMEGA(3,1)` <dbl>, `OMEGA(3,2)` <dbl>, `OMEGA(3,3)` <dbl>, OBJ <dbl>
```

2.5 Access the parameter estimates

The `get_prm()` function can be used to access the parameter estimates. To get a nice parameter table printed to the console use the function `prm_table()` instead. The arguments `.problem`, `.subprob` and `.method` can be used to select the parameter estimates to output.

```
# Raw output for editing
prm <- get_prm(xpdb, digits = 4)
prm
```

```
## # A tibble: 11 x 10
##    type  name   label    value       se     rse fixed diagonal     m     n
##  * <chr> <chr>  <chr>    <dbl>    <dbl>   <dbl> <lgl> <lgl>    <dbl> <dbl>
##  1 the   THETA1 TVCL   2.63e+1  0.892    0.0339 F     NA        1.00 NA
##  2 the   THETA2 TVV    1.35e+0  0.0438   0.0325 F     NA        2.00 NA
##  3 the   THETA3 TVKA   4.20e+0  0.809    0.192  F     NA        3.00 NA
##  4 the   THETA4 LAG    2.08e-1  0.0157   0.0755 F     NA        4.00 NA
##  5 the   THETA5 Prop.~ 2.05e-1  0.0224   0.110  F     NA        5.00 NA
##  6 the   THETA6 Add. ~ 1.06e-2  0.00366  0.347  F     NA        6.00 NA
##  7 the   THETA7 CRCL ~ 7.17e-3  0.00170  0.237  F     NA        7.00 NA
##  8 ome   OMEGA~ IIV CL 2.70e-1  0.0233   0.0862 F     T         1.00  1.00
##  9 ome   OMEGA~ IIV V  1.95e-1  0.0320   0.164  F     T         2.00  2.00
## 10 ome   OMEGA~ IIV KA 1.38e+0  0.202    0.146  F     T         3.00  3.00
## 11 sig   SIGMA~ ""     1.00e+0 NA       NA      T     T         1.00  1.00
```

```
# Nicely formated table
prm_table(xpdb, digits = 4)
```

```
##
## Reporting transformed parameters:
## For the OMEGA and SIGMA matrices, values are reported as standard deviations for the diagonal elemen
##
## Estimates for $prob no.1, subprob no.0, method foce
##  Parameter  Label      Value        RSE
##  THETA1     TVCL       26.29        0.03391
##  THETA2     TVV        1.348        0.0325
##  THETA3     TVKA       4.204        0.1925
##  THETA4     LAG        0.208        0.07554
```

```
##  THETA5     Prop. Err  0.2046       0.1097
##  THETA6     Add. Err   0.01055      0.3466
##  THETA7     CRCL on CL 0.007172     0.2366
##  OMEGA(1,1) IIV CL     0.2701       0.08616
##  OMEGA(2,2) IIV V      0.195        0.1643
##  OMEGA(3,3) IIV KA     1.381        0.1463
##  SIGMA(1,1)            1        fix  -
```

For the OMEGA and SIGMA matrices, values are reported as standard deviations for the diagonal elements and as correlations for the off-diagonal elements. The relative standard errors (RSE) for OMEGA and SIGMA are reported on the approximate standard deviation scale (SE/variance estimate)/2. Use `transform = FALSE` to report untransformed parameters.

2.6 Access the run summary

The `get_summary()` function can be used to access the generated run summary from which the `template_titles`. If the argument `.problem` is specified a subset of the summary can be returned based on `$PROBLEM`.

Note that general summary information are listed as problem 0.

