Package ‘NMproject’

October 12, 2022

Type Package

Title Script Based 'NONMEM' Model Development

URL https://tsahota.github.io/NMproject/,
https://github.com/tsahota/NMproject

BugReports https://github.com/tsahota/NMproject/issues

Version 0.6.9

Maintainer Tarj Sahota <t.sahota0@gmail.com>

Description Industrialisation of 'NONMEM'
<https://www.iconplc.com/innovation/nonmem/> via fully and rapidly reusable
model development 'workflows' entirely within 'RStudio'. Quickly get started
with new models by importing 'NONMEM' templates from the built-in code
library. Manipulate 'NONMEM' code from within R either via the tracked
'manual edit' interface or 'programmatically' via convenience functions.
Script 'workflows' by piping sequences of model building steps from control
file creation, to execution, to post-processing and evaluation. Run caching
makes 'workflows' R markdown friendly for easy documentation of thoughts and
modelling decisions alongside executable code. Share, reuse and recycle
'workflows' for new problems.

License GPL (>= 3)

Encoding UTF-8

Imports shiny, dygraphs, DT, git2r (>= 0.18.0), dplyr (>= 0.7.2),
methods, stringr (>= 1.3.1), rlang (>= 0.2.1), diffobj (>=
0.1.11), tidyr (>= 1.0.0), miniUI (>= 0.1.1), rstudioapi (>=
0.7), crayon (>= 1.3.4), rprojroot, htmltools, rmarkdown,
magrittr, usethis, lifecycle (>= 1.0.0)

RoxygenNote 7.1.1

Suggests testthat, knitr, ggplot2, data.tree, xpose, lubridate,
pmxTools, Hmisc, rsample, renv, desc, purrr, digest, covr,
devtools, roxygen2, xfun

Collate 'NMproject-options.R' 'NMproject-package.R' 'Vectorize.R'
'addin-apps.R' 'apply_manual_edit.R' 'basic-ctl-manipulation.R'

'utils.R' 'basic_methods.R' 'boot.R' 'nm_object.R' 'cache.R' 'check_installation.R' 'covariate-explore.R' 'cran_note_handle.R' 'data_filter.R' 'decision.R' 'deprecated.R' 'derived-data-prep.R' 'developer-funs.R' 'directory-management.R' 'dollar_input.R' 'find-nonmem.R' 'import-code.R' 'init_funs.R' 'input_data.R' 'job_stats.R' 'low-level-ctl-handling-funs.R' 'make_OCC_every_dose.R' 'make_project.R' 'manual-edit.R' 'monitoring.R' 'nm-gettersetters.R' 'nm_diff.R' 'nm_read_table.R' 'nm_render.R' 'nm_tran.R' 'nm_tree.R' 'nmsave.R' 'ofv.R' 'output_table.R' 'param-manipulate.R' 'plot_iter.R' 'post-run-summaries.R' 'prompt_overwrite.R' 'psn_style_scm.R' 'read_ext.R' 'rmd_to_vignette.R' 'run_nm.R' 'shiny-apps.R' 'show-file.R' 'sim-diagnostics.R' 'snippet_setup.R' 'stepwise-covariate.R' 'subroutine.R' 'system-hooks.R' 'text.R' 'todo.R' 'update_parameters.R' 'with_temp_git_config.R' 'zzz.R'

NeedsCompilation no

Author Tarj Sahota [aut, cre, cph],
AstraZeneca [cph],
Nuria Buil Bruna [ctb],
Stein Schalkwijk [ctb]

Repository CRAN

Date/Publication 2022-09-29 22:40:02 UTC

R topics documented:

add_mixed_param *Add a mixed effect parameter to $PK (or $PRED)*

Description

[Stable]

Primarily an internal function. This will (by default) add a parameter (mixed effect) to your code $PK/$PRED and $THETA/$OMEGA.

Usage

```
add_mixed_param(
  m,
  name,
  init = 1,
  unit = "",
  trans = c("LOG"),
  position = NA_integer_,
  after = character()
)
```

Arguments

`m`	An nm object.
`name`	Character. Name of NONMEM variable to create.
`init`	Numeric (default = `1`). Initial value of fixed effect.
`unit`	Character (default = `""`). Unit of variable.
`trans`	Character (default = `"LOG"`). Transformation of the variable.
`position`	Integer. Not used.
`after`	Character. Pattern to match and include the mixed effect after.

Value

An nm object with modified `ctl_contents` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 %>% dollar("PK")
m1 %>% dollar("THETA")
```

```
m1 <- m1 %>% add_mixed_param("ALAG1", init = 1.1, unit = "h", trans = "LOG")

m1 %>% dollar("PK")
m1 %>% dollar("THETA")
```

add_remove_covs *Add/remove a covariate to a NONMEM model*

Description

[Stable]

Follows PsN coding conventions to add covariates into a model. The advantage is no need to create a .scm file, just directly modify the NONMEM control file contents. This function is used by `covariate_step_tibble()` for stepwise covariate model development.

Usage

```
add_cov(
  ctl,
  param,
  cov,
  state = 2,
  continuous = TRUE,
  time_varying,
  additional_state_text = list(),
  id_var = "ID",
  force = FALSE,
  force_TV_var = FALSE,
  init,
  lower,
  upper
)

remove_cov(
  ctl,
  param,
  cov,
  state = 2,
  continuous = TRUE,
  time_varying,
  id_var = "ID"
)
```

Arguments

`ctl`	An nm object or an object coercible to `ctl_list`.
`param`	Character. Name of parameter.
`cov`	Character. Name of covariate.
`state`	Numeric or character. Number or name of state (see details).
`continuous`	Logical (default = `TRUE`). Is covariate continuous?
`time_varying`	Optional logical. is the covariate time varying?
`additional_state_text`	Optional character (default = empty). Custom state variable to be passed to `param_cov_text`.
`id_var`	Character (default = `"ID"`). Needed if time_varying is missing.
`force`	Logical (default = ‘FALSE“). Force covariate in even if missing values found.
`force_TV_var`	Logical (default = `FALSE`). Force covariates only on `TV` notation parameters.
`init`	Optional numeric/character vector. Initial estimate of additional parameters.
`lower`	Optional numeric/character vector. lower bound of additional parameters.
`upper`	Optional numeric/character vector. Upper bound of additional parameters.

Details

Available `states`:

"2" or "linear" PARCOV= (1 + THETA(1)*(COV -median))

"3" or "hockey-stick" IF(COV.LE.median) PARCOV = (1 + THETA(1)*(COV - median)) IF(COV.GT.median) PARCOV = (1 + THETA(2)*(COV - median))

"4" or "exponential" PARCOV= EXP(THETA(1)*(COV - median))

"5" or "power" PARCOV= ((COV/median)**THETA(1))

"power1" PARCOV= ((COV/median))

"power0.75" PARCOV= ((COV/median)**0.75)

"6" or "log-linear" PARCOV= (1 + THETA(1)*(LOG(COV) - log(median)))

`remove_cov` only works with covariates added with `add_cov`.

Value

An nm object with modified `ctl_contents` field.

See Also

`covariate_step_tibble()`, `test_relations()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

temp_data_file <- paste0(tempfile(), ".csv")

## dataset has missing WTs so create a new one and assign this to the run
input_data(m1) %>%
  dplyr::group_by(ID) %>%
  dplyr::mutate(WT = na.omit(WT)) %>%
  write_derived_data(temp_data_file)

m1 <- m1 %>% data_path(temp_data_file)

m1WT <- m1 %>% child("m1WT") %>%
  add_cov(param = "V", cov = "WT", state = "power")

m1 %>% dollar("PK")
m1WT %>% dollar("PK")  ## notice SCM style code added

nm_diff(m1WT)

## Not run:
run_nm(c(m1, m1WT))
rr(c(m1, m1WT))
summary_wide(c(m1, m1WT))

## End(Not run)

unlink(temp_data_file)


# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

temp_data_file <- paste0(tempfile(), ".csv")

## dataset has missing WTs so create a new one and assign this to the run
input_data(m1) %>%
  dplyr::group_by(ID) %>%
  dplyr::mutate(WT = na.omit(WT)) %>%
  write_derived_data(temp_data_file)

m1 <- m1 %>% data_path(temp_data_file)
```

```
m1WT <- m1 %>% child("m1WT") %>%
  add_cov(param = "V", cov = "WT", state = "power")

m1 %>% dollar("PK")
m1WT %>% dollar("PK")  ## notice SCM style code added

## reverse this by removing WT

m1noWT <- m1WT %>% child("m1noWT") %>%
  remove_cov(param = "V", cov = "WT")

m1noWT %>% dollar("PK")
m1noWT %>% dollar("THETA")

unlink(temp_data_file)
```

append_nonmem_var *Include NONMEM variables in output table*

Description

[Experimental]

This is designed to be used in situations where you don't want to rerun NONMEM, but need a variable defined in the control file. This will parse the `$PK`/'$PRED" and compute it as an additional row in R. Safest way is to just rerun the model with the variable in $TABLE, but this is for those who are too time constrained. It is advisable to QC the output.

Usage

```
append_nonmem_var(output_table, r, var)
```

Arguments

`output_table`	Output from `output_table()`.
`r`	An nm object.
`var`	Character. Name of variable to extract (needs to be defined in $PK/$PRED).

Value

A modified version of `output_table` with addition `var` column.

`apply_manual_edit` *Apply a manual edit patch*

Description

[Stable]

It is best to allow the "manual edit" RStudio 'Addin' to write this function in your script for you. After a tracked manual edit is performed, a patch file is created and saved in the "patches" subdirectory of `nm_dir("models")`. This function applies the patch to the object.

Usage

```
apply_manual_edit(m, patch_id, return_merge_conf_ctl = FALSE)
```

Arguments

`m`	An nm object.
`patch_id`	Character name of patch. Corresponds to the file name in the "patches" subdirectory of `nm_dir("models")`.
`return_merge_conf_ctl`	Logical (default = `FALSE`). If there a merge conflict produced, should the ctl file be returned?

Details

Generally best to to apply patches before automatic edits and changes in directories e.g. via `run_in()`. If patches are applied to NONMEM control file sections that are likely to change in the future, the patch may fail to apply. In this case, it is best to view the patch (via the "view patch" RStudio 'Addin') and manually re-implement the changes again in a new manual edit.

Value

An nm object with modified `ctl_contents` field.

`bind_covariate_results` *Add run results into a covariate tibble*

Description

[Stable]

Extracts results from completed covariate runs and combines them into the covariate `tibble()`.

The goal of NMproject's covariate modelling functions is to provide a stepwise covariate method *with manual decision* making. This important to ensure that the full model selection/evaluation criteria (should be defined in statistical analysis plans) can be applied at every step rather than just log likelihood ratio testing, where the most significant model may be unstable, may worsen model predictions or may only be slightly more significant than a more physiologically plausible covariate relationship.

The functions `test_relations()`, `covariate_step_tibble()`, `bind_covariate_results()` together comprise NMproject stepwise covariate method with manual decision. The goal is to be part way between PsN's SCM and completely manual process at each forward and backward elimination step. The syntax of how covariates are included is the same as PsN's SCM routine - See PsN documentation for more information.

Usage

```
bind_covariate_results(dsc, nm_col = "m", parameters = "new")
```

Arguments

`dsc`	An output `tibble` from `covariate_step_tibble()`.
`nm_col`	Character (default = `"m"`). Name of column to store nm objects.
`parameters`	Character (default = `"new"`). Passed to `summary_wide()`.

Value

An modified version of `dsc` with additional columns from `summary_wide()` for model selection purposes.

See Also

`covariate_step_tibble()` and `nm_render()` for rendering diagnostic reports for (subsets of) models in `nm_col`.

Examples

```
## requires NONMEM to be installed
## Not run:
## create tibble of covariate step with model objects as column m
dsm1 <- m1 %>% covariate_step_tibble(
  run_id = "m1_f1",
  dtest = dtest,
  direction = "forward"
)

## run all models greedily
```

```
dsm1$m <- dsm1$m %>% run_nm()

wait_finish(dsm1$m)

## extract results and put into tibble
dsm1 <- dsm1 %>% bind_covariate_results()

## plot goodness of fit diagnostics top 3 models (in terms of p-value)
dsm1$m[1:3] %>% nm_render("Scripts/basic_gof.Rmd")


## End(Not run)
```

block-omega-sigma *Create or remove $OMEGA/$SIGMA BLOCKs*

Description

[Stable]

Manipulate $OMEGA (and $SIGMA) BLOCKs to introduce or remove correlations.

