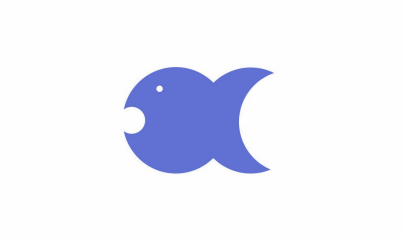
PsN Validation Tool

Pirana Software & Consulting BV



Version 1.0.0
Installation guide and Manual

April 5, 2013

Contents

1 Introduction **2**
1.1 How it works 2
1.2 Disclaimers 3

2 Installation **4**

3 Reference manual **6**
3.1 Setting up the validation 6
3.2 Running the validation 8
3.3 Validation of PsN toolkit commands 9
3.3.1 Tool-specific settings 9
3.4 Custom R scripts 16

Chapter 1

Introduction

ValPsN is a software tool developed by Pirana Software & Consulting BV for comparing and validating output of PsN and NONMEM. *ValPsN* compares the numeric output from a sequence of PsN runs (execute, bootstrap, vpc, etc...) with that of a reference run. Differences in PsN output might result either from differences in NONMEM output, or differences in subsequent numerical operations performed by PsN. Therefore, by 'validating' PsN in this way, at the same time the underlying NONMEM infrastructure is tested and validated.

This installation guide and manual contains important guidance for the installation and use of *ValPsN*, and should be read before starting a validation. We welcome suggestions for further improvements and bug reports.

1.1 How it works

The basic principle is simple: *ValPsN* compares output produced from a specific installation of PsN on your system (*test*), with output derived from an earlier PsN run (*reference*). It is thus implicitly assumed that the output from the *reference* is trustworthy and correct. While the framework for the validation is implemented in Perl, the comparison of the numeric output is performed in R, using predefined R-scripts. For every available PsN tool (e.g. execute, bootstrap, vpc etc.) a standard R-script is included in *ValPsN* that performs the specific validation. It is however also possible to specify a custom R-script if it adheres to a few requirements (specified in the Reference chapter).

The proposed workflow for a validation of PsN is as follows:

1. Collect a reference library with output from PsN toolkit that is known / believed to be correct. Of course, for an appropriate validation of PsN, output from all

toolkit commands available in PsN, or at least from a selection of the most often used tools should be included in the validation run.

2. Create a configuration file for *ValPsN*. This is a flat text file that specifies e.g. which PsN version to use, where to store the test output, which PsN tools to run, as well as the speficic options for each tool. Please see the Reference chapter of this manual.

3. Run *ValPsN* and generate the *test* output, which is then compared to the *reference* output. You can also choose to use previously generated output as *test* output, instead of rerunning the specific PsN tools in the validation run itself.

1.2 Disclaimers

PsN You will note that the command line interface of ValPsN closely resembles that of PsN. This is to facilitate the use for those already accustomed to PsN. However, we would like to stress that ValPsN does not use any PsN library nor does it share any code with PsN.

Operating system The current version of ValPsN is developed primarily for Linux (and Mac OSX). Although this has not been tested yet, the software should run fine on Windows as well, since only standard Perl and PsN toolkit commands are used. This installation guide and manual assumes Unix-type installation.

Sequential execution The *ValPsN* script is implemented sequentially: it will run a sequence of PsN commands (each one followed by an R scripts), in the order specified. At current, there is no option to run these tests in parallel, so the whole validation procedure may take a long time. An easy workaround for this is to split up the validation procedure in separate validations that can be run in parallel, e.g. a validation run in which a set of execute commands is performed, one run in which a bootstrap is run, and one in which the other PsN tools such as vpc, llp, scm are validated.

Chapter 2

Installation

A zip-file containing the *ValPsN* software is provided (e.g. valpsn_[version].zip), either from the Pirana website or from a different download site provided by the developer. Unpack the zip file, and copy the contents to an appropriate folder on the machine or server of interest, e.g. in /opt/valpsn/, or /home/username/valpsn.

Check that in your installation folder you now have at least the following files and folders:

```
/doc          # Contains some documents, e.g. this manual
/ini          # Example setup up files for validation runs
/library      # Library of example PsN output files
/R            # Default validation scripts
valpsn.pl     # the ValPsN tool
val_ex_1.ini  # example / reference configuration file
test.ini      # test configuration file
```

For ValPsN to function properly, three things need to be accessible on the machine or on the cluster.