```
run_sum <- get_summary(xpdb, .problem = 0)
run_sum
```

```
## # A tibble: 12 x 5
##    problem subprob descr               label       value
##      <dbl>   <dbl> <chr>               <chr>       <chr>
##  1       0       0 Run description     descr       NONMEM PK example for ~
##  2       0       0 Run directory       dir         analysis/models/pk/
##  3       0       0 Run errors          errors      na
##  4       0       0 ESAMPLE seed number esampleseed na
##  5       0       0 Run file            file        run001.lst
##  6       0       0 Number of ESAMPLE   nesample    na
##  7       0       0 Reference model     ref         000
##  8       0       0 Run number          run         run001
##  9       0       0 Software            software    nonmem
## 10       0       0 Run start time      timestart   Mon Oct 16 13:34:28 CE~
## 11       0       0 Run stop time       timestop    Mon Oct 16 13:34:35 CE~
## 12       0       0 Software version    version     7.3.0
```

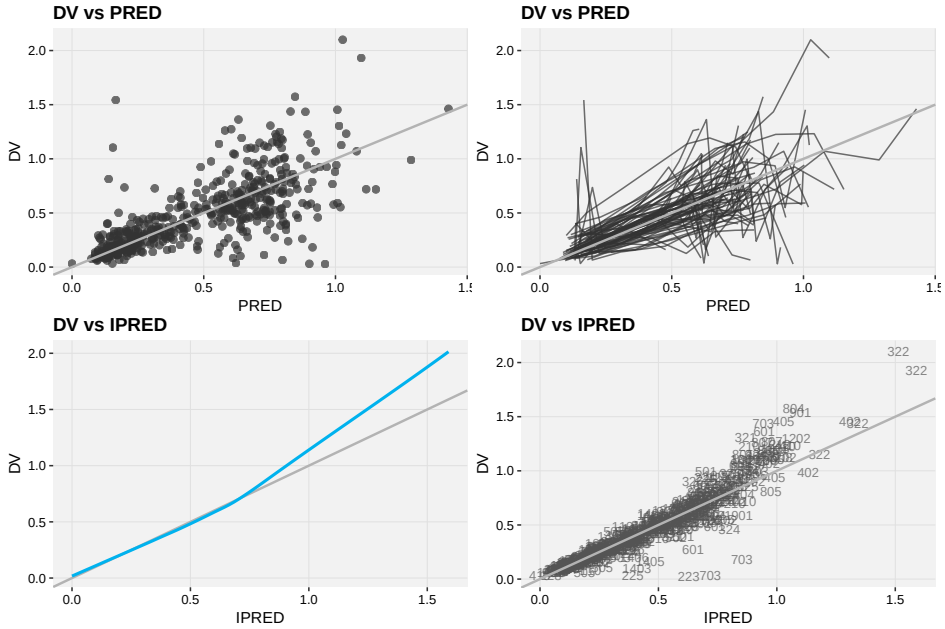
3 Basic GOF plots

3.1 Scatter plots

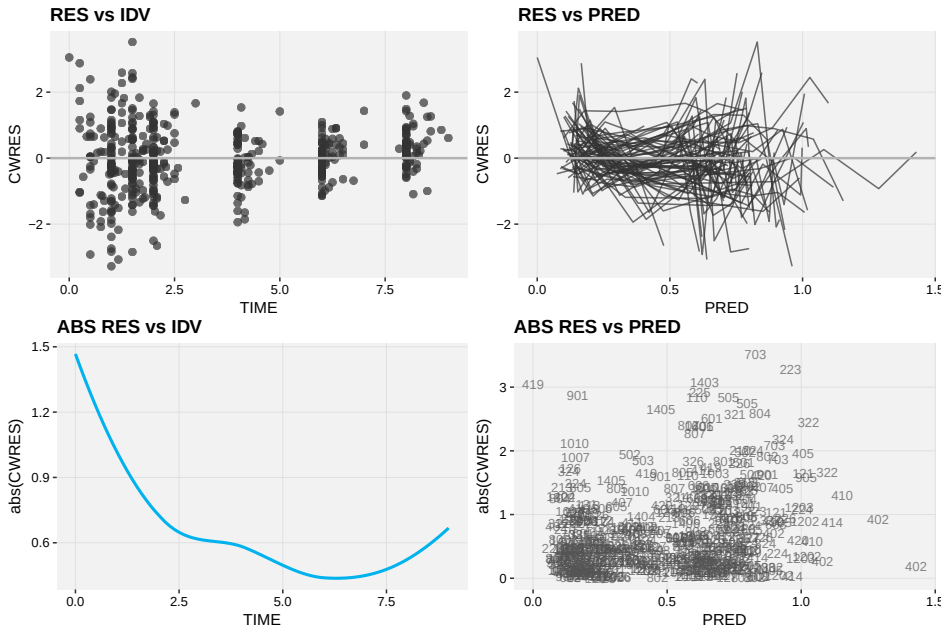
Use the `dv_vs_xxx()` to create basic scatter plots for `IPRED` and `PRED`. Use `type=` to switch between line `l`, point `p`, smooth `s` and text `t`. Similarly residual scatter plots can be created by using `res_vs_xxx()` or `absval_res_vs_xxx()` for `IDV` and `PRED`.