Usage

```
block(iomega, eta_numbers = NA, diag_init = 0.01)

unblock(iomega, eta_numbers)
```

Arguments

`iomega`	A `tibble` output from `init_omega()` or `init_sigma()`.
`eta_numbers`	Numeric vector. ETA numbers to put into a block or unblock for `block()` and `unblock()`, respectively. Must be contiguous.
`diag_init`	Numeric. Default value for off diagonal elements.

Value

An nm object with modified `ctl_contents` field.

See Also

`init_theta()`, `init_omega()`, `init_sigma()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

io <- m1 %>% init_omega()
io <- io %>% block(c(2, 3))
m1 <- m1 %>% init_omega(io)
m1 %>% dollar("OMEGA") ## to display $OMEGA


# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

## first create a block
io <- m1 %>% init_omega()
io <- io %>% block(c(2, 3))
m1 <- m1 %>% init_omega(io)
m1 %>% dollar("OMEGA") ## to display $OMEGA

## now unblock
io <- io %>% unblock(c(2, 3))
m1 <- m1 %>% init_omega(io)
m1 %>% dollar("OMEGA") ## to display $OMEGA
```

---

check_installation *Check NMproject installation*

Description

This function uses `testthat` to test whether pre-requisites are met. It first checks for availability of PsN via `system_nm()`, this is a requirement for NMproject. Subsequent checks are not mandatory, but recommended for full functionality - these comprise a check to see if NMTRAN checks are configured and a check to see if code completion has been set up.

Usage

```
check_installation()
```

Value

Logical TRUE or FALSE indicating whether the tests have succeeded or not. A test failure does not necessarily mean that NMproject incorrectly configured. Test messages will say whether they are needed or just recommended to pass.

child *Make child nm object from parent*

Description

[Stable]

Child objects inherit attributes of parent but with a new run_id. The control file will be inherited too with $TABLEs updated.

Usage

```
child(m, run_id = NA_character_, type = "execute", parent = m, silent = FALSE)
```

Arguments

m	Parent nm object.
run_id	Character. New run_id to assign to child object.
type	Character (default = "execute"). Type of child object.
parent	Optional nm object (default = m) . Parent object will by default be m, but this argument will force parent to be a different object.
silent	Logical (default = FALSE). Should warn if conflicts detected.

Details

Specifying parent will force parent to be different from m. This is useful in piping when a parent object is modified prior to being used in the child object.

Value

An new nm object with modified parent_* fields updated to be the * fields of the parent object, m.

Examples

```
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m2 <- m1 %>% child("m2")

nm_diff(m2, m1)
```

`code_library` *Code Library*

Description

Function not designed for direct use. Instead use the RStudio `code library` entry on the RStudio 'Addins' menu. This will open the shiny app. Select the file, and click "preview" to view and `import()` to bring into the "staging" area of your project. See vignette at `https://tsahota.github.io/NMproject/` for a video showing use of the app. NONMEM control files will intentionally not be imported straight in the "Models" directory and instead go into "staging/Models". This staging location can be referred to when creating `nm` objects with `new_nm(..., based_on = "staging/Models/[filename]")`.

Usage

```
code_library(
  extn = NULL,
  fields = "Description",
  viewer = TRUE,
  silent = FALSE,
  return_info = FALSE
)
```

Arguments

`extn`	Vector string of extensions to include (default = `NULL` includes all).
`fields`	Character vector of fields to extract.
`viewer`	Logical indicating if viewer should be used to display results (default=`FALSE`).
`silent`	Logical indicating if messages should be silenced (default=`FALSE`).
`return_info`	Logical (default = `FALSE`). Return data.frame of results (FALSE= returns file paths).

Details

Requires `getOption("code_library_path")` to be set.

Value

If `return_info = TRUE`, invisibly returns output a `tibble` with code library information. Otherwise (this may be deprecated soon), will return paths to code library files.

See Also

`ls_code_library()`, `stage()`, `import()`

Examples

```
code_library(viewer = FALSE, return_info = TRUE)
```

`coef_widelong` *Extract parameter values*

Description

[Stable]

Pulls parameters, standard errors, OFVs and condition numbers out of ext files, applies transformations. This function is useful when numeric values are needed. `rr` is easier to read, however it returns characters. A wide and long format is available via two different functions.

Usage

```
coef_wide(m, trans = TRUE)

coef_long(m, trans = TRUE)
```

Arguments

`m`	An nm object.
`trans`	Logical (default = `TRUE`). Transform parameters using comments in $THETA/$OMEGA/$SIGMA.

Value

`data.frame` of extracted model parameter values. `coef_wide()` returns a `data.frame` in wide format. Vector valued objects `m`, will be stacked vertically with one row per run. `coef_long()` returns a `data.frame` in long format. Vector valued objects `m`, will be stacked horizontally.

NONMEM coding conventions used by NMproject

The convention for $THETA comments used by NMproject is `value ; name ; unit ; transformation`

e.g. `$THETA 0.1    ; KA ; h-1 ; LOG`

The options for THETA transformations are: `LOG`, `LOGIT`, `RATIO` and missing. `LOG` and `LOGIT` refer to log and logit transformed THETAs, respectively where the parameters should be back-transformed for reporting. `RATIO` refers to ratio data types, i.e. parameters that are positive and have a meaningful zero. Most parameters like KA, CL, EMAX fall into this category, but covariates effects which can go negative do not. RSEs are calculated for ratio data. Missing transformations are suitable for all other parameters, here no RSEs will be calculated, only raw SE values will be reported.

The convention for $OMEGA is similar but without a unit item: `value ; name ; transformation`

e.g. `$OMEGA 0.1    ; IIV_KA ; LOG`

The options for OMEGA are either `LOG` or missing. `LOG` indicating that the individual parameter distribution is log normally distributions and should be reported as a CV% (and associated RSE%) rather than as the raw NONMEM estimate.

The convention for $OMEGA is just : `value ; name`.

THETA transformations using `trans = TRUE`

The value of FINAL and RSE% (always accompanied with a `%` symbol in outputs) in the returned `tibble` is the reported standard error (where applicable) where θ and $se(\theta)$ are the NONMEM reported values of parameters and standard errors, respectively:

`LOG` $FINAL = exp(\theta), RSE = 100\sqrt{(}exp(se(\theta)^2) - 1)$

`RATIO` $FINAL = \theta, RSE = 100se(\theta)/\theta$

`LOGIT` $FINAL = 100/(1 + exp(-\theta)), SE = se(\theta)$

missing $FINAL = \theta, SE = se(\theta)$

OMEGA transformations using `trans = TRUE`

The value of FINAL and RSE% (always accompanied with a `%` symbol in outputs) in the returned `tibble` is the reported standard error (where applicable) where ω^2 and $se(\omega^2)$ are the NONMEM reported values of parameters and standard errors, respectively

`LOG` $FINAL = 100\sqrt{(}exp(\omega^2) - 1), RSE = 100(se(\omega^2)/\omega^2)/2$

missing $FINAL = \omega^2, SE = se(\omega^2)$

SIGMA transformations using `trans = TRUE`

The value of FINAL and RSE% (always accompanied with a `%` symbol in outputs) in the returned `tibble` is the reported standard error (where applicable) where σ^2 and $se(\sigma^2)$ are the NONMEM reported values of parameters and standard errors, respectively. All sigmas are reported as standard deviations.

all sigmas $FINAL = \sqrt{(}\sigma^2), RSE = 100se(\sigma^2)/\sigma^2$

See Also

`rr()`

comment_lines *Comment and uncomment lines of control file*

Description

[Stable]

Comment out lines of code with that are matched by a patter string.

Usage

```
comment_out(m, pattern = ".*")

uncomment(m, pattern = ".*")
```

Arguments

`m`	An nm object.
`pattern`	Character regex. Passed to `gsub()`.

Value

An nm object with modified `ctl_contents` field.

See Also

`gsub_ctl()`, `target()`

completed_nm *Create an nm object from an already completed PsN run*

Description

[Experimental]

This is useful if the run was made outside of NMproject and we want to leverage NMproject to do postprocessing and results extraction. It is not recommended to apply `run_nm()` or `write_ctl()` to this object as that would break reproducibility

Usage

```
completed_nm(mod_file, data_path = data_name(mod_file), force = FALSE)
```

Arguments

`mod_file`	Path to a model file
`data_path`	Optional path to the corresponding data set
`force`	Passed to `new_nm()`.

Value

An nm object with class "completed_nm".

`cond_num` *Condition number of run*

Description

[Stable]

Extracts condition number from .ext file.

Usage

```
cond_num(r)
```

Arguments

`r`	An nm object.

Value

The `numeric` value of the condition number.

See Also

`ofv()`, `rr()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

is_finished(m1) ## FALSE until run is completed
cond_num(m1) ## NA until m1 is finished
```

`convert_to_simulation` *Convert a NONMEM run to a simulation*

Description

[Stable]

Replaces $EST with $SIM.

Usage

```
convert_to_simulation(m, seed = 12345, subpr = 1)
```

Arguments

`m`	A nm object.
`seed`	Numeric (default = `12345`). seed value to include in $SIM.
`subpr`	Numeric (default = `1`). SUBPR value to include in $SIM.

Details

Will only change $EST/$SIM, therefore it will not be sufficient to change a categorical estimation control file to simulation. You will likely need to perform a `manual edit` for categorical data simulation.

Value

An nm object with modified `ctl_contents` field.

Examples

```
## Not run:

## requires NONMEM to be installed

m1s <- m1 %>%
  child(run_id = "m1s") %>%
  update_parameters(m1) %>%
  convert_to_simulation(subpr = 50) %>%
  run_nm()

m1s %>% nm_render("Scripts/basic_vpc.Rmd")
m1s %>% nm_render("Scripts/basic_ppc.Rmd")


## End(Not run)
```

`covariance_matrix` *Get covariance matrix*

Description

[Experimental]

Usage

```
covariance_matrix(m)
```

Arguments

`m`	An nm object.

Details

Extracts covariance matrix

Value

A `matrix` with parameter correlations.

Examples

```
## requires NONMEM to be installed
## Not run:

covariance_matrix(m)


## End(Not run)
```

`covariance_plot` *Plot $COV matrix*

Description

[Stable]

Usage

```
covariance_plot(r, trans = TRUE)
```

Arguments

`r`	An nm object.
`trans`	Logical (default = TRUE). Applies the transformations specified in $THETA/$OMEGA/$SIGMA comments before plotting.

Details

Plots the correlation plot from the $COV NONMEM output.

Value

A `ggplot2` object with parameter correlations.

See Also

`nm_render()`

Examples

```
## requires NONMEM to be installed
## Not run:

covariance_plot(m1)


## End(Not run)
```

`covariate_step_tibble` *Prepare forward covariate step*

Description

[Stable]

Takes a base nm object and a set of relationships to test (from `test_relations()`) and prepares a `tibble` of NONMEM runs.

The goal of NMproject's covariate modelling functions is to provide a stepwise covariate method *with manual decision* making. This important to ensure that the full model selection/evaluation criteria (should be defined in statistical analysis plans) can be applied at every step rather than just log likelihood ratio testing, where the most significant model may be unstable, may worsen model predictions or may only be slightly more significant than a more physiologically plausible covariate relationship.

The functions `test_relations()`, `covariate_step_tibble()`, `bind_covariate_results()` together comprise NMproject stepwise covariate method with manual decision. The goal is to be part way between PsN's SCM and completely manual process at each forward and backward elimination step. The syntax of how covariates are included is the same as PsN's SCM routine - See PsN documentation for more information.

Usage

```
covariate_step_tibble(
  base,
  run_id,
  run_in = nm_dir("models"),
  dtest,
  direction = c("forward", "backward"),
  ...
)
```

Arguments

`base`	An nm object.
`run_id`	Base run_id to construct run_ids of covariate runs.
`run_in`	Character. See `run_in()`.
`dtest`	`dplyr::tibble` with testing relations (from `test_relations()`).
`direction`	Character. `"forward"` (default) or `"backward"`.
`...`	Additional arguments passed to `add_cov()`.