- one (or more) installations of PsN
- one (or more) installations of NONMEM
- an installation of R (available from http://cran.r-project.org)

To test if ValPsN is set up correctly, and is able to run PsN tools and invoke NONMEM, run this command (from the folder where ValPsN is installed):

```
perl valpsn.pl test.ini
```

This will run a quick validation run, and invokes PsN, NONMEM and R using default settings. It will create the folder

`~/valpsn_test`

in your home folder, unless otherwise specified in test.ini. Completion should take less than 15 seconds, and should report that the validation succeeded. Of course, this is not part of a validation yet, it just tests if the validation infrastructure is installed and accessible by *ValPsN*. If however one of these three tools spawns an error, or the tools cannot be accessed, please fix these respective problems before continuing validation. It is difficult to give specific troubleshooting advice here, but problems may be due to e.g. incorrectly set paths, unavailable or incorrectly set environment variables, or insufficient access priviliges.

Chapter 3

Reference manual

In the file `val_ex_1.ini` an example validation is implemented. Note that all files for this example, including model files and datasets are included, as well as the output from the PsN commands, so this example can be run as a test. Feel free to include this examples in your own validation library as well.

The execute command is at the core of PsN, since it performs a single NONMEM run, and is used in all other tools that perform estimation or simulation. A validation of execute is therefore also a validation of the NONMEM infrastructure. It is therefore advised to collect a considerable number of (different types of) models in your validation library and test these with the execute command. The other PsN tools only parse output from execute and thus the validation steps there are aimed at validating the PsN algorithms. It is advised to implement more than one instance of each toolkit command in the validation procedure.

3.1 Setting up the validation

In the main folder, locate the file `val_ex_1.ini`, and make a copy, e.g. to the file `test1.ini`, and edit to your liking. The contents of this configuration file are explained step-by-step below.

The `[folder]` section defines the folder structure, e.g. where the tool can find the model library, and where it should put the output:

```
[folders]
lib_folder    = /Users/ronkeizer/Dropbox/projects/psn_validate/library
                # Absolute path to model library
out_folder    = /Users/ronkeizer/ValPsn/runs
                # Absolute path where to run validation tests
R_script_folder = /Users/ronkeizer/Dropbox/projects/psn_validate/R
```

```
report_folder  = /Users/ronkeizer/ValPsn/reports
                 # Absolute path to folder for validation reports
run_folder     = PsN_3.5.3_20120930
                 # Folder for this specific validation run,
                 # will be created below the "out_folder"
report         = PsN_3.5.3_20120930.txt
                 # Report file to generate, will be
                 # created in the "report_folder"
```

Secondly, the software settings for the validation run are defined:

```
[software]
perl_executable=perl    # location of perl executable, if not in PATH
psn_executables=        # location of folder with PsN executables,
                        # No need to specify this if in PATH.
psn_version=            # Optionally specify a different version to
                        # use, e.g. psn_vesion=3.5.3. If not speci-
                        # fied, the default version is used.
R_executable=R          # Location of R executable, if not in PATH
nm_mod_extension=mod    # File extension of NONMEM model files
nm_out_extension=lst    # Filename extension of NONMEM output files
```

The next section defines the default settings for the validation run. These settings will be carried over into each specific validation step, unless overriden in the respective step itself.

```
[validation]            # General validation settings
                        # Note: all these settings can be overridden
                        # in each specific PsN test!
run_psn=1               # Default is 1|TRUE. set this to FALSE if you
                        # only want to generate a validation report
                        # based on output from a previous validation
                        # procedure.
run_test=1              # Actually perform the tests using the R-scripts?
                        # By default set to 1
verbose_level=2         # 1: condensed info
                        # 2: extended run/test info (default)
                        # 3: also include NONMEM output
nm_version=default      # NONMEM version to test. Can also be
                        # issued specifically per test, which
                        # overrides the general setting. If not
                        # specified, PsN will use "default".
extra_arguments=        # Optional arguments to add to all PsN
                        # commands, e.g.:
                        # -clean=1 -model_dirname
threads=5               # Number of threads to use for PsN commands
tolerance=0.005         # Allowed relative difference between test
                        # and reference estimate. Specify as
                        # fraction: (test-ref)/ref. So the default
                        # 0.001 = 0.1\% difference allowed.
clean=2                 # PsN -clean option: (0/1/2/3)
```