```
gridExtra::grid.arrange(
  dv_vs_pred(xpdb, title = "DV vs PRED", subtitle = NULL, caption = NULL, type = 'p'),
  dv_vs_pred(xpdb, title = "DV vs PRED", subtitle = NULL, caption = NULL, type = 'l'),
  dv_vs_ipred(xpdb, title = "DV vs IPRED", subtitle = NULL, caption = NULL, type = 's'),
  dv_vs_ipred(xpdb, title = "DV vs IPRED", subtitle = NULL, caption = NULL, type = 't'),
```

```
  ncol = 2
)
```



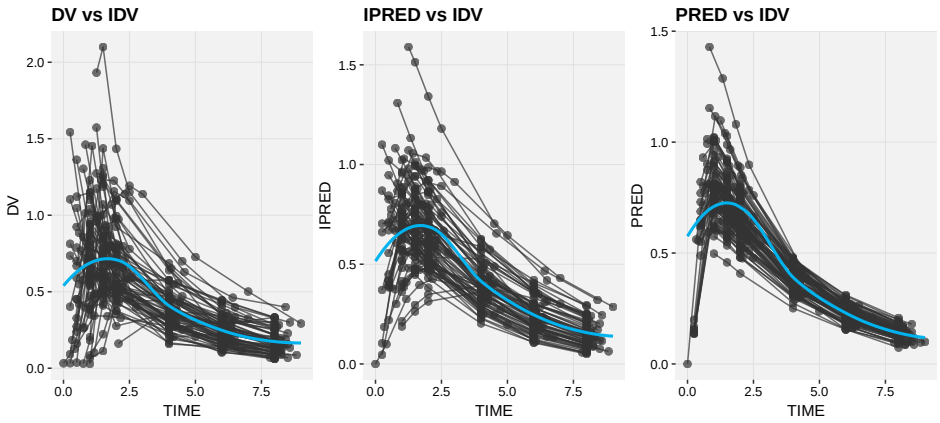

```
gridExtra::grid.arrange(
  res_vs_idv(xpdb, title = "RES vs IDV", subtitle = NULL, caption = NULL, type = 'p'),
  res_vs_pred(xpdb, title = "RES vs PRED", subtitle = NULL, caption = NULL, type = 'l'),
  absval_res_vs_idv(xpdb,title="ABS RES vs IDV",
                    subtitle = NULL,caption = NULL,type = 's'),
  absval_res_vs_pred(xpdb,title="ABS RES vs PRED",
                     subtitle = NULL,caption = NULL,type = 't'),
  ncol = 2
)
```



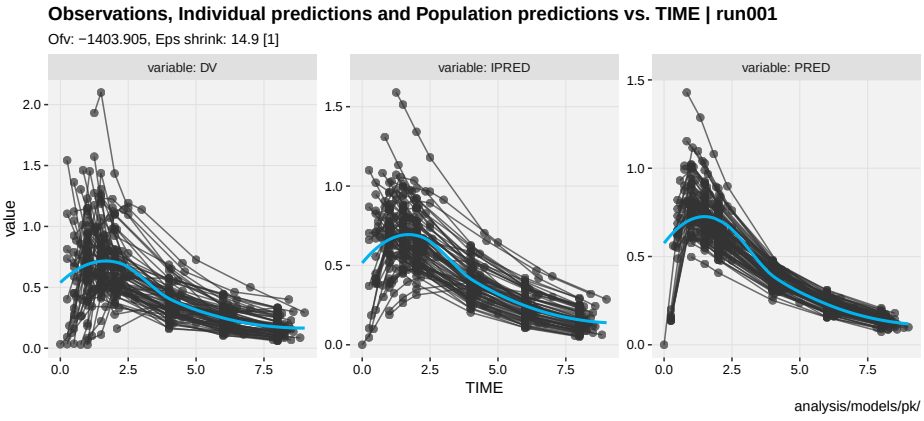
3.2 More scatter plots

Use the `xxx_vs_idv` to plot scatter with trends for `DV`, `IPRED` and `PRED`. The `dv_preds_vs_idv()` lets you plot DV, PRED and IPRED side by side.