Value

Will return `dtest` a `dplyr::tibble` with appended columns.

See Also

`test_relations()`, `bind_covariate_results()`, `add_cov()`

Examples

```
dtest <- test_relations(param = c("KA", "K", "V"),
                        cov = c("LIN1", "LIN2", "LIN3", "RND1", "RND2", "RND3"),
                        state = c("linear", "power"),
                        continuous = TRUE) %>%
         test_relations(param = c("KA", "K", "V"),
                        cov = "BN1",
                        state = "linear",
                        continuous = FALSE)

# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

temp_data_file <- paste0(tempfile(), ".csv")

## dataset has missing WTs so create a new one and assign this to the run
input_data(m1) %>%
  dplyr::group_by(ID) %>%
  dplyr::mutate(WT = na.omit(WT)) %>%
  write_derived_data(temp_data_file)

m1 <- m1 %>% data_path(temp_data_file)

dtest <- test_relations(param = c("K", "V"),
                        cov = c("WT"),
                        state = c("linear", "power"),
                        continuous = TRUE)

## requires NONMEM to be installed
## Not run:
## create tibble of covariate step with model objects as column m
dsm1 <- m1 %>% covariate_step_tibble(run_id = "m1_f1",
                                     dtest = dtest,
                                     direction = "forward")

## run all models greedily
dsm1$m <- dsm1$m %>% run_nm()

## extract results and put into tibble
dsm1 <- dsm1 %>% bind_covariate_results()

## sort by BIC (for example) and view
dsm1 <- dsm1 %>% arrange(BIC)
dsm1

## check condition number, covariance,...
## run any diagnostics here

## when happy with selection, select run for subsequent step

m1_f1 <- dsm1$m[1] ## select most signifcant BIC
# alternative select by relationship
m1_f1 <- dsm1 %>%
  filter(param = "CL", cov = "BWT", state = "power") %$%
  m

## do next forward step

dsm2 <- m1_f1 %>% covariate_step_tibble(run_id = "m1_f2",
                                        dtest = dtest,
                                        direction = "forward")

## continue ...

## End(Not run)
```

cov_cov_plot *Plot correlation between two covariates*

Description

[Stable]

Useful for exploratory plots.

Usage

```
cov_cov_plot(
  d,
  cov,
  continuous,
  log_transform_plot = rep(FALSE, length(cov)),
  dcov_info,
  by = "ID"
)
```

Arguments

`d`	Dataset with covariates.
`cov`	Vector of length 2 for covariate names.
`continuous`	Logical vector of length 2 for whether cov is continuous or not.
`log_transform_plot`	Should plot be log transformed or not.
`dcov_info`	Optional `data.frame` with covariate information.
`by`	Character (default = `"ID"`) variable to split over.

Value

`ggplot2` object displaying covariate-covariate correlations.

`cov_forest_data` *Produce dataset for covariate forest plotting*

Description

[Stable]

The main workhorse for computing uncertainty quantiles of covariate effects in different subpopulations.

Usage

```
cov_forest_data(m, covariate_scenarios)
```

Arguments

`m`	An nm object.
`covariate_scenarios`	A `data.frame`. Need columns `cov`, `value` and (optional) `text`. See details for more information.

Details

The column `cov` in `covariate_scenarios` refers to covariate variables in the dataset. The column `value` refers to covariate values of importance. Typically these will be quantiles of continuous variables and categories (for categorical covariates). The column `text` is option but is a labelling column for `cov_forest_plot()` to adjust how the covariate scenarios are printed on the axis

Value

`dplyr::tibble` with quantile information suitable for `cov_forest_plot()`.

Examples

```
## requires NONMEM to be installed
## Not run:

dcov <- input_data(m1, filter = TRUE)
dcov <- dcov[!duplicated(dcov$ID), ]

covariate_scenarios <- dplyr::bind_rows(
  dplyr::tibble(cov = "HEALTHGP", value = c(0, 1)),
  dplyr::tibble(cov = "HEPATIC", value = unique(dcov$HEPATIC[dcov$HEPATIC > -99])),
  dplyr::tibble(cov = "BWTIMP", value = c(50, 80, 120)),
  dplyr::tibble(cov = "ECOG", value = c(0, 1, 2, 3)),
  dplyr::tibble(cov = "BEGFRIMP", value = quantile(dcov$BEGFR[dcov$BEGFR > -99])),
  dplyr::tibble(cov = "RACE", value = c(1, 2), text = c("white", "black")),
  dplyr::tibble(cov = "PPI", value = c(0, 1)),
  dplyr::tibble(cov = "H2RA", value = c(0, 1))
)

dplot <- cov_forest_data(m1, covariate_scenarios = covariate_scenarios)
cov_forest_plot(dplot)

## End(Not run)
```

`cov_forest_plot` *Plot covariate forest plots*

Description

[Stable]

Uses `ggplot2` to take outputs from `cov_forest_data()` and display a forest plot.

Usage

```
cov_forest_plot(d)
```

Arguments

`d`	A `data.frame` from `cov_forest_data()`.

Value

A `ggplot2` forest plot.

See Also

`cov_forest_data()`

Examples

```
## requires NONMEM to be installed
## Not run:

dcov <- input_data(m1, filter = TRUE)
dcov <- dcov[!duplicated(dcov$ID), ]
covariate_scenarios <- bind_rows(
  tibble(cov = "HEALTHGP", value = c(0, 1)),
  tibble(cov = "HEPATIC", value = unique(dcov$HEPATIC[dcov$HEPATIC > -99])),
  tibble(cov = "BWTIMP", value = c(50, 80, 120)),
  tibble(cov = "ECOG", value = c(0, 1, 2, 3)),
  tibble(cov = "BEGFRIMP", value = quantile(dcov$BEGFR[dcov$BEGFR > -99])),
  tibble(cov = "RACE", value = c(1, 2), text = c("white", "black")),
  tibble(cov = "PPI", value = c(0, 1)),
  tibble(cov = "H2RA", value = c(0, 1))
)

dplot <- cov_forest_data(m1, covariate_scenarios = covariate_scenarios)
cov_forest_plot(dplot)

## End(Not run)
```

`ctl_contents` *Get/set control file contents*

Description

This function is an alias for `based_on()`.

Usage

```
ctl_contents(...)
```

Arguments

`...`	Arguments to be passed to `based_on()`.

Value

An nm object with modified `ctl_contents` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 %>% ctl_contents()
```

`ctl_path` *Get and set path to NONMEM control file*

Description

[Stable]

Similar to `ctl_name()` & `run_in()`, this allows you to retrieve and specify the relative path to the control file that will be written by the `run_nm()`.

Usage

```
ctl_path(m, text)
```

Arguments

`m`	An nm object.
`text`	Optional character. Name of path to control file (see details). Typically, this file does not yet normally exist, but will house the code code for this run.

Details

Note that `text` can contain an `"{run_id}"` string. E.g. `"Models/run{run_id}.mod"` will use the name `"Models/runm1.mod"` if `run_id(m1)` is `"m1"`.

Value

`character` with path to NONMEM control file to be copied immediately prior to running (with `run_nm()`).

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))


ctl_name(m1)
ctl_path(m1)

m1 <- m1 %>% ctl_path("Models/nm_{run_id}.ctl")
ctl_path(m1)
```

`data_path` *Get/set path to dataset*

Description

[Stable]

Mainly used to associate a dataset with an nm object. Requires ctl_contents to already be specified.

Usage

```
data_path(m, text)
```

Arguments

`m`	An nm object.
`text`	Optional character. Path to input dataset.

Value

if text is not specified, will return the `data_path` name otherwise will return an nm object with modified `data_path` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

data_path(m1) ## display data name
```

`decision` *Make decision point*

Description

[Experimental]

Formalise process of decision making. Creates a decision point in the workflow where subsequent parts of your workflow depend on this decision, e.g. if you compare a 1 compartment and 2 compartment and decide based on the OFV and goodness of fit plots that the 1 compartment model is better and subsequent steps will build off of this, it is worth putting a decision point in your code so that if you are to rerun the workflow with a new/updated dataset, the decision can be revisited prior to moving onto the parts of the workflow that depend on the 1 compartment decision. The function requests inputs (`values` and `files`) that you base a decision on and stop for users to remake decision if inputs change.

Usage

```
decision(
  auto = logical(),
  inputs = c(),
  file_inputs = c(),
  outcome = character(),
  force = FALSE
)
```

Arguments

`auto`	Optional logical. logical statement for automatic decisions.
`inputs`	Optional non file names upon which decision depends.
`file_inputs`	Optional file names upon which decision depends.

`outcome`	Character. Description of the decision outcome.
`force`	Logical (default = `FALSE`). Force a stop in the workflow so decision has been remade.

Details

There are two ways to use `decision`:

Automatic: An `auto` decision (see examples below) works like `stopifnot()`. It requires a logical (TRUE/FALSE) condition. Doing this this way ensures that creates fewer points in your workflow where at the cost of removing. If updating a workflow (e.g. with an updated dataset), so long as the TRUE/FALSE is TRUE, the workflow will proceed uninterrupted. If the condition flips to FALSE the workflow will stop as it will be assumed that subsequent steps will no longer be valid.

Manual: Manual: Requires specification of either `input` or `file_inputs` (or both) AND `outcome`. Inputs represent information you have considered in your decision and `outcome` is a text description of the resulting decision. The assumption made is that if inputs have not changed since the last decision was made.

Value

No return value, called for side effects.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m2 <- m1 %>% child("m2")
m2WT <- m1 %>% child("m2WT")

## Not run:

if(interactive()){

## a decision based on summary statistics
decision(
  inputs = summary_wide(c(m1, m2, m2WT)),
  outcome = "m1 is better"
) # next line must be end of chunk

## a decision based also on goodness of fit plots
decision(
  inputs = summary_wide(c(m1, m2, m2WT)),
  file_inputs = c(
    "Results/basic_gof.m1.nb.html",
    "Results/basic_gof.m2.nb.html"
  ),
  outcome = "m1 is better"
) # next line must be end of chunk

## a decision based on an automatic TRUE/FALSE criteria
## here we're ensuring m1 has the lowest AIC
decision(AIC(m1) == min(AIC(m1, m2, m3)))

}

## End(Not run)
```

`delete_dollar` *Delete a NONMEM subroutine from control file contents*

Description

[Stable]

Usage

```
delete_dollar(m, dollar)
```

Arguments

`m`	An nm object.
`dollar`	Character. Name of subroutine.

Value

An nm object with modified `ctl_contents` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 %>% dollar("TABLE")
m1 <- m1 %>% delete_dollar("TABLE")
m1 %>% dollar("TABLE")  ## missing
```

`dollar` *Get/set existing subroutine*

Description

[Stable]

The fast way to see the contents of a particular subroutine directly in the R console. It can also be used to set the contents of a NONMEM subroutine in place of manual edits

Usage

```
dollar(m, dollar, ..., add_dollar_text = TRUE)
```

Arguments

`m`	An nm object.
`dollar`	Character. Name of NONMEM subroutine to target.
`...`	Additional arguments to be passed to `text()`. If specified these will set the contents of the subroutine. See examples below.
`add_dollar_text`	Logical (default = `TRUE`). Should the $XXXX string be added to text.

Value

If `dollar` is specified returns the relevant subroutine of the control file as a character. Otherwise returns an nm object with modified `ctl_contents` field.

See Also

`insert_dollar()`, `delete_dollar()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 %>% dollar("PK") ## displays existing $PK

m1 %>% dollar("THETA")

c(m1, m1) %>% dollar("THETA") # display $THETAs for multiple NONMEM runs
```

`dollar_subroutine` *Get/set $SUBROUTINE values in control file*

Description

[Stable]

These are mostly back end functions used by `subroutine()` and will make simple ADVAN/TRANS/TOL adjustments to the NONMEM control file. No other file changes outside $SUBROUTINE will be made which makes `advan` and `trans` less useful than the higher level `subroutine()` function.

Usage

```
advan(m, text)

trans(m, text)

tol(m, text)
```

Arguments

`m`	An nm object.
`text`	Optional number/character number to set to.

Value

If `text` is specified returns an nm object with modified `ctl_contents` field. Otherwise returns the value of the advan, trans, or tol.