The `tolerance` parameter is important. It will be obvious that too high a tolerance will render your validation unuseful, but specifying a too low tolerance might fail your validation on too many tests. Small changes in results can be expected between NONMEM

runs, and can be due to differences in (in order of likeliness) the NONMEM version, Fortran compiler and compiler settings, machine hardware and operating system. A tolerance of 0.005 is suggested for comparing the parameter estimates for the execute command, which will most likely be small enough to detect relevant differences, but large enough to not be affected too much by the issues mentioned above. Of course, if your desire is to achieve perfect agreement between test and reference numbers, you can always set the tolerance to a much lower figure. For tools like boostrap and VPC, it is advised to set the tolerance (much) higher when the `-seed` function is not used.

What follows next, are the specification settings for each validation step. There is no limit on the number of tests to be included in the validation run, and of course multiple tests can (and should) be run for the different PsN toolkit functions. Each specification of a test starts with the header that specifies the function in brackets, followed by the specific settings for that tool, e.g.

```
[execute]
folder=NM_examples
model=run4.mod
reference=run4.lst
# tests
test_ofv=TRUE
test_parameters=TRUE
# /tests
```

3.2 Running the validation

Every validation run is started by invoking the valpsn.pl script. This bash script invokes the main perl script that acutally performs the validation. The script will read in the configuration file, and perform all the specified validation elements in the sequence specified in the configuration file, e.g.:

```
perl valpsn.pl -ini=ex1\_20130201.ini
```

The script will now run through the validation steps specified in the configuration file, in each step first running the PsN command (if requested), and next invoking the R script to compare the output with the reference. At the end of the validation, *ValPsN* will report a summary of the validation procedure, e.g.:

```
--------------------------------------------------------------------------------
Summary: Succes (10/10), failed (0/10), not tested (0/10)
--------------------------------------------------------------------------------
```

If tests have failed, it will report which tests have failed, so they can be checked and run again separately.
Similar to PsN toolkit commands, you can run

```
perl valpsn.pl -h
```

or

```
perl valpsn.pl -help
```

to get some more help on the command line arguments that are available.

3.3 Validation of PsN toolkit commands

Below is a reference guide split per PsN toolkit command, which specifies all options available in the configuration file.

Disclaimer: The descriptions given for the PsN tools are reproduced from the PsN website (http://psn.sourceforge.net), if available. Also note that PsN is a tool in constant development, and default PsN settings or algorithms may change between versions. It is therefore advised to be as explicit as possible in specifying the PsN command to run in the validation procedure. Output format from PsN commands may also change between versions, and the R scripts may therefore need some adaptation over time.

3.3.1 Tool-specific settings

Settings specified in the tool-specific sections override those in the `[validation]` section above. Tests implemented in the R script are only implemented if they are set to 1 (or TRUE):

```
compare_bootstrap_parameters=1 # Compare parameters pairwise
```

execute

The execute script is a PsN tool that allows you to run multiple modelfiles either sequentially or in parallel. It is an nmfe replacement with advanced extra functionality. Execute creates subdirectories where it puts NONMEMs input and output files, to make sure that parallel NONMEM runs do not interfere with each other. The top directory is by default named 'modelfit_dirX' where 'X' is a number that starts at 1 and is increased by one each time you run execute.