```
gridExtra::grid.arrange(
  dv_vs_idv(xpdb, title = "DV vs IDV", subtitle = NULL, caption = NULL),
  ipred_vs_idv(xpdb, title = "IPRED vs IDV",
               subtitle = NULL, caption = NULL),
  pred_vs_idv(xpdb, title = "PRED vs IDV",
              subtitle = NULL, caption = NULL),
  ncol = 3
)
```



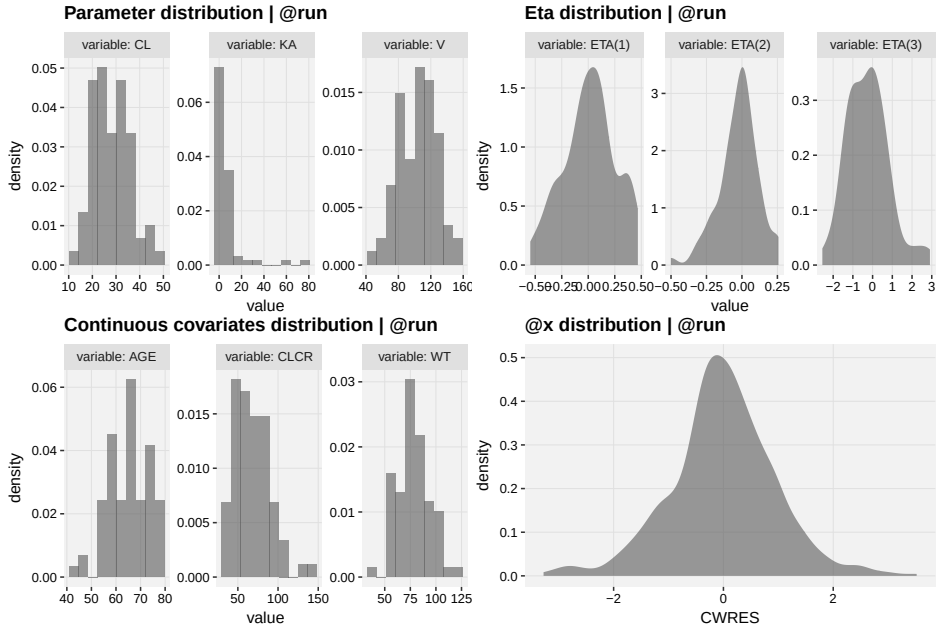

```
dv_preds_vs_idv(xpdb)
```



4 Distributions

Use `xxx_distrib()` to create distribution plots for `parameters`, `etas`, `covariates` and `residuals`. Use the `type=` to change plot type from histogram `h` to desity `d`.

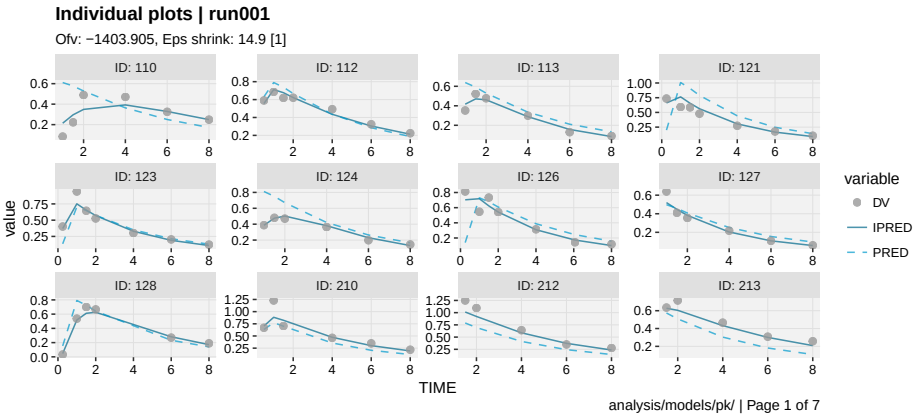
```
gridExtra::grid.arrange(
  prm_distrib(xpdb, subtitle = NULL, caption = NULL, type = 'h'),
  eta_distrib(xpdb, subtitle = NULL, caption = NULL, type = 'd'),
  cov_distrib(xpdb, subtitle = NULL, caption = NULL, type = 'h'),
  res_distrib(xpdb,res = "CWRES",subtitle = NULL, caption = NULL, type = 'd'),
  ncol=2)
```



5 Individual plots

The `ind_plots` plots the individually faceted fits.

```
ind_plots(xpdb, page = 1,
          ncol = 4, nrow = 3)
```



6 Visual Predictive Checks

6.1 Introduction

VPC can be created either by:

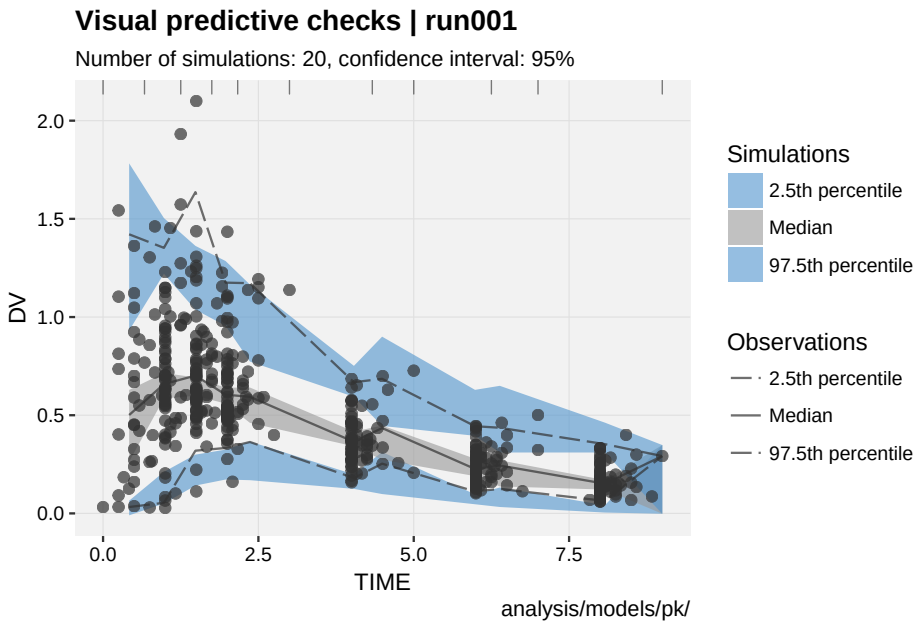
1. Using an xpdb containing a simulation and an estimation problem
2. Using a PsN generated VPC folder

The VPC functionality in xpose is build around the vpc R package. For more details about the way the vpc package works, please check the documentation website.

6.2 Basics of VPC in xpose

The VPC computing and plotting parts have been separated into two distinct functions: `vpc_data()` and `vpc()` respectively.