See Also

`subroutine()`

`exclude_rows` *Exclude rows of NONMEM dataset*

Description

[Stable]

A mechanism for excluding outliers during data cleaning. Create exploratory plots, identify rows of the dataset to be considered outliers for exclusion, and then feed that filtered dataset into this function to exclude them from the dataset. Requires a corresponding `IGNORE` statement - see argument descriptions for more details.

Usage

```
exclude_rows(d, dexcl, exclude_col = "EXCL")
```

Arguments

`d`	A `data.frame` for containing the full NONMEM dataset. Should contain a column for identifying excluded rows named with the `exclude_col` argument.
`dexcl`	A smaller `data.frame` consisting of rows to be ignored. Need not contain all columns of `d` but each column should be present in `d`.
`exclude_col`	Character (default = `"EXCL"`). Name of a binary exclude column in `d`. This should be accompanied with a `IGNORE=(EXCL.GT.0)` statement in $DATA.

Value

A modified version of `d` with `exclude_col` set to `1` for rows coinciding with `dexcl`.

See Also

`read_derived_data()`, `write_derived_data()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

d <- input_data(m1)
d$EXCL <- 0  ## start with no rows excluded

## use with dplyr
dexcl <- d %>%
  dplyr::filter(ID == 6, TIME > 3) %>%
  dplyr::select(ID, TIME, DV, EXCL)
dexcl ## view rows to be excluded
d <- d %>% exclude_rows(dexcl)

d %>% dplyr::filter(ID %in% 6)
```

`fill_input` *Fill $INPUT*

Description

[Stable]

Uses dataset to automatically fill $INPUT in control file.

Usage

```
fill_input(m, ...)
```

Arguments

`m`	An nm object.
`...`	Either `keep`, `drop`, or `rename` arguments. See examples.

Details

If a new dataset with different columns is assigned to an `nm` object, `$INPUT` will not be correct and so it may necessary to apply `fill_input()` again.

See examples for how to use `drop` and `rename` arguments to control how `$INPUT` is written.

Value

An nm object with modified `ctl_contents` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 %>% dollar("INPUT") ## shows placeholder for column names

m1 <- m1 %>% fill_input()
m1 %>% dollar("INPUT") ## view $INPUT

## following will will drop the "RATE" column
m1 <- m1 %>% fill_input(drop = "RATE")
## no RATE column so will not drop anything
m1 %>% dollar("INPUT")

## following will rename "DATE" to be "DAT0"
m1 <- m1 %>% fill_input(rename = c("DAT0" = "DATE"))
## no DATE column so will not rename anything
m1 %>% dollar("INPUT") ## view $INPUT
```

`find_nonmem` *Find location of NONMEM installation*

Description

[Experimental]

Attempts to find location of NONMEM installation directory used by PsN. Can be useful for finding the location of parafiles etc.

Usage

```
find_nm_install_path(name = "default")

find_nm_tran_path(name = "default", warn = TRUE)
```

Arguments

`name`	Character name of NONMEM installation (according PsN).
`warn`	Logical (default = `TRUE`) to warn if fails to find NMTRAN.exe.

Details

The function `find_nm_install_path()` will attempt to use a locally available PsN installation to get this information. If the PsN installation is on a remote server, this function will not work (it will return a `NULL`)

The function `find_nm_tran_path()` will attempt to use a locally available PsN installation to get this information. If the PsN installation is on a remote server, this function will not work (it will return a `NULL`). This is normally used to set `nm_tran_command()`. If this function cannot find installation, you will need to set `nm_tran_command()`, manually.

Value

If functions cannot find installation they will return `NULL` without errors or warning, otherwise they will return the located paths.

See Also

`nm_tran_command()`

gsub_ctl *Pattern replacement for control file contents*

Description

[Stable]

A wrapper around gsub so that control files may be modified using gsub syntax. Can be useful for simple find replace operations in a control stream. Ensure you use the "view diff" app afterwards to make sure the find replace proceeded as intended.

Usage

```
gsub_ctl(m, pattern, replacement, ..., dollar = NA_character_)
```

Arguments

`m`	An nm object.
`pattern`	Argument passed to `gsub()`.
`replacement`	Argument passed to `gsub()`.
`...`	Additional arguments passed to `gsub()`.
`dollar`	Character name of subroutine.

Value

An nm object with modified `ctl_contents` field.

See Also

`apply_manual_edit()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))


m1 %>% dollar("EST")

m1 <- m1 %>% gsub_ctl("ISAMPLE=300", "ISAMPLE=600")

m1 %>% dollar("EST")
```

ignore — *Get/set ignore statement from control file contents*

Description

[Stable]

Usage

```
ignore(ctl, ignore_char)
```

Arguments

`ctl`	An nm object.
`ignore_char`	Optional character. Ignore statement to set in $DATA.

Value

If `ignore_char` is specified returns an nm object with modified `ctl_contents` field. If no IGNORE present, returns `FALSE`. Otherwise returns the value of the IGNORE statement in $DATA.

See Also

`data_ignore_char()`, `data_filter_char()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv")) %>%
  fill_input()


ignore(m1) ## display ignore statement, currently none
m1 %>% dollar("DATA")

m1 <- m1 %>% ignore("ID > 10") ## changes ignore to ignore IDs > 10.

m1 %>% dollar("DATA")
```

`import` *Import staged files into project*

Description

[Stable]

This function is used by the "code libary" RStudio 'Addin' to bring external code into your project.

Usage

```
import(
  copy_table,
  overwrite = FALSE,
  silent = FALSE,
  skip = "\\.mod$",
  find_replace_dir_names = TRUE
)
```

Arguments

`copy_table`	A `data frame` or character. if `data.frame` should be output from `stage()`, if character path, result will be `stage()`d first.
`overwrite`	Logical (default = `FALSE`).
`silent`	Logical (default = `FALSE`).
`skip`	Character (default = `"\\.mod$"`). Pattern to skip. Model files will be imported directly into the project in order to avoid conflicts and will instead reside only in the staging area.
`find_replace_dir_names`	Logical (default = TRUE). Will attempt to find replace strings in scripts to reflect `nm_default_dirs()`.

Value

Invisibly returns `copy_table` argument.

See Also

`code_library()`, `stage()`

Examples

```
## requires NMproject directory structure
## Not run:

## both of these following operations are easier in the shiny code library
## RStudio 'Addin'.
```

```
ls_code_library("Models/ADVAN2.mod") %>%
  import() ## ends up in "staging/Models"

ls_code_library("Scripts/AUC.R") %>%
  import() ## ends up "scripts" directory

## End(Not run)
```

`init_theta` *Get/set initial parameters*

Description

[Stable]

These functions are useful to obtain and modify initial values of `$THETA`, `$OMEGA` and `$SIGMA`.

Usage

```
init_theta(m, replace, ...)

init_omega(m, replace, ...)

init_sigma(m, replace, ...)
```

Arguments

`m`	An nm object.
`replace`	Optional `tibble` for replacement.
`...`	Additional arguments for mutating initial estimate NONMEM subroutines. See examples.

Details

It's easiest to learn this function by view examples, the vignette and the demo `setup_nm_demo()`. It is a good idea to view the resulting `data.frame` to see the columns that are able to be manipulated.

Value

If `replace` is specified returns an nm object with modified `ctl_contents` field. Otherwise returns a `tibble` or list of `tibbles` with initial estimation information.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))


m1 <- m1 %>%
  fill_input() %>%
  init_theta(init = c(-2, 0.5, 1)) %>%
  init_sigma(init = c(0.1, 0.1)) # %>%
  # run_nm()

init_theta(m1) ## display current $THETA in tibble-form
init_omega(m1) ## display current $OMEGA in tibble-form


## here we supply a named vector in a different order
m1 <- m1 %>% init_theta(init = c(KA = -2, V = 1))
m1 %>% dollar("THETA")

## can also manipulate other aspects (like the FIX column) similarly
m1 <- m1 %>% init_theta(init = c(KA = -2, V = 1),
                        FIX = c(KA = TRUE))
m1 %>% dollar("THETA")

## perturb all parameters by ~10%
m1 <- m1 %>% init_theta(init = rnorm(length(init), mean = init, sd = 0.1))

m1 %>% dollar("THETA")
```

`input_data` *Read input dataset of an nm object*

Description

[Stable]

Uses `data_path` field of object to locate data and read in.

Usage

```
input_data(m, filter = FALSE, na = ".", ...)
```

Arguments

`m`	An nm object.
`filter`	Logical (default = `FALSE`). Applies NONMEM ignore statement to filter dataset.
`na`	Character. Passed to `utils::read.csv()`
`...`	Additional arguments passed to either `read_derived_data()` (if `write_derived_data()` was used to create derived dataset) or `utils::read.csv()`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))


d <- input_data(m1)
head(d)

## only non-ignored rows
d_nonignore <- input_data(m1, filter = TRUE)
```

`insert_dollar` *Insert a new subroutine into control file_contents*

Description

[Stable]

Mostly a back end function used by other functions.

Usage

```
insert_dollar(m, dollar, text, after_dollar)
```

Arguments

`m`	An nm object.
`dollar`	Character. Name of subroutine to insert.
`text`	Character vector. Text to fill.
`after_dollar`	Character name of preceding subroutine. The new subroutine will be inserted immediately after it.

Value

An nm object with modified `ctl_contents` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 <- m1 %>% insert_dollar("MODEL", "
$MODEL
COMP = (CENTRAL)
", after_dollar = "SUB")

m1 %>% dollar("MODEL")
```

---

is_finished *Tests if job is finished*

Description

[Stable]

Usage

```
is_finished(r, initial_timeout = NA)
```

Arguments

r	An nm object.
initial_timeout	Deprecated. See wait_finish().

Value

A logical vector with TRUE or FALSE values.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

is_finished(m1) # FALSE
```

is_nm *Test if object is an nm coercible object*

Description

[Stable]

Mostly internal functions to test object types.

Usage

```
is_nm_list(x)

is_nm_generic(x)
```

Arguments

`x`	Object.

Value

A logical vector with `TRUE` or `FALSE` values.

is_nmproject_dir *Is the directory an NMproject directory*

Description

[Stable]

Find out whether current (or specified) directory is an NMproject directory or not.

Usage

```
is_nmproject_dir(path = getwd())
```

Arguments

`path`	Optional path to test if it's an NMproject or not.

Value

Logical `TRUE` or `FALSE`

is_successful *Test if NONMEM ran without errors*

Description

[Experimental]

Usage

```
is_successful(r)
```

Arguments

r	An nm object.

Value

`TRUE` if run was successful, `FALSE` otherwise.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

is_successful(m1) ## FALSE
```

job_stats *Get job stats for a completed NONMEM run*

Description

[Experimental]

Gets attributes of the run like run time, queue time.

Usage

```
job_stats(m)
```

Arguments

m	An nm object.

Value

A wide format `tibble` with information about the job execution times.

Examples

```
## Below code requires NONMEM to be installed
## Not run:

#' # create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

dc <- tibble(cores = c(1, 3, 10, 30)) %>%
  mutate(m = m1 %>%
    child(run_id = cores) %>%
    run_in("Models/m1_coretest") %>%
    cmd("execute {ctl_name} -parafile={parafile} -dir={run_dir} -nodes={cores}") %>%
    parafile("/opt/NONMEM/nm75/run/mpilinux8.pnm") %>%
    cores(cores))

dc$m %>% cmd()

dc$m %>%
  run_nm() %>%
  wait_finish()

## extract job statistics and plot cores vs Rtime or Ttime
## to get plots of run time and total time vs number of CPUs

dc$m %>%
  job_stats() %>%
  ggplot(aes(x = cores, y = Rtime)) +
  theme_bw() +
  geom_point()

## End(Not run)
```

`job_time_spacing` *Setup default job_time_spacing option*

Description

[Stable]

This allows organisations/individuals with their own job time spacing, used by `run_nm()` when `threads` > 1.

Usage

```
job_time_spacing(seconds)
```

Arguments

`seconds`	Optional numeric value. Values correspond to what will be set.

Value

if `seconds` is missing, will return value of `getOption("job_time_spacing")` otherwise will set option `job_time_spacing`.

Examples

```
job_time_spacing()
job_time_spacing(1)
```

`ls_code_library` *List files in code library*

Description

[Stable]

A low level function to interact with the code library. It is easier in most cases to use the shiny "code library" RStudio 'Addin'.

Usage