The execute command performs a single NONMEM run. The output from execute is non-numeric, i.e. it is a flat text file (usually .lst or .res file)) that contains a lot of info about the model, dataset and run process. It does include the parameter estimates, and possibly the estimates of uncertainty in parameter estimates. To convert these parameter estimates information into a numeric format (csv), the separate PsN tool sumo is used. Therefore, any validation of execute is also a validation of sumo.

```
[execute]
folder=NM_examples          # The folder with the model to run, must be a
                            # subfolder from "lib_folder"
model=run4.mod              # NM model
reference=run4.lst          # The reference NM output file in the lib_folder.
                            # By default equal to NM model filename with
                            # the default NM output extension
nm_version=default          # Overrides default NM version.
extra_files=                # If extra files are reqd for the run, these [x]
```

```
                          # will be copied to the run folder
extra_arguments=          # Optional arguments to specify to PsN [x]
R_script=execute.R        # Validation script (R). Optional, default is:
                          # <psn_command>.R
                          # All R-scripts are in the /R folder.
command_explicit=         # Explicitly specify the whole command line, [x]
                          # which overrides all other settings.
ofv_abs_tol=3             # Absolute tolerance allowed in OFV.
                          # By default, the relative default tolerance is used.
run_psn=0                 # Actually run the PsN command for this tool?
                          # if 0 or FALSE, the command wil not be run
                          # and the test (if implemented) will be performed
                          # using previously generated output (if avail.)
                          # By default 1
run_test=0                # Actually perform the test using the R-script?
                          # By default 1
## possible tests
test_ofv=TRUE
test_parameters=TRUE
## /tests
```

Tip: not all of the above arguments need to be included in the ini-file. A simpler setup could be:

```
[execute]
folder=NM_examples
model=run4.mod
run_psn=1
run_test=1
## tests
test_all=1
## /tests
```

bootstrap

Bootstrap is a tool for calculating bias, standard errors and confidence intervals of parameter estimates. It does so by generating a set of new datasets by sampling individuals with replacement from the original dataset, and fitting the model to each new dataset, see Efron B, An Introduction to the Bootstrap, Chap. & Hall, London UK, 1993. To compute standard errors for all parameters of a model using the non-parametric bootstrap implemented here, roughly 200 model fits are necessary. To assess 95% confidence intervals approximatly 2000 will suffice.

```
[bootstrap]
run_psn=0              # run the bootstrap? If 0, bootstrap output should already be present
run_test=0             # perform the actual validation test in R?
folder=NM_examples     # folder containing the model to be used in the bootstrap
model=run4.mod         # model to use in the bootstrap
reference=bootstrap_dir1 # the bootstrap reference folder, in which a
                       # file called raw_results_<run>.csv and
                       # bootstrap_results.csv should be present.
seed=12345             # seed to be used for randomization
```

```
samples=100             # number of samples to use in the bootstrap
tolerance=0.01          # relative tolerance (for the parameters)
ofv_abs_tol=3           # absolute tolerance (for the OFV)

## tests
test_ofv_mean=1           # test on difference in mean OFV
test_ofv_sd=1             # test on difference in sd OFV
test_parameter_mean=1     # test on difference in mean of parameters
test_parameter_sd=1       # test on difference in sd of parameters
test_bootstrap_ofv=1      # Compare OFVs for every
                          # bootstrap sample with the
                          # reference OFVs. Only makes sense if
                          # -seed is set!
test_bootstrap_parameters=0 # Compare parameters for every
                          # bootstrap sample with the
                          # reference. Only makes sense if
                          # -seed is set!
## /tests
```

vpc / npc

NPC Numerical Predictive Check is a model diagnostics tool. VPC Visual Predictive Check is another closely related diagnostics tool. A set of simulated datasets are generated using the model to be evaluated. Afterwards the real data observations are compared with the distribution of the simulated observations. By default no model estimation is ever performed. The input to the npc script is the model to be evaluated, the number of samples (simulated datasets) to generate, parameter values options for the simulations, and stratification options for the evaluation. It is also possible to skip the simulation step entirely by giving two already created tablefiles as input. The input to the vpc script is the input to npc plus an additional set of options.

Since a large portion of the processing is identical between the scripts, and the numeric output for the vpc and npc commands are the same, a separate validation test for npc is currently not available in ValPsN.