```
xpdb %>%
 vpc_data() %>%
  vpc()
```



6.3 Creating VPC using the xpdb data

To create VPC using the xpdb data, at least one simulation and one estimation problem need to present. Hence in the case of NONMEM the run used to generate the xpdb should contain several `$PROBLEM`. In `vpc_data()` the problem number can be specified for the observation (`obs_problem`) and the simulation (`sim_problem`). By default xpose picks the last one of each to generate the VPC.

6.4 Creating the VPC using a PsN folder

The `vpc_data()` contains an argument `psn_foler` which can be used to point to a PsN generated VPC folder. As in most xpose function `template_titles` keywords can be used to automatize the process e.g.

`psn_folder = '@dir/@run_vpc'` where `@dir` and `@run` will be automatically translated to initial (i.e. when the xpdb was generated) run directory and run number `'analysis/models/pk/run001_vpc'`.

In this case, the data will be read from the `/m1` sub-folder (or `m1.zip` if compressed). Note that PsN drops unused columns to reduce the simtab file size. Thus, in order to allow for more flexibility in R, it is recommended to use multiple stratifying variables (`-stratify_on=VAR1,VAR2`) and the prediction corrected (`-predcorr` adds the PRED column to the output) options in PsN to avoid having to rerun PsN to add these variables later on. In addition, `-dv`, `-idv`, `-lloq`, `-uloq`, `-predcorr` and `-stratify_on` PsN options are automatically applied to xpose VPC.

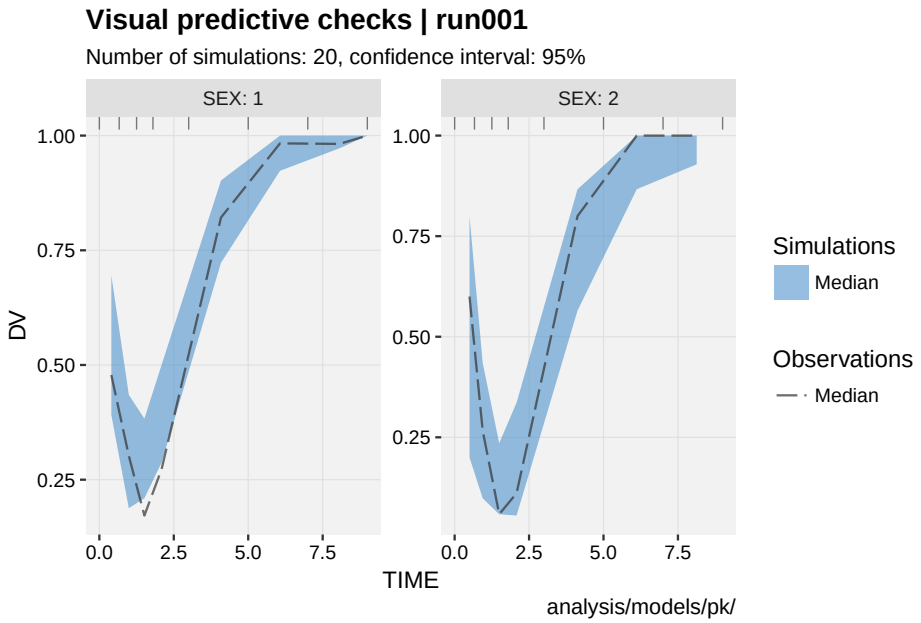
The PsN generated binning can also applied to xpose VPC with the `vpc_data()` option `psn_bins = TRUE` (disabled by default). However PsN and the vpc package work slightly differently so the results may not be optimal and the output should be evaluated carefully.