```
ls_code_library(pattern = ".")
```

Arguments

`pattern`	Optional character. Filter the code library use regex.

Value

Character vector of matched file paths.

See Also

`code_library()`, `stage()`, `import()`

Examples

```
ls_code_library("Models/ADVAN2.mod")

## requires NMproject directory structure to operate in
## Not run:
ls_code_library("Models/ADVAN2.mod") %>%
  stage()

## End(Not run)
```

`ls_scripts` *List scripts*

Description

[Stable]

Usage

```
ls_scripts(folder = ".", extn = "r|R|Rmd|rmd", recursive = TRUE)
```

Arguments

`folder`	String describing folder to search recursively in.
`extn`	Character (can be regex) giving extension to limit search to.
`recursive`	Logical (default = `TRUE`). Should directories be searched recursively.

Value

Character vector of matched file paths.

Examples

```
## find all scripts with the string "AUC("
ls_scripts("~/path/to/analysis/Scripts") %>% search_raw("AUC\\(")
```

make_boot_datasets *Prepare a bootstrap tibble*

Description

[Stable]

Creates bootstrap datasets and returns corresponding `nm` objects. Requires the necessary `rsample` splitting objects to be present. See examples.

Usage

```
make_boot_datasets(
  m,
  samples = 10,
  data_folder = file.path(nm_dir("derived_data"), "bootstrap_datasets"),
  overwrite = FALSE,
  id_var = "ID",
  ...
)
```

Arguments

`m`	An nm object.
`samples`	Number of samples.
`data_folder`	Folder (relative path) to store datasets.
`overwrite`	Logical (default = `FALSE`). Overwrites previous files.
`id_var`	Character (default = `"ID"`). Name of ID column in dataset.
`...`	Arguments passed to `fill_input()`.

Value

A `tibble` with `samples` rows and an nm object object column `m` for execution of the bootstrap.

Examples

```
## The following only works inside an NMproject directory structure and
## and requires NONMEM installed
## Not run:

# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))
```

```
d <- input_data(m1)

## in your dataset production script
d <- d %>%
  mutate(
    WT_C = cut(WT, breaks = 2, labels = FALSE),
    STRATA = paste(SEX, WT_C, sep = "_")
  )

d_id <- d %>% distinct(ID, STRATA)

set.seed(123)

## create large set of resamples (to enable simulation to grow
## without ruining seed)
bootsplits <- rsample::bootstraps(d_id, 100, strata = "STRATA")

dir.create("DerivedData", showWarnings = FALSE)
bootsplits %>% saveRDS("DerivedData/bootsplit_data.csv.RData")

## In a model development script, the following, performs a
## 100 sample bootstrap of model m1

m1_boot <- m1 %>% make_boot_datasets(samples = 100, overwrite = TRUE)

m1_boot$m %>% run_nm()

## the following bootstrap template will wait for results to complete
m1_boot$m %>% nm_list_render("Scripts/basic_boot.Rmd")

## End(Not run)
```

make_OCC_every_dose *Make an OCC column for NONMEM IOV use*

Description

[Experimental]

Creates and OCC column that increments in accordance to specified condition. To be used in a `dplyr::mutate()` statement `dplyr::group_by()`'d by "ID".

Usage

```
make_OCC_every_dose(d, dose_trigger, new_OCC_trigger)
```

Arguments

d	A data.frame. NONMEM ready input dataset.

`dose_trigger`	Logical expression for defining a dosing row.
`new_OCC_trigger`	Logical expression for defining when OCC should increment.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

d <- input_data(m1)

## OCC increments on every dosing interval with more than 4 samples
d %>% make_OCC_every_dose(!is.na(AMT), any(!is.na(DV)))
```

make_xv_datasets *Write (bootstrap) cross validation datasets*

Description

[Experimental]

Similar to `make_boot_datasets()`, but sets up "out of bag" datasets for model evaluation.

Usage

```
make_xv_datasets(
  dboot,
  data_folder = file.path(nm_dir("derived_data"), "bootstrap_datasets"),
  overwrite = FALSE,
  id_var = "ID"
)
```

Arguments

`dboot`	Output from `make_boot_datasets()`.
`data_folder`	Folder to store datasets.
`overwrite`	Logical. Overwrite previous files or not.
`id_var`	Character (default = `"ID"`). Name of ID column.

Value

A `tibble` of nm objects similar to `make_boot_datasets()` output.

`map_nm` *A purrr-like looping function over nm objects*

Description

[Experimental]

Loop over an nm object when a vectorized operation is not available.

Usage

```
map_nm(.x, .f, ...)

map2_nm(.x, .y, .f, ...)

imap_nm(.x, .f, ...)

pmap_nm(.l, .f, ...)
```

Arguments

`.x`	An nm object.
`.f`	A function, formula, or vector (not necessarily atomic). If a **function**, it is used as is. If a **formula**, e.g. `~ .x + 2`, it is converted to a function. There are three ways to refer to the arguments: • For a single argument function, use `.` • For a two argument function, use `.x` and `.y` • For more arguments, use `..1`, `..2`, `..3` etc This syntax allows you to create very compact anonymous functions. If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. If a component is not present, the value of `.default` will be returned.
`...`	Additional arguments passed on to the mapped function.
`.y`	An optional nm object.
`.l`	A list of vectors, such as a data frame. The length of `.l` determines the number of arguments that `.f` will be called with. List names will be used if present.

Value

A modified nm object.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

## vectorized run
m1 %>% child(1:3) %>% run_id(paste0(run_id(.), "modified"))

m1 %>% child(1:3) %>%
  map_nm(~ .x %>% run_id(paste0(run_id(.), "modified")))
```

`new_nm` *Create a new (parent) nm object*

Description

[Stable]

Create a new parent `nm` object. Normally the first NONMEM object you create will be using this function. Subsequent objects created with the `child()` function will inherit the properties of the parent run.

Usage

```
new_nm(based_on, run_id = NA_character_, data_path, cmd, force = FALSE)
```

Arguments

`based_on`	Character. Relative path to an existing control file from which to base this run. NMproject will not modify or run this control file. Instead it will create a new control file specified by the `ctl_name` field (see Details below).
`run_id`	Character. Run identifier. This is used to name the run and output files such as $TABLE outputs.
`data_path`	Character. Path to dataset. If this is not specified, NMproject will try to guess based on the current $DATA components of the file specified in `based_on`. However it is recommended to specify this explicitly as a relative path.
`cmd`	Optional character. PsN command to use. If unspecified will use `getOption("nm_default_fields")` value of `cmd`. Use glue notation for inheritance. See details.
`force`	(Default = `FALSE`). Forces object creation even if `based_on` model is in the `nm_dir("models")` directory.

Details

The cmd field uses glue notation. So instead of specifying execute runm1.mod -dir=m1, it is best to specify execute {ctl_name} -dir={run_dir}. The values of ctl_name and run_dir refer to object fields and if these change value like when the child() function is used to create a separate child object, the cmd field will update automatically.

Value

An object of class nm_list. Attributes can be viewed by printing the object in the console.

Object fields

Each field has a corresponding function (documented in nm_getsetters) of the same name to access and modify it's value.

type The PsN run type. Default is execute.

run_id The run identifier. E.g. m1.

run_in The directory to copy control files and run NONMEM. Default = "Models".

executed For internal use.

ctl_contents Stores the contents of the control file to be written to disk when the run_nm() function is used on the object.

data_path Path to the NONMEM ready dataset (from base project directory).

cmd See details above.

cores Numbers of cores to use. Requires a cmd value that uses the {cores} glue field.

run_dir PsN directory to run the NONMEM run. Default is to the be the same as the run_id for simplicity.

results_dir Location to store results files from diagnostic reports executed with nm_render().

unique_id For internal use.

lst_path Normally does not require setting. Path to the expected .lst file.

See Also

nm_getsetters(), child()

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 ## display object fields
cmd(m1)
ctl_name(m1)
run_dir(m1)
```

`nmsave` *Save plots in results_dir*

Description

[Experimental]

`nmsave_plot` is a wrapper around `ggplot2::ggsave()` for nm objects, `nmsave_table` is a wrapper for saving `data.frames` to file in the form of a .csv file.

Usage

```
nmsave_plot(
  r,
  object,
  file_name,
  directory = results_dir(r),
  width = 7,
  height = 5,
  dpi = 300,
  ...
)

nmsave_table(r, object, file_name, directory = results_dir(r), ...)
```

Arguments

`r`	An nm object.
`object`	A list of plotting objects.
`file_name`	Character. Name of results file.
`directory`	Character (default = `results_dir(r)`). Where to save.
`width`	Passed to `ggplot2::ggsave()`.
`height`	Passed to `ggplot2::ggsave()`.
`dpi`	Passed to `ggplot2::ggsave()`.
`...`	Passed to `ggplot2::ggsave()`.

Value

An nm object with modified `result_files` field.

`nm_create_analysis_project`
Create analysis project

Description

[Stable]

This is the underlying function used by: `File` -> `New Project` -> `New Directory` -> `New NMproject`. It creates a new analysis working directory with a directory structure similar to an R package.

Usage

```
nm_create_analysis_project(
  path,
  dirs = nm_default_dirs(),
  style = c("analysis", "analysis-package"),
  use_renv = FALSE,
  readme_template_package = "NMproject",
  ...
)
```

Arguments

`path`	Character path (relative or absolute) to project. If just specifying a name, this will create the analysis project in the current working directory. See details for naming requirements.
`dirs`	Character list or vector. Default = `nm_default_dirs()`. Can also handle an ordered string which is supplied by the RStudio project template interface.
`style`	Character. Either `"analysis"` or `"analysis-package"` See details for `path` requirements and function behaviour.
`use_renv`	Logical (default = `FALSE`). Should `renv` be used or not in project.
`readme_template_package`	Package name from which to load the README template (default = `"NMproject"`)
`...`	Deprecated.

Details

The function works like as is inspired by `starters::create_analysis_project()`. There is no restriction on directory name. It is therefore possible to violate R package naming conventions.

When `style = "analysis"` is selected, the analysis directory will be package-like in structure, with the package name `"localanalysis"`. For `style = "analysis-package"`, `path` should contain only (ASCII) letters, numbers and dot, have at least two characters and start with a letter and not end in a dot. See Description file requirements for more information.

This is to cater to users who like underscores and aren't interested in creating a package.

Default modelling directories

Default modelling directories can be modified with `nm_default_dirs` option (see `options()` for information on how to modify this). A (partially) named list of directories to be used by `nm_create_analysis_project` Required names are `"models"`, `"scripts"` and `"results"`. By default these are set to `"Models"`, `"Scripts"` and `"Results"`, respectively. Additional nameless characters (e.g. `"SourceData"`) correspond to additional modelling directories.

"SourceData": intended for unmodified source datasets entering the analysis project.

"DerivedData": intended for cleaned and processed NONMEM ready datasets

"Scripts": intended for all R scripts

"Models": intended for all NONMEM modelling

"Results": intended as default location for run diagnostics, plots and tables

See Also

`nm_default_dirs()` for modifying default directory structure.

`nm_default_dirs` *Setup analysis subdirectories*

Description

[Stable]

This allows organisations/individuals with their own directory to customize their directory structure

Usage

```
nm_default_dirs(dir_list)
```

Arguments

`dir_list`	Optional named list or vector. Names `"scripts"` and `"models"` must be present. The rest can be unnamed.

Value

if `dir_list` is missing, will return value of `getOption("nm_default_dirs")` otherwise will set option `nm_default_dirs`.

Examples

```
orig_list <- nm_default_dirs()
orig_list

nm_default_dirs(list(
  models = "Models",
  scripts = "Scripts",
  results = "Results",
  source_data = "SourceData",
  derived_data = "Data"
))

nm_default_dirs()
nm_default_dirs(orig_list)
```

`nm_default_fields` *Setup default nm object fields*

Description

[Stable]

This allows organisations/individuals with their own nm object field preferences to set these.

Usage

```
nm_default_fields(field_list)
```

Arguments

`field_list`	Optional named list or vector. Names correspond to function names and object fields, values correspond to what will be set.

Value

if `field_list` is missing, will return value of `getOption("nm_default_fields")` otherwise will set option `nm_default_fields`.