Note: In the current version of PsN (3.5.3), the -seed option is broken. Therefore, it is advised to allow a fairly large tolerance for the tests. For the same reason it may be best to skip the test_pi_median_ci altogether, unless large numbers of simulations are performed.

```
[vpc]
model=run4.mod
folder=NM_examples
run_psn=0
run_test=1
seed=12345
samples=500
reference=vpc_dir1 # the vpc reference folder, in which a
```

```
                  # file called raw_results_<run>*.csv and the file
                  # vpc_results.csv should be present.
tolerance=0.05 # If seed is not used, tolerance should probably be fairly large
extra_arguments=-bin_by_count -no_of_bins=8 -dv=CP

##tests
test_obs_median=1 # tests median for the observed data
test_obs_5=1      # tests 5th pctile of the observed data
test_obs_95=1     # tests 95th pctile of the observed data
test_pi_median=1  # tests median of the simulated data
test_pi_median_ci=1 # tests the ci of the median of the simulated data
test_pi_95=1      # tests the 95th pctile of the simulated data
test_pi_5=1       # tests the 5th pctile of the simulated data
# or: test_all=1  # perform all of the above tests
## /tests
```

cdd

The Case Deletions Diagnostics (cdd) algorithm is a tool primarily used to identify influential components of the dataset, usually individuals. The cdd works by identifying groups in the data set and creating one new data set for each member of the group, where that member has been removed. The model is run once with each new data set. The PsN implementation of the cdd can take any column as base for the grouping and all rows with the same value in that column will be considered a group as long as no individual contain multiple values in that column. One should take care that the grouping creates sensible individual records. PsN will renumber the ID column so that two individuals with the same ID will not end up next to each other.

```
[cdd]
run_psn=0
run_test=1
folder=examples2
model=run4.mod
case_column=CENT
reference=cdd_dir1
ofv_abs_tol=1
## tests
test_jackknife_par_bias=1
test_jackknife_ofv_bias=1
## /tests
```

llp

The Log Likelihood Profiling (llp) tool is used to calculate confidence intervals of parameter values. Without the llp the confidence intervals can be calculated with the standard errors of the parameters under the assumption that the parameter values are normally distributed. The llp, however, makes no assumption of the shape of the distribution. The llp tool will calculate the confidence intervals for any number of parameters in the

model, working with one parameter at a time. By first fitting the original model and then fixing the parameter at values close to the NONMEM estimate, the llp obtains the difference in likelihood between the original model and new, reduced model. The logarithm of the difference in likelihood is χ^2 distributed and when that value is 3.84, the parameter value is at the 95% confidence limit. The search for the limit is done on both sides of the original parameter value, and thus the llp makes no assumption of symmetry or the parameter value distribution

```
[llp]
folder=examples1
model=run4.mod
extra_arguments=-thetas='1,2' -rse_thetas='20,20'
reference=llp_dir1 # the llp reference folder, in which a
                   # file called raw_results_<run>*.csv and the file
                   # llp_results.csv should be present.
run_psn=0
run_test=1
## tests
test_ci=1          # compare confidence intervals
test_ofv=1         # compare OFVs
## /tests
```

scm

The Stepwise Covariate Model (scm) building tool of PsN implements Forward Selection and Backward Elimination of covariates to a model. In short, one model for each relevant parametercovariate relationship is prepared and tested in a univariate manner. In the first step the model that gives the best fit of the data according to some criteria is retained and taken forward to the next step. In the following steps all remaining parameter-covariate combinations are tested until no more covariates meet the criteria for being included into the model. The Forward Selection can be followed by Backward Elimination, which proceeds as the Forward Selection but reversely, using stricter criteria for model improvement. The Stepwise Covariate Model building procedure is run by the PsN tool scm.

```
[scm]
run_psn=0
run_test=0
folder=PSP
model=run4.mod # not required
extra_files=psp.scm
extra_arguments=-config_file=psp.scm
reference=scm_run5
## tests
test_final_model_same=1 # test if final covariate model is the same
test_ofv_final_model=1 # test if OFV for final covariate model is
the same
## /tests
```

xv-scm

Cross-validated scm, xv_scm, depends heavily on the scm program, and all scm options apply also to xv_scm except that options search_direction, gof, p_value, p_forward, p_backward and update_derivatives are ignored. A word of caution: xv_scm produces many files and takes up much disk space. It is wise to delete all the split_X subdirectories once the results are collected.

Configuration currently not covered in manual.

boot-scm

Bootstrap scm, boot_scm, depends heavily on the scm program, and all scm options apply also to boot_scm. Please refer to scm_userguide.pdf for help on scm options.

Configuration currently not covered in manual.

sse

SSE Stochastic Simulation and Estimation is a tool for model comparison and hypothesis testing. First, using a given model, henceforth called the input model, a number of simulated datasets are generated. Then the input model and a set of alternative models are fitted to the simulated data. Finally, a set of statistical measures are computed for the parameter estimates and objective function values of the various models.