```
xpdb %>%
 vpc_data(psn_folder = '@dir/run001_vpc', psn_bins = TRUE) %>%
 vpc()
```

6.5 Options in `vpc_data()`

- The option `vpc_type` allows to specify the type of VPC to be computed: "continuous" (default), "categorical", "censored", "time-to-event".
- The `stratify` options defines up to two stratifying variable to be used when computing the VPC data. The `stratify` variables can either be provided as a character vector (`stratify = c('SEX', 'MED1')`) or a formula (`stratify = SEX~MED1`) . The former will result in the use of `ggforce::facet_wrap_paginate()` and the latter of `ggforce::facet_grid_paginate()` when creating the plot. With "categorical" VPC the "group" variable will also be added by default.
- More advanced options (i.e. binning, pi, ci, predcorr, lloq, etc.) are accessible via the `opt` argument. The `opt` argument expects the output from the `vpc_opt()` functions argument.

```
xpdb %>%
 vpc_data(vpc_type = 'censored', stratify = 'SEX',
          opt = vpc_opt(bins = 'jenks', n_bins = 7, lloq = 0.5)) %>%
 vpc()
```



6.6 Options in `vpc()`

- The option `vpc_type` works similarly to `vpc_data()` and is only required if several VPC data are associated with the xpdb.
- The option `smooth = TRUE/FALSE` allows to switch between smooth and squared shaded areas.
- The plot VPC function works similarly to all other xpose functions to map and customize aesthetics. However in this case the `area_fill` and `line_linetype` each require three values for the low, median and high percentiles respectively.

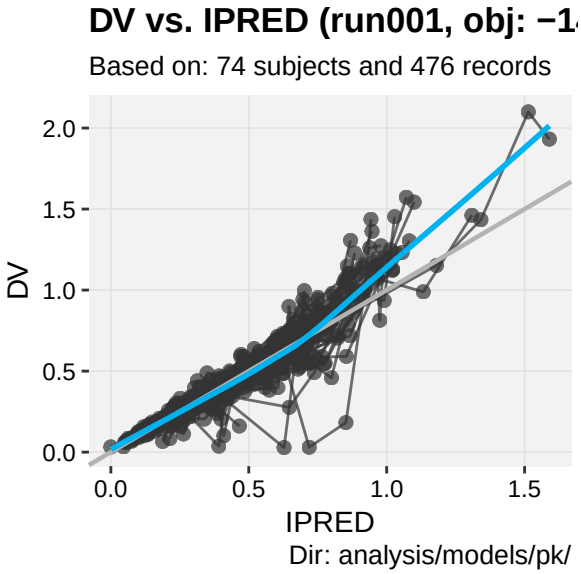
7 Cutomize plots

7.1 Labels

All xpose plots have by default an informative title, subtitle and caption. For example all plots using individual model predictions (IPRED) will display the epsilons' shrinkage. These titles can easily be edited as templates using @keywords which will be replaced by their actual value stored in the summary level of the `xpdb` object when rendering the plots. Keywords are defined by a word preceded by a `@` e.g. `'@ofv'`. A list of all available keyword can be accessed via `help('template_titles')`. The title, subtitle or caption can be disabled by setting them to `NULL`. Suffix can be automatically added to title, subtitle and caption of all plots. The suffixes can be defined in the `xp_theme`.

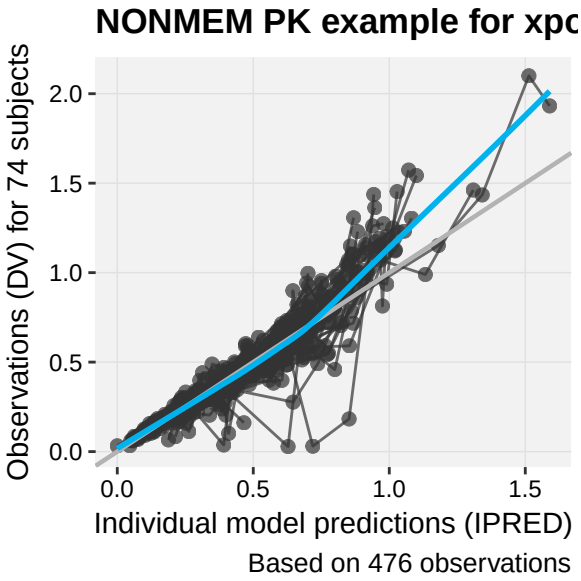
There are two ways to go about this:

```
# Using xpose
dv_vs_ipred(xpdb,
            title    = '@y vs. @x (@run, obj: @ofv)',
            subtitle = 'Based on: @nind subjects and @nobs records',
            caption  = 'Dir: @dir')
```