Examples

```
nm_default_fields()
nm_default_fields(list(
  cmd = "execute {ctl_name} -dir={run_dir}"
))
nm_default_fields()
```

nm_diff — *Compute diff between two NONMEM runs*

Description

[Stable]

The easiest way to use this function is via the "view diff" RStudio 'Addin'.

NMproject's control file manipulation functions (e.g. `subroutine()`) may not work for all control files. It is the responsibility of the user to check automatic manipulations are done properly. Displaying diffs provides a means of manually checking what was done.

Usage

```
nm_diff(m, ref_m, format = "raw")
```

Arguments

`m`	An nm object.
`ref_m`	An optional nm object (base/reference object). If not specified, it will compute the diff the initial control file contents associated with the object at the time of object create. This information is stored in the `ctl_orig` field.
`format`	Character (default = `"raw"`) argument passed to `diffobj::diffChr()`

Value

Invisibly returns a `character` vector of the diff.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m2 <- m1 %>% child(run_id = "m2") %>%
  subroutine(advan = 2, trans = 2)

nm_diff(m2, m1)
```

nm_dir *Get a directory name*

Description

[Stable]

Get subdirectory (relative) paths in a configuration independent way. The configuration can be modified with nm_default_dirs(). Can be useful in scripts, where you need to refer to locations of model files or output files.

Usage

```
nm_dir(name, ...)
```

Arguments

name	Character. Directory type. Should be either "scripts", "models" or "results".
...	Deprecated.

See Also

nm_default_dirs()

Examples

```
nm_dir("scripts") ## will return the path to the "scripts" directory
nm_dir("models")
nm_dir("results")
```

nm_getsetters *Functions to access and modify fields of nm objects*

Description

[Stable]

The fields of an object can be viewed by printing the object. Each field has a corresponding function of the same name to access and modify it's value.

Usage

```
run_dir(m, text)

cmd(m, text)

type(m, text)

parent_run_id(m, text)

parent_run_in(m, text)

parent_ctl_name(m, text)

parent_results_dir(m, text)

unique_id(m, text)

ctl_name(m, text)

results_dir(m, text)

run_in(m, text)

run_id(m, text, update_ctl = FALSE)

result_files(m, text)

lst_path(m, text)
```

Arguments

`m`	An nm object.
`text`	Optional character for replacing field. If present function will modify field (of same name as function) otherwise will return value of field (of same name as function).
`update_ctl`	Should NONMEM code be updated to reflect the new run_id. Assumes `text` is not missing. Default = FALSE.

Details

Easiest way to see all fields of an object is to type the object into the console and hit enter. This will display the value of each field.

The fundamental structure of all these functions is the same:

To access the value of a field: `m %>% fieldname()` or equivalently `fieldname(m)`.

To modify the value of a field: `m <- m %>% fieldname("newvalue")`

Some fields like `cmd` are glue fields. In these cases inserting expressions inside braces in `text` will evaluate the expression (see examples).

The function `lst_path()` sets the expected location of the lst file and all output files relative to the `run_in` location.

Value

The value of the specified field of `m` if `text` is missing. Otherwise an nm object with modified field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))


run_dir(m1)

m1 <- m1 %>% run_dir("{run_id}_dir")
run_dir(m1)


## set cmd field of m1
m1 <- m1 %>% cmd("execute {ctl_name} -dir={run_dir}")

m1 %>% cmd()
## displays "execute runm1.mod -dir=m1"

## can also view field when viewing object
m1
```

`nm_getsetters_execution`
Execution related functions to access and modify fields of nm objects

---

Description

[Stable]

The fields of an object can be viewed by printing the object. Each field has a corresponding function of the same name to access and modify it's value.

Usage

```
cores(m, text)

parafile(m, text)

walltime(m, text)
```

```
executed(m, text)
```

Arguments

`m`	An nm object.
`text`	Optional character for replacing field. If present function will modify field (of same name as function) otherwise will return value of field (of same name as function).

Details

Easiest way to see all fields of an object is to type the object into the console and hit enter. This will display the value of each field. some fields like `cmd` are glue fields. In these cases inserting expressions inside braces in `text` will evaluate the expression

The fundamental structure of all these functions is the same:

To access the value of a field: `m %>% fieldname()` or equivalently `fieldname(m)`

To modify the value of a field: `m <- m %>% fieldname("newvalue")`

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 <- m1 %>% cmd("execute -parafile={parafile} {ctl_name} -dir={run_dir} -nodes={cores}")

m1 <- m1 %>% cores(8) %>% parafile("mpilinux8.pnm")

cmd(m1)
cores(m1)
```

`nm_list_gather` *Get all nm_list objects*

Description

[Stable]

Get all nm objects in an environment. By default this is the global workspace.

Usage

```
nm_list_gather(x = .GlobalEnv, max_length = 1)
```

Arguments

x	An environment (default = `.GlobalEnv`) to search or `data.frame` with (`nm_list` column) or `nm_list`.
max_length	Exclude objects with length above this.

Value

A single `nm_list` object with all model objects in environment `x`.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m2 <- m1 %>% child("m2")

m_all <- nm_list_gather()

identical(
  m_all %>% subset(run_id(m_all) %in% "m1"),
  m1
)
```

---

nm_output_path *Find an output file associated with a run*

Description

[Stable]

This is primarily a backend function used to identify output file paths associated with nm objects.

Usage

```
nm_output_path(m, extn, file_name)
```

Arguments

m	An nm object.
extn	Character. Name of extension.
file_name	Optional character. Name of file name.

Value

The path to the relevant output file of `m`.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 %>% nm_output_path("ext") ## path to ext file
```

`nm_read_table` *Fast read of NONMEM output table*

Description

[Stable]

Reads in $TABLE outputs rapidly. `output_table()` is a higher level function for reading output files and combining with input datasets.

Usage

```
nm_read_table(file, ...)
```

Arguments

`file`	File argument from `utils::read.table()`.
`...`	Other arguments to be passed to `utils::read.table()`.

Value

A `data.frame` from a relevant $TABLE output file.

See Also

`output_table()`

nm_render *Create run reports*

Description

[Stable]

A wrapper around `rmarkdown::render` for nm objects. Use markdown templates to create a customised set of diagnostics to reuse on multiple models. In the demo an example is shown in `Scripts/basic_gof.Rmd`, but ideally you'll create your own customised version with everything you need to evaluate your model. To create an R markdown diagnostic template go to `FILE` -> `New File` -> `R markdown` -> `From Template` the select from one of the following:

- model diagnostic
- VPC diagnostic
- PPC diagnostic
- bootstrap results (`nm_list_render`)

These are intentionally minimal templates that can be run as notebooks or as automated diagnostics run with `nm_render`. Follow the instructions at the top of the template for more details.

Usage

```
nm_render(
  m,
  input,
  output_file = NA,
  args = list(),
  force = getOption("nm.force_render"),
  ...
)

nm_list_render(
  m,
  input,
  output_file = NA,
  args = list(),
  force = getOption("nm.force_render"),
  ...
)
```

Arguments

`m`	An nm object.
`input`	Character. Same as `rmarkdown::render()` arg.
`output_file`	Character. Same as `rmarkdown::render()` arg.

`args`	List. Same as "params" arg in `rmarkdown::render()`.
`force`	Logical (default = `getOption("nm.force_render")`). Will force execution.
`...`	Additional argument passed to `rmarkdown::render()`.

Details

`input` must refer to a properly specified Rmd document. The R markdown template "model diagnostic" in RStudio sets this up for you.

These R markdown templates are usable as R Notebooks (e.g. for code development and debugging) if the object `.m` is defined in the global work space first.

`nm_list_render()` is mostly used for bootstraps, and other routines where a parent run spawns multiple children in the form of an nm_list

Value

The same nm object, `m`, with modified `results_files` field.

Examples

```
## requires NONMEM to be installed
## Not run:
m1 %>% nm_render("Scripts/basic_gof.Rmd")

## to run "Scripts/basic_gof.Rmd" as an R Notebook
## first define .m

.m <- m1 ## Now you can run "Scripts/basic_gof.Rmd" as a Notebook

## End(Not run)
```

`nm_summary` *Generate a summary of NONMEM results*

Description

[Stable]

Get wide (or a long) `tibble` showing summary results.

Usage

```
summary_wide(
  ...,
  include_fields = character(),
  parameters = c("none", "new", "all"),
  m = TRUE,
  trans = TRUE
)

summary_long(..., parameters = c("none", "new", "all"))
```

Arguments

`...`	Arguments passed to `summary()`, usually a vector of nm object + options.
`include_fields`	Character vector of nm object fields to include as columns in the output. Default is empty.
`parameters`	Character. Either `"none"` (default), `"new"`, or `"all"` indicating whether parameter values should be included in the summary `tibble`. Specifying `"new"` means that only parameters that aren't in the `parent` run are included in outputs. This is useful if wanting to know the value of an added parameter but not all the parameters (e.g. in a covariate analysis).
`m`	Logical (default = `TRUE`). Should model object be included as the `m` column.
`trans`	Logical (default = `TRUE`). Should parameters be transformed in accordance with $THETA/$OMEGA/$SIGMA comments. This is only valid if `parameters` is `"new"` or `"all`.

Value

A wide format `tibble` with run results.

A long format `tibble` with run results coerced to `character` form.

Examples

```
## requires NONMEM to be installed

## Not run:

summary_wide(c(m1, m2))
summary_long(c(m1, m2))


## End(Not run)
```

`nm_tran` *Run NMTRAN step of a NONMEM job*

Description

[Stable]

This is the function behind the "nm_tran" RStudio 'Addin', which is the recommended way to use this functionality. Highlight your code (e.g see examples below for a code segment), and then open the "nm_tran" RStudio 'Addin'.

Useful especially on grid infrastructures where it may take a while for NONMEM to start return control file and dataset errors. Runs initial NMTRAN step of NONMEM in a temporary directory where control file and dataset checks are performed. Stops before running NONMEM.

Usage

```
nm_tran(x)
```

Arguments

x	An nm object.

Value

The same `x` object is returned, called for side effects.

See Also

`run_nm()`, `nm_tran_command()` for configuration.

Examples

```
## requires NONMEM to be installed

## Not run:

## highlight the code below and use the "nm_tran" RStudio 'Addin'

m1 <- new_nm(run_id = "m1",
             based_on = "staging/Models/ADVAN2.mod",
             data_path = "DerivedData/data.csv") %>%
  cmd("execute {ctl_name} -dir={run_dir}") %>%
  fill_input() %>%
  init_theta(init = c(-2, 0.5, 1)) %>%
  init_sigma(init = c(0.1, 0.1)) %>%
  run_nm()

## End(Not run)
```

`nm_tran_command` *Get/set nm_tran_command*

Description

[Stable]

The function `nm_tran()` needs the location of `NMTRAN.exe` to function. This is guessed at package load, assuming PsN is on the $PATH environmental variable. If this is not the case, then you can manually set the path and command used.

Usage

```
nm_tran_command(text)
```

Arguments

`text`	Optional character. If specified will set `nm_tran_command` otherwise it will display the current option value.

Details

`text` can just be the path to `NMTRAN.exe` in which case `nm_tran_command` will use the format `/path/to/NMTRAN.exe < {ctl_name}` to launch `NMTRAN.exe` where `{ctl_name}` is the name of the control file. Specifying

`nm_tran_command("/path/to/NMTRAN.exe < {ctl_name}")` is equivalent to: `nm_tran_command("/path/to/NMTRAN.exe`

More complicated formats are possible with different installations which can be seen examples.

As with all NMproject configuration options set this up either at the beginning of your script, in your `.Rprofile` or for all users in `Rprofile.site`. See FAQ for setting up configuration options permanently.

Value

If `text` is missing will get and return the current NMTRAN command.

See Also

`find_nm_tran_path()`, `nm_tran()`

Examples

```
orig_cmd <- nm_tran_command()
orig_cmd

# the following two are equivalent
nm_tran_command("/opt/NONMEM/nm75/tr/NMTRAN.exe")
nm_tran_command()
nm_tran_command("/opt/NONMEM/nm75/tr/NMTRAN.exe < {ctl_name}")
nm_tran_command()

nm_tran_command(orig_cmd)
```

`nm_tree` *Make data.tree object*

Description

[Experimental]

Draw a tree diagram showing model development path.

Usage

```
nm_tree(..., summary = FALSE)
```

Arguments

`...`	Arguments passed to `nm_list_gather()`.
`summary`	Logical (default = `FALSE`). Should `summary_wide()` variables be appended.

Value

A `data.tree` object.

`NONMEM_version` *NONMEM version info*

Description

[Stable]

Gets version information about the NONMEM installation, PsN installation and compilers. Can be useful for documentation purposes.

Usage

```
NONMEM_version()
```

Value

Returns list with version info for NONMEM, PsN, perl and fortran compiler (only gfortran currently).

ofv *Get Objective Function Value (OFV)*

Description

[Stable]

Extracts OFV from .ext file.

Usage

```
ofv(r)
```

Arguments

`r`	An nm object.

Value

The `numeric` value of the OFV.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

is_finished(m1) ## FALSE until run is completed
ofv(m1) ## NA until m1 is finished
```

omega_matrix *Get OMEGA matrix from run*

Description

[Experimental]

Obtain in matrix form the OMEGA matrix. This is primarily to feed into other packages such as `mrgsolve`.

