NONMEM Bayesian Modeling with bbr.bayes : : CHEAT SHEET

Setup

bbr.bayes is an extension to the bbr package for Bayesian modeling. First you need to set up bbr, as described in bbr's "Getting Started" vignette.

Make sure this is in your .Rprofile:

```
options("bbr.bbi_exe_path" = file.path(getwd(), "bin", "bbi"))
```

Run this once to install bbi to the path set with bbr.bbi_exe_path:

```
bbr::use_bbi()
```

Run this once per modeling directory to create a bbi.yaml:

```
bbr::bbi_init( "path/to/model/dir",
  .nonmem_dir = "/opt/NONMEM", # location of nonmem
  .nonmem_version = "nm75")    # nonmem version
```

Load both the core bbr package and the bbr.bayes package.

```
library(bbr)

library(bbr.bayes)
```

Creating a Model

To submit or interact with models in bbr you need a model object. If you have a .ctl file, you can create an nmbayes model object using new_model(). The example below expects to find a control stream file in *model/nonmem/1.ctl*.

```
MODEL_DIR <- here::here("model/nonmem")

mod1 <- new_model(file.path(MODEL_DIR, 1),

 .model_type = "nmbayes")
```

If you previously created a model, read it into a model object using read_model().

```
mod2 <- read_model(file.path(MODEL_DIR, 2))
```

Use a non-Bayes model as the starting point

copy_model_as_nmbayes() copies a model of type "nonmem" to a new model of type "nmbayes".

```
mod3 <- copy_model_as_nmbayes(
  .parent_mod = mod2,  .new_model = 3)
```

Submitting a Model

Submit a model to be run

```
mod1 %>% submit_model()
```

Additional basic commands:

Overwrite output from a previous run

```
submit_model(mod1, .overwrite = TRUE)
```

Underneath, submit_model() creates and executes a submodel for each chain. See ?bbr_nmbayes for details.

Monitor the running models:

```
tail_lst(mod1)
tail_output(mod1)
```

Parallelization (see bbi.yaml sidebar, next page)

```
submit_model(
 mod1, .bbi_args = list(parallel = TRUE, threads = 8))
```

print_bbi_args() gives a complete list of .bbi_args

Extract posterior draws

read_fit_model() returns a posterior draws object with samples of a completed model.

```
draws1 <- read_fit_model(mod1)
```

You can also use the posterior::as_draws() methods:

```
draws1 <- posterior::as_draws(mod1)
draws_df1 <- posterior::as_draws_df(mod1)
```

Working with Draws Objects

Once you have a draws object, you can use any of posterior's functions.

```
posterior::nchains(draws1)
#> 4
posterior::subset_draws(draws1, variable = "THETA") %>%
 posterior::summarize_draws("median", "quantile2", "rhat", "ess_tail")
#> # A tibble: 8 × 6
#>   variable  median    q5   q95  rhat ess_tail
#>   <chr>      <dbl> <dbl> <dbl> <dbl>    <dbl>
#> 1 THETA[1]  0.239  -0.128 0.406 1.72     24.2
#> 2 THETA[2]  4.00    3.80  4.07  1.68     18.8
#> …
```

Generating Diagnostics

nm_join_bayes() feeds samples to an mrgsolve model to generate quantities like EPRED and IPRED.

```
mod_ms <- mrgsolve::mread(...)
res <- nm_join_bayes(mod, mod_ms)
dplyr::select(res, NUM, TIME, IPRED, NPDE)
#> # A tibble: 4,292 × 4
#>       NUM  TIME IPRED   NPDE
#>     <dbl> <dbl> <dbl>  <dbl>
#>  1      1  0      NA     NA
#>  2      2  0.61  67.5   0.126
#>  …
```

METRUM RESEARCH GROUP

Example Workflow

Copy initial NONMEM Bayes model

```
mod10 <- copy_model_as_nmbayesl(
  .parent_mod = mod1, .new_model = 10,
  .inherit_tags = TRUE) %>%
  replace_tag("FOCE", "BAYES")
```

Submit model

```
submit_model(mod10)
```

View model results

```
library(posterior)
draws <- as_draws(mod10)
# MCMC diagnostics
subset_draws(draws,
  variable = c("THETA", "OMEGA", "SIGMA")) %>%
  summarize_draws()
# model diagnostics
```

Add notes

```
mod10 <- mod10 %>%
  add_notes("Low ESS for some parameters.")
```

Create new model based on initial model

```
mod20 <- copy_model_from(
  mod10, 20, .inherit_tags = TRUE)
```

Open .ctl and edit

```
open_model_file(mod20)
```

Modify tags of new model

```
mod20 <- mod20 %>%
   replace_tags("BAYES", "NUTS")
```

Compare changes to parent model

```
model_diff(mod20)  # compare control streams
tags_diff(mod20)   # compare tags
```

Submit new model

```
submit_model(mod20)
```

View new model results and add notes

```
mod20 <- mod20 %>%
   add_notes("All diagnostics look reasonable.")
```

Tags and Notes

The model object has tags and notes fields, to annotate the Model during development. Tags are concise and describe model structure, while notes are free form text to notate decisions and observations. Both are meant to replace descriptions in the $PROB of your control stream.

Note: when adding or modifying these attributes, **you must reassign the modified model object**. You can pipe several modifications together.

```
mod <- mod %>%
   add_tags("ETA-V2") %>%
   replace_tags("Prop RUV", "Add RUV") %>%
   add_notes("First model to use additive error")
```

Modifying the model object

Helper functions exist to add, replace, or remove the tags, notes, description, based_on, and bbi_args fields.

```
mod <- mod %>% add_description("Base model")
mod <- mod %>% replace_based_on("1001", "1002")
mod <- mod %>% add_bbi_args(list(parallel = TRUE))
```

Creating a Run Log

Call run_log() to return a tibble summarizing all model under the specified model directory.

```
run_log(MODEL_DIR) %>%
  collapse_to_string(tags, notes) %>%
  dplyr::select(run, model_type, tags, notes)
#> # A tibble: 3 × 4
#>    run   model_type tags          notes
#>    <chr> <chr>      <chr>         <chr>
#> 1 1      nonmem     ETA-CL, FOCE  <NA>
#> 2 10     nmbayes    ETA-CL, BAYES Low ESS for some pa…
#> 3 20     nmbayes    ETA-CL, NUTS  All diagnostics loo…
```

Checking model/data are up to date

Pass a model object or run log tibble to check_up_to_date() to verify none of the control streams or data files on disk have changed since the models were run.

Or use config_log() or add_config(), which contain model_has_changed and data_has_changed columns.

Global Settings in bbi.yaml

The modeling directory includes a bbi.yaml file containing defaults for many configuration options. Pass .bbi_args to submit_model() to override these for a particular run.

```
// bbi.yaml
...
overwrite: true
parallel: true
threads: 8
```

Path Helper Functions

```
get_model_path(mod)
get_data_path(mod)

# Get paths to {name}-{chain}.lst
# files in chain submodels
chain_paths(mod,
  extension = "lst"))

# Get paths to {name}.tab files.
chain_paths(mod,
  name = get_model_id(mod),
  extension = "tab")
```