```
# Using ggplot
dv_vs_ipred(xpdb) +
  labs(title    = '@descr',
       subtitle = NULL,
       caption  = 'Based on @nobs observations',
```

```
       x        = 'Individual model predictions (@x)',
       y        = 'Observations (@y) for @nind subjects')
```

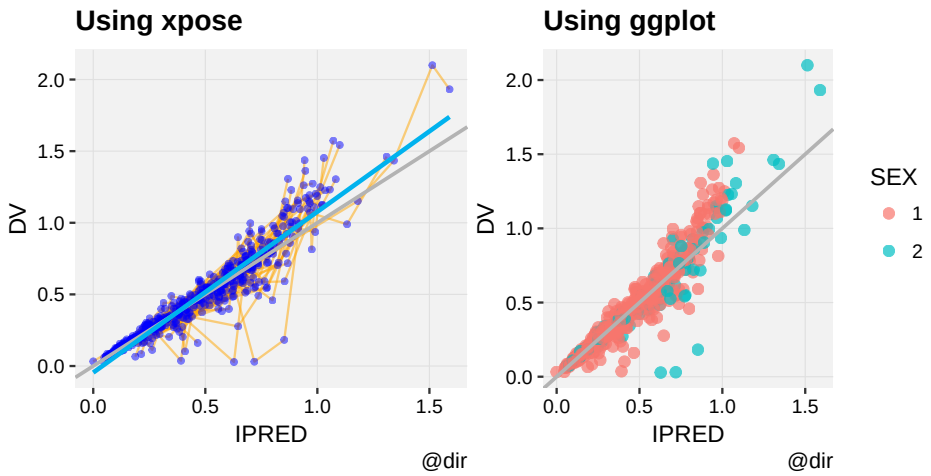


7.2 Modify aesthetics

By default the aesthetics are read from the `xp_theme` level in the xpdb object but these can be modified in any plot function. xpose makes use of the `ggplot2` functions mapping for any layer (e.g. points, lines, etc.) however to direct the mapping to a specific layer, a prefix appealing to the targeted layer should be used. The format is defined as `layer_aesthetic = value`. Hence to change the color of points in ggplot2 the argument `color = 'green'` could be used in `geom_point()`, while in xpose the same could be achieved with `point_color = 'green'`.

In basic goodness-of-fit plots, the layers have been named as: `point_xxx`, `line_xxx`, `smooth_xxx`, `guide_xxx`, `xscale_xxx`, `yscale_xxx` where `xxx` can be any option available in the `ggplot2` layers: `geom_point`, `geom_line`, `geom_smooth`, `geom_abline`, `scale_x_continuous`, etc.

```
#Using xpose
a <- dv_vs_ipred(xpdb,
            title = "Using xpose", subtitle = NULL,
            # Change points aesthetics
            point_color = 'blue', point_alpha = 0.5,
            point_stroke = 0, point_size = 1.5,
            # Change lines aesthetics
            line_alpha = 0.5, line_size = 0.5,
            line_color = 'orange', line_linetype = 'solid',
            # Change smooth aesthetics
            smooth_method = 'lm')

#Using ggplot
b <- dv_vs_ipred(xpdb,
                type = 'p',title = "Using ggplot", subtitle = NULL,
                aes(point_color = SEX))