Usage

```
omega_matrix(r)
```

Arguments

`r`	An nm object.

Value

A `matrix` object.

Examples

```

## requires NONMEM to be installed
## Not run:

## matrix of initial estimates
m1 %>% omega_matrix()

## matrix of final estimates
m1 %>%
  update_parameters() %>%
  omega_matrix()

## End(Not run)
```

`output_table` *Reads all $TABLE outputs and merge with input dataset*

Description

[Stable]

Produces a single merged output dataset will all columns of $INPUT dataset. This is useful for reuse of exploratory data plots as diagnostic plots as all columns including text columns used for `ggplot` facetting will be present.

Usage

```
output_table(r, only_append = c(), ...)

output_table_first(r, ...)
```

Arguments

`r`	An object of class nm.
`only_append`	Optional character vector. If missing will append all, otherwise will append only those variables requested.
`...`	Optional additional arguments to pass on to read.csv of orig data.

Value

A list of tibbles with merged version of all output $TABLEs and the input data. Additional columns will be INNONMEM which will be TRUE for rows that were not ignored by NONMEM. For simulation control files there is also DV_OUT which will contain simulated DV values. DV will always be unmodified from the input dataset.

output_table_first will return a tibble with a single run.

See Also

nm_render(), input_data()

Examples

```
## requires NONMEM to be installed

## Not run:

## exploratory data plot
read_derived_data("DerivedData/data.csv") %>%
  ggplot(aes(x = TIME, y = DV)) +
  theme_bw() +
  geom_point() +
  geom_line(aes(group = ID)) +
  facet_wrap(~STUDYTXT)

m1 %>%
  output_table_first() %>%
  ggplot(aes(x = TIME, y = DV)) +
  theme_bw() +
  geom_point() +
  geom_line(aes(group = ID)) +
  facet_wrap(~STUDYTXT) +
  ## additional layer for overlaying IPRED curves
  geom_line(aes(y = IPRED, group = ID))

## End(Not run)
```

---

overwrite_behaviour *Overwrite behaviour of NMproject*

Description

[Stable]

This is best used via the "overwrite behaviour" RStudio 'Addin'. Sets the strategy for how to handle overwriting of previously executed control files.

Usage

```
overwrite_behaviour(txt = c("ask", "overwrite", "stop", "skip"))
```

Arguments

`txt`	Character either `"run"`, `"stop"`, or `"skip"`.

Value

if `txt` is missing returns `getOption(\"nm.overwrite_behaviour\")` otherwise returns no value and is called for side effects (setting the `nm.overwrite_behaviour` option).

`parallel_execute` *Generic execute command for parallelised runs*

Description

[Stable]

Character to be used with the `cmd()` function to launch a parallelised run.

Usage

```
parallel_execute
```

Format

An object of class `character` of length 1.

Details

Requires `cores` and `parafile` fields to be set.

Value

A `character` object.

See Also

`nm_getsetters()`.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv")) %>%
  cmd(parallel_execute) %>%
  parafile("/opt/NONMEM/nm75/run/mpilinux8.pnm") %>%
  cores(8)

cmd(m1)

m2 <- m1 %>% child("m2") ## inherits same command as above

parallel_execute ## view the character to see how psn interfaces with SGE
```

`param_cov_diag` *Plot relationship between a parameter and covariate*

Description

[Stable]

Plots posthoc parameter-covariate relationships from NONMEM run.

Usage

```
param_cov_diag(r, param, cov, ..., categorical = FALSE, plot_tv = TRUE)
```

Arguments

`r`	An nm object.
`param`	Character. Name of parameter.
`cov`	Character. Name of covariate.
`...`	Additional arguments passed to `dplyr::mutate()`.
`categorical`	Logical (default = `FALSE`).
`plot_tv`	Logical.

Details

The mutate statement is to add variables not included in original $TABLE.

Value

A `ggplot2` plot object.

`parent_run` *Get parent object of nm object*

Description

[Stable]

Will pull the parent run of an nm object from the run cache. Run needs to have been executed for this to work.

Usage

```
parent_run(m, n = 1L)
```

Arguments

`m`	An nm object.
`n`	Numeric. Generation of parent (default = 1).

Value

An nm object. Will not return parent object, if the parent object has not been run.

`plot_iter` *Plot iterations vs parameters/OBJ*

Description

[Stable]

Non interactive `ggplot2` based version of the OFV/parameter vs iteration plot in shiny run monitor `shiny_nm()`. Used mainly for inclusion in diagnostic reports.

Usage

```
plot_iter(r, trans = TRUE, skip = 0, yvar = "OBJ")
```

Arguments

`r`	An nm object.
`trans`	Logical (default = `TRUE`). Should parameter transformations be performed in accordance with $THETA/$OMEGA/$SIGMA comments
`skip`	Numeric (default = 0). The number of iterations to skip when plotting. For observing stationarity it is often useful to remove the beginning iterations where the OFV and parameters may move a lot.
`yvar`	Character (default = `"OBJ"`). Name of variable/parameter to display.

Value

A `ggplot2` object.

See Also

`shiny_nm()`, `nm_render()`

`ppc_data` *PPC functions: process data from simulation and plot*

Description

[Stable]

Usage

```
ppc_data(
  r,
  FUN,
  ...,
  pre_proc = identity,
  max_mod_no = NA,
  DV = "DV",
  statistic = "statistic"
)

ppc_whisker_plot(d, group, var1, var2, statistic = "statistic")

ppc_histogram_plot(d, var1, var2, statistic = "statistic")
```

Arguments

`r`	An nm object (a simulation run).
`FUN`	Statistic function accepting a NONMEM dataset `data.frame` as an argument and returns `data.frame` with a column `"statistic"`.
`...`	Additional arguments for `FUN`.
`pre_proc`	Function to apply to dataset prior to compute statistics.
`max_mod_no`	Integer. Maximum model number to read (set low for debugging).
`DV`	Character (default = `"DV"`).
`statistic`	Character (default = `"statistic"`). Name of statistic column returned by FUN.
`d`	Output from `ppc_data()`.
`group, var1, var2`	Grouping variables for plotting.

Value

The function `ppc_data()` return a `data.frame` with observed and predicted statistics. The `ppc_*_plot()` plotting functions return `ggplot` objects.

See Also

`nm_render()`

Examples

```
## requires NONMEM to be installed
## Not run:

idEXPstat <- function(d, ...) { ## example individual statistic function
  ## arg = nonmem dataset data.frame
  ## return data.frame with statistic column
  d %>%
    group_by(ID, ...) %>%
    filter(is.na(AMT)) %>%
    summarise(
      AUC = AUC(time = TIME, conc = DV),
      CMAX = max(DV, na.rm = TRUE),
      TMAX = TIME[which.max(DV)]
    ) %>%
    tidyr::gather(key = "exposure", value = "statistic", AUC:TMAX) %>%
    ungroup()
}

EXPstat <- function(d, ...) { ## example summary statistic function
  ## arg = nonmem dataset data.frame
  ## return data.frame with statistic column
  d %>%
    idEXPstat(...) %>% ## reuse idEXPstat for individual stats
    ## summarise over study and any other variables (...)
    group_by(exposure, ...) %>%
    summarise(
      median = median(statistic, na.rm = TRUE),
      cv = 100 * sd(statistic, na.rm = TRUE) / mean(statistic, na.rm = TRUE)
    ) %>%
    tidyr::gather(key = "type", value = "statistic", median:cv)
}

dppc <- m1s %>% ppc_data(EXPstat)

dppc %>% ppc_whisker_plot()
dppc %>% ppc_forest_plot()

## End(Not run)
```

psn_style_scm *PsN style stepwise covariate method*

Description

[Experimental]

Intent is not to replicate PsN SCM. This is mainly here for illustrative and comparison purposes. Should replicate the model selection in PsN's SCM functionality with greedy setting.

Usage

```
psn_style_scm(base, run_in, dtest, alpha_forward = 0.05, alpha_backward = 0.01)
```

Arguments

`base`	An nm object (base model).
`run_in`	A directory to run in.
`dtest`	Output of `test_relations()`.
`alpha_forward`	Numeric (default = `0.05`). Alpha level for forward inclusion.
`alpha_backward`	Numeric (default = `0.01`). Alpha level for backward deletion.

Value

The nm object of the selected model.

See Also

`test_relations()`, `covariate_step_tibble()`, `bind_covariate_results()`.

read_derived_data *Read derived data*

Description

[Stable]

Read the derived data directly instead of via the nm object which is what `input_data()` does.

Usage

```
read_derived_data(name, na = ".", silent = FALSE, ...)
```

Arguments

`name`	Name or path of file (with or without extension).
`na`	Character to be passed to `utils::read.csv()`.
`silent`	Logical (default = `TRUE`). Should messages be suppressed.
`...`	Additional arguments to be passed to `utils::read.csv()`.

Value

A `data.frame` object of the NONMEM dataset.

See Also

`write_derived_data()`, `input_data()`, `exclude_rows()`.

Examples

```
## requires NMproject directory structure to operate in
## Not run:

## read a dataset that's been copie into SourceData
d <- read.csv("SourceData/orig_data.csv")

## modify it
d <- d[d$ID < 10, ]

d %>% write_derived_data("DerivedData/data.csv")

## load it again either with
d <- read_derived_data("data")

## or more commonly if it is associated with run (e.g. m1),
## you can use input_data() to load it via the nm object

d <- input_data(m1)

## End(Not run)
```

`remove_parameter` *Remove parameter from NONMEM control file*

Description

[Stable]

Attempts to remove a parameter from the NONMEM control assuming it has been written according to NMproject conventions (i.e. TVPARAM notation and TVPARAM + IIV_PARAM comments in $THETA/$OMEGA). The presence of any code that depends on the removed parameter will cause the control file to break.

Usage

```
remove_parameter(m, name)
```

Arguments

`m`	An nm object.
`name`	Character. Parameter name to remove.

Value

An nm object with modified `ctl_contents` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 <- m1 %>% remove_parameter("KA")

nm_diff(m1)
```

`rename_parameter` *Rename a parameter in NONMEM control stream*

Description

[Stable]

Usage

```
rename_parameter(m, ...)
```

Arguments

`m`	An nm object.
`...`	Named arguments with character values indicated old names.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 <- m1 %>% rename_parameter(V2 = "V")

m1 %>% dollar("PK")
m1 %>% dollar("THETA")
```

`rmd_to_vignettes` *Convert R markdown scripts to vignettes*

Description

[Experimental]

Copies (by default) all scripts `s01_XXX.Rmd`, `s02_XXX.Rmd` into the "vignettes" and reformats so they meet vignette standards. Use of `devtools::build_vignettes()` can then be used to build vignettes.

Usage

```
rmd_to_vignettes(script_files, overwrite = FALSE)
```

Arguments

`script_files`	Optional character vector of scripts. If empty will find scripting making the `s##_XXX.Rmd` convention. Must be .Rmd files
`overwrite`	Logical (default = `FALSE`). Overwrites existing vignettes of the same name.

Details

Uses of `decision()` must pass without stopping so these must have been run interactively prior to use of `devtools::build_vignettes()`.

Value

No return value, called for side effects.

rr — *Run record*

Description

[Stable]

Displays the transformed parameters of a completed or running model. Normally used inside of a diagnostic template, but can be useful for quickly seeing parameter estimates of several models.

Usage