Configuration currently not covered in manual.

mcmp

The Monte Carlo Mapped Power (MCMP) method provides a fast and accurate prediction of the power and sample size relationship. Efficient power calculation methods have been suggested for Wald test-based inference in mixed-effects models but the only available alternative for Likelihood ratio test-based hypothesis testing has been to perform computer-intensive multiple simulations and re-estimations. The MCMP method is based on the use of the difference in individual objective function values (ΔiOFV) derived from a large dataset simulated from a full model and subsequently re-estimated with the full and reduced models.

See Vong C. et al, AAPSJ 2012 Jun; 14(2):176-86.

Configuration currently not covered in manual.

lasso

Covariate models for population pharmacokinetics and pharmacodynamics are often built with a stepwise covariate modelling procedure (SCM, available in PsN). When analysing a small dataset this method may produce a covariate model that suffers from selection bias and poor predictive performance. The lasso is a method suggested to remedy these problems. It may also be faster than SCM and provide a validation of the covariate model. In the lasso all covariates must be standardised to have zero mean and standard deviation one. Subsequently, the model containing all potential covariate parameter relations is fitted with a restriction: the sum of the absolute covariate coefficients must be smaller than a value, t. The restriction will force some coefficients towards zero while the others are estimated with shrinkage. This means in practice that when fitting the model the covariate relations are tested for inclusion at the same time as the included relations are estimated. For a given SCM analysis, the model size depends on the P-value required for selection. In the lasso the model size instead depends on the value of t which can be estimated using cross-validation.

```
[lasso]
run_psn=0
run_test=1
folder=examples2
model=run4.mod
lst=run4.lst
relations=CL:WT-2,AGE-2,SEX-1,CENT-1,,V1:WT-2,AGE-2,SEX-1,CENT-1
reference=lasso_dir1
## tests
test_coef=1 # compare the coefficients for the covariate relationships
## /tests
```

3.4 Custom R scripts

While default R scripts are provided with *ValPsN* that perform validations for output from most PsN commands, custom R scripts can be used as well to perform the validation. If additional tests are required, it is probably easiest to use the existing default R-script as starging point. Below is some guidance on how the R-scripts are implemented within the *ValPsN* framework.

First, in the configuration (ini-) file, the custom R-script needs to be specified:

```
[execute]
model=run4.mod
reference=run4.lst
...
R_script=execute_custom.R
...
```

In which it is assumed that `execute_custom.R` is located within the R-folder specified in the [folder] settings.

When the script is invoked, a copy of the script is made, to which a header is prepended automatically which specifies the current folder and the validation settings, e.g. :

```
cwd <- "/Users/username/ValPsn/runs/PsN_3.5.3_20120930/test1_execute"
setwd(cwd)
args <- list (
  command_explicit = "",
  extra_arguments = "",
  extra_files = "",
  folder = "examples1",
  model = "run4.mod",
  nm_mod_extension = "mod",
  nm_out_extension = "lst",
  nm_output = "execute_test_dir/NM_run1/psn.lst",
  nm_version = "default",
  ofv_abs_tol = "1",
  R_script = "execute.R",
  reference = "run4.lst",
  run_psn = "0",
  run_test = "1",
  test_ofv = 1,
  test_parameters = 1,
  threads = "5",
  tolerance = "0.01",
  verbose_level = "2"
)
```

So in the R-script, the settings in the `args` list-object can be used to customize the validation. *ValPsN* will also copy the required result files from the reference output to the current folder (for most toolkit commands these are the csv-files, but for some also log-files and raw results files are copied). So if both the test and reference output are imported into R, any test can be performed on these data. Please have a look in the

default R scripts how this is implemented. The validation framework requires that at the end of the R-script, R needs to report back whether the test succeeded or failed, which is implemented e.g. as follows:

```
## Write overall test succes
if (all_res) {
  cat ("Overall test result: SUCCESS\n")
} else {
  cat ("Overall test result: FAILED!\n")
}
```

in which `all_ref` is a boolean variable specifying whether the test was successful or not. The syntax of the output should be exactly as specified here to be registered by the framework.