gridExtra::grid.arrange(a,b,ncol=2)
```

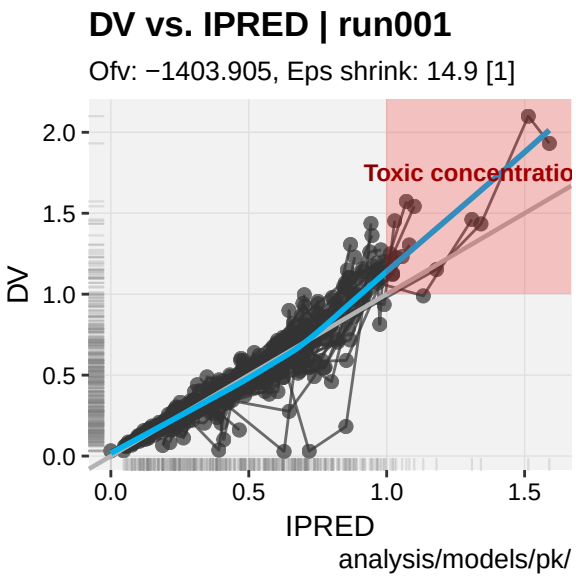


7.3 Additional layers

`xpose` offers the opportunity to add any additional layers from `ggplot2`. Example, a `ggplot2::geom_rug()` layer could be added to the `dv_vs_ipred()` plot along with some annotations (`ggplot2::annotate()`). Note: the additional layers do not inherit from the xpose aesthetic mapping (i.e. colors or other options need to be defined in each layer as shown below).

Layers can also be used to modify the aesthetics scales for example `ggplot2::scale_color_manual()`, or remove a legend `ggplot2::scale_fill_identity()`.

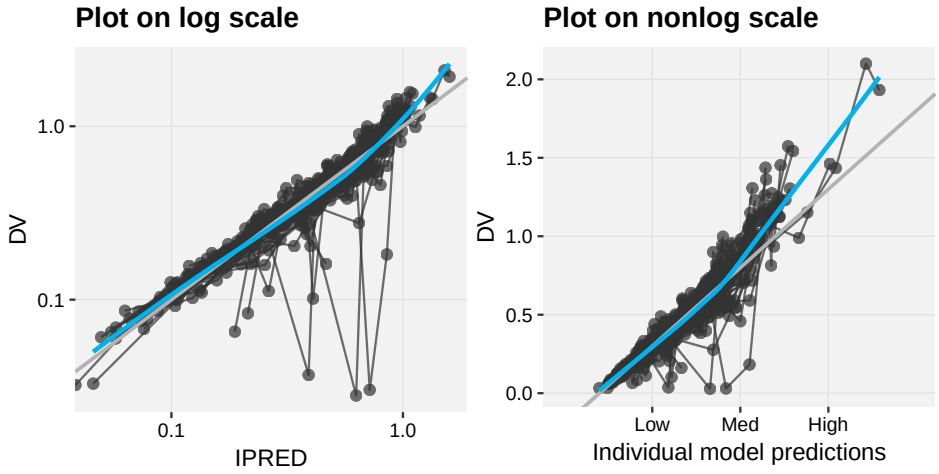
```
dv_vs_ipred(xpdb) +
  geom_rug(alpha = 0.2, color = 'grey50',
           sides = 'lb', size = 0.4) +
  annotate(geom = 'text',
           fontface = 'bold',
           color = 'darkred',
           size  = 3,
           label = 'Toxic concentrations', x = 1.35, y = 1.75) +
  annotate(geom = 'rect',
           alpha = 0.2, fill = 'red',
           xmin = 1, xmax = Inf,
           ymin = 1, ymax = Inf)
```



7.4 Scales

The argument `log` allows to log-transform the axes. Accepted values are `x`, `y` or `xy`. Additional arguments can be provided to the scales via the mapping by using the naming convention `xscale_xxx` or `yscale_xxx` where `xxx` is the name of a `ggplot2` scale argument such as `name`, `breaks`, `labels`, `expand`.

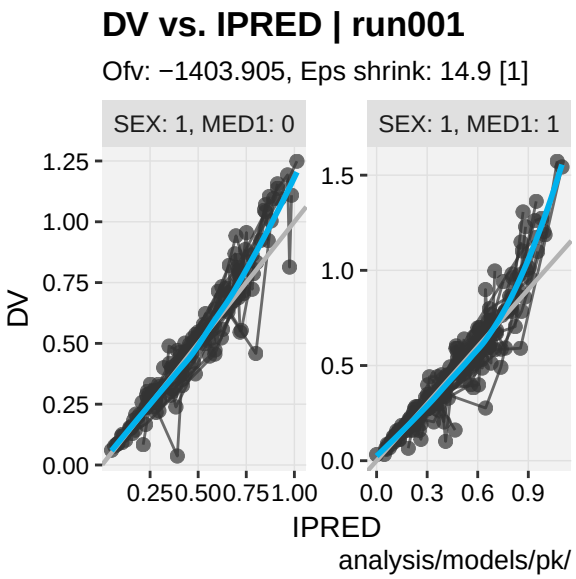
```
a <- dv_vs_ipred(xpdb, log = 'xy',
                 title = 'Plot on log scale',
                 subtitle = NULL,caption=NULL)

b <- dv_vs_ipred(xpdb,
             title = 'Plot on nonlog scale',
             subtitle = NULL, caption = NULL,
             xscale_breaks = c(0.3, 0.8, 1.3),
             xscale_labels = c('Low', 'Med', 'High'),
             xscale_expand = c(0.2, 0),
             xscale_name = 'Individual model predictions')

gridExtra::grid.arrange(a,b,ncol=2)
```



7.5 Facets

Panels (or faceting) can be created by using the `facets` argument as follows:

```
# Example with a string
dv_vs_ipred(xpdb, facets = c('SEX', 'MED1'), ncol = 2, nrow = 1, page = 1)
```




```
# Example with a formula
dv_vs_ipred(xpdb, facets = SEX~MED1, margins = TRUE)
```