```
rr(m, trans = TRUE)
```

Arguments

`m`	An nm object.
`trans`	Logical. If `TRUE` (default) will transform using control file $THETA/OMEGA conventions.

Value

A `tibble` with NONMEM run results.

NONMEM coding conventions used by NMproject

The convention for $THETA comments used by NMproject is `value ; name ; unit ; transformation`

e.g. `$THETA 0.1    ; KA ; h-1 ; LOG`

The options for THETA transformations are: `LOG`, `LOGIT`, `RATIO` and missing. `LOG` and `LOGIT` refer to log and logit transformed THETAs, respectively where the parameters should be back-transformed for reporting. `RATIO` refers to ratio data types, i.e. parameters that are positive and have a meaningful zero. Most parameters like KA, CL, EMAX fall into this category, but covariates effects which can go negative do not. RSEs are calculated for ratio data. Missing transformations are suitable for all other parameters, here no RSEs will be calculated, only raw SE values will be reported.

The convention for $OMEGA is similar but without a unit item: `value ; name ; transformation`

e.g. `$OMEGA 0.1    ; IIV_KA ; LOG`

The options for OMEGA are either `LOG` or missing. `LOG` indicating that the individual parameter distribution is log normally distributions and should be reported as a CV% (and associated RSE%) rather than as the raw NONMEM estimate.

The convention for $OMEGA is just : `value ; name`.

THETA transformations using `trans = TRUE`

The value of FINAL and RSE% (always accompanied with a `%` symbol in outputs) in the returned `tibble` is the reported standard error (where applicable) where θ and $se(\theta)$ are the NONMEM reported values of parameters and standard errors, respectively:

`LOG` $FINAL = exp(\theta), RSE = 100\sqrt(exp(se(\theta)^2) - 1)$

`RATIO` $FINAL = \theta, RSE = 100se(\theta)/\theta$

`LOGIT` $FINAL = 100/(1 + exp(-\theta)), SE = se(\theta)$

missing $FINAL = \theta, SE = se(\theta)$

OMEGA transformations using `trans = TRUE`

The value of FINAL and RSE% (always accompanied with a `%` symbol in outputs) in the returned `tibble` is the reported standard error (where applicable) where ω^2 and $se(\omega^2)$ are the NONMEM reported values of parameters and standard errors, respectively

`LOG` $FINAL = 100\sqrt(exp(\omega^2) - 1), RSE = 100(se(\omega^2)/\omega^2)/2$

missing $FINAL = \omega^2, SE = se(\omega^2)$

SIGMA transformations using `trans = TRUE`

The value of FINAL and RSE% (always accompanied with a `%` symbol in outputs) in the returned `tibble` is the reported standard error (where applicable) where σ^2 and $se(\sigma^2)$ are the NONMEM reported values of parameters and standard errors, respectively. All sigmas are reported as standard deviations.

all sigmas $FINAL = \sqrt(\sigma^2), RSE = 100se(\sigma^2)/\sigma^2$

See Also

`nm_render()`

Examples

```
## requires NONMEM to be installed
## Not run:

rr(m1)

## compare m1 and m2

rr(c(m1, m2))

## End(Not run)
```

`run_all_scripts` *Run all project scripts sequentially*

Description

[Stable]

Runs/renders all scripts `s01_XXX`, `s02_XXX` in the designated "scripts" directory.

Usage

```
run_all_scripts(index, quiet = FALSE)
```

Arguments

`index`	Numeric index for subsetting list of scripts before running.
`quiet`	Argument passed to `rmarkdown::render()`.

Details

Works with .R and .Rmd extensions. Behaviour is to `source()` .R files and use `rmarkdown::render()` on .Rmd files

Value

Invisibly returns `TRUE` if file creation is successful.

`run_dir_path` *Get path to run_dir*

Description

[Stable]

The function `run_dir()` gives the directory name, whereas this function gets the (relative) path of `run_dir()`.

Usage

```
run_dir_path(m)
```

Arguments

`m`	An nm object.

Value

A path to the `run_dir` field of `m`.

See Also

nm_getsetters().

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

run_dir_path(m1)
```

---

run_nm *Run NONMEM jobs*

Description

[Stable]

Run nm objects. Uses `system_nm()` to submit the `cmd()` value of object.

Usage

```
run_nm(
  m,
  threads = Inf,
  ignore.stdout = FALSE,
  quiet = getOption("quiet_run"),
  intern = getOption("intern"),
  force = FALSE,
  cache_ignore_cmd = FALSE,
  cache_ignore_ctl = FALSE,
  cache_ignore_data = FALSE
)
```

Arguments

`m`	An nm object.
`threads`	Numeric. Number of jobs to run concurrently (default = `Inf`). Will block the console until all jobs are submitted.
`ignore.stdout`	Logical (default=`TRUE`). Parameter passed to `system_nm()`.
`quiet`	Logical (default=`FALSE`). Should `system_nm()` output be piped to screen?
`intern`	Logical. `intern` argument to be passed to `system_nm()`.
`force`	Logical (default = `FALSE`). Force run even results unchanged.

`cache_ignore_cmd`
: Logical (default = `FALSE`). Should check `cmd` field with cache?

`cache_ignore_ctl`
: Logical (default = `FALSE`). Should check control file contents with cache?

`cache_ignore_data`
: Logical (default = `FALSE`). Should check dataset with cache?

Details

In grid environment it is recommended to run `nm_tran()` via the RStudio 'Addin' prior to executing this code.

By default, when highlighting code and evaluating it via an RStudio app, `run_nm()` will not execute and will just return the `nm` object.

For vector `nm` objects of length more than 1, all runs will be launched at the same time with a gap of `getOption("job_time_spacing")` seconds (default = 0). This could overwhelm resources if not in a grid environment.

`run_nm` is a variant of `run_nm_single()` containing a `threads` argument that will submit `run_nm()`'s in batches and wait for them to complete. If you need all the runs to complete ensure you use a `wait_finish()` statement afterwards as R console will only be blocked for until the last batch has been submitted which will be before all runs have completed.

The `job_time_spacing` argument

Value

`m` with `job_info` fields populated.

See Also

`nm_tran()`

Examples

```
## requires NONMEM to be installed
## Not run:
m1 <- new_nm(
  run_id = "m1",
  based_on = "staging/Models/ADVAN2.mod",
  data_path = "DerivedData/data.csv"
) %>%
  cmd("execute {ctl_name} -dir={run_dir}") %>%
  fill_input() %>%
  run_nm()

## End(Not run)
```

search_raw *Search for files matching raw text search*

Description

[Stable]

Searches through the list of supplied for matching strings of text. Useful in finding files that you know contain certain text snippets.

Usage

```
search_raw(files, text, search_title = TRUE, search_contents = TRUE)
```

Arguments

`files`	Vector string of files (either names or paths).
`text`	String (can be regex) to search for.
`search_title`	Logical (default=`TRUE`). Should matching occur in title.
`search_contents`	Logical (default=`TRUE`). Should matching occur in file contents.

Value

A subset of `files` with contents matching `text`.

See Also

`ls_scripts()`, `ls_code_library()`, `stage()`

Examples

```


ls_scripts("Scripts") %>% search_raw("AUC") ## finds all scripts containing string "AUC"

## regex match find instances of AUC() function being used
ls_scripts("Scripts") %>% search_raw("AUC\\(")

## requires NMproject directory structure to operate in
## Not run:
## bring file(s) into project
ls_scripts("/path/to/other/analysis/scripts/dir") %>%
  search_raw("AUC\\(") %>%
  import()

## End(Not run)
```

`setup_code_completion` *Set up code completion for NMproject*

Description

[Experimental]

Intelligent code completion is an experimental way to type NMproject code. This function modifies/creates `r.snippets`. Needs to be run interactively. Will ask for user confirmation since snippets are an RStudio config setting

Usage

```
setup_code_completion(force = FALSE, snippet_path = find_snippet_path())
```

Arguments

`force`	Logical. The default is `FALSE` which will require user confirmation before editing `r.snippets`.
`snippet_path`	Character path to the `r.snippets` file.

Value

No return value, called for side effects.

`setup_nm_demo` *Setup demo in current directory*

Description

[Stable]

Following through the demo is the fastest way to learn the syntax of NMproject. The default demo is a Theophylline ("theopp") pharmacometric analysis. Scripts will be copied numbered `s01_XXX.Rmd`, `s02_XXX.Rmd` in the `"Scripts"` directory and a dataset into `"SourceData"`. The `"staging"` area will also be pre-filled with the code library model, `"ADVAN2.mod"`. To practice copying this yourself, see `code_library()` for how the app works.

Usage

```
setup_nm_demo(
  demo_name = "theopp",
  overwrite = FALSE,
  additional_demo_locations = NULL
)
```

Arguments

`demo_name`	Character. Name of demo. Default = "theopp". See details to find other demos
`overwrite`	Logical. Default changed to FALSE.
`additional_demo_locations`	Character vector. default = NULL. Locations for demo directories.

Details

Available `demo_name` correspond to directory locations in `system.file("extdata","examples",package = "NMproject")`

Value

Invisibly returns a `tibble` with imported file information.

See Also

`code_library()`

`sge_parallel_execute` *Generic execute command for SGE grids*

Description

[Stable]

Character to be used with the `cmd()` function to launch a parallelised job on SGE.

Usage

```
sge_parallel_execute

sge_parallel_execute2

sge_parallel_execute_batch
```

Format

An object of class `character` of length 1.

An object of class `character` of length 1.

An object of class `character` of length 1.

Details

Requires `cores` and `parafile` fields to be set.

`sge_parallel_execute2` doubles the amount slots taken by the job (e.g. to avoid hyperthreading).

`sge_parallel_execute_batch` eliminates pre-processing on master. Job submitted from compute node. This job will occupy one slot for the duration of the NONMEM run. node. Job pre-processed and submitted from a compute node

Value

A `character` object.

See Also

`nm_getsetters()`.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv")) %>%
  cmd(sge_parallel_execute) %>%
  parafile("/opt/NONMEM/nm75/run/mpilinux8.pnm") %>%
  cores(8)

cmd(m1)

m2 <- m1 %>% child("m2") ## inherits same command as above

sge_parallel_execute ## view the character to see how psn interfaces with SGE
```

`shiny_nm` *Run monitor & summary app*

Description

[Stable]

Interactively monitor NONMEM runs. This interface is intentionally limited to monitoring runs, and does not include the ability to create, modify, launch or post-process runs since actions performed in the shiny app are not traceable/reproducible and not part of the workflow you create when scripting.

Usage

```
shiny_nm(m, envir = .GlobalEnv)
```

Arguments

`m`	Either an nm object, or data.frame or list or environment contain nm_lists.
`envir`	If missing, the environment to search.

Value

No return value, called for side effects.

Examples

```
if(interactive()){

#' # create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

shiny_nm() ## use all objects in global workspace
shiny_nm(m1) ## only m1

## if model objects are inside a tibble
d <- dplyr::tibble(m = m1)

shiny_nm(d$m) ## only d$m
shiny_nm(d) ## all nm_lists in d (data.frame/list/environment)

}
```

`show_ctl` *Show an uneditable version of the control file*

Description

[Stable]

Opens a read-only version of the NONMEM control file for browsing.

Usage

```
show_ctl(r)
```

Arguments

`r`	An nm object.

Value

No return value, called for side effects.

See Also

`show_out()`.

`show_out` *Show an uneditable version of the lst file*

Description

[Stable]

Opens a read-only version of the NONMEM control file for browsing.

Usage

```
show_out(r)
```

Arguments

`r`	An nm object.

Value

No return value, called for side effects.

See Also

`show_ctl()`.

`simple_field` *Interface for getting and setting your own simple fields in nm objects*

Description

[Stable]

Usage

```
simple_field(m, ...)
```

Arguments

`m`	An nm object.
`...`	Arguments to get/set fields.

Value

If ... contains an assignment, an nm object with modified field, otherwise returns the field value.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 <- m1 %>% simple_field(stars = 3)
m1 %>% simple_field(stars)
m1 ## see that stars is a field of the nm object.
```

`stage` *Stage files in project staging area ready for import*

Description

[Stable]

Staging is a preliminary step of bringing code from external to the project into the project. The intent is it remains a snapshot of code as it was at the time of `importing`. This aids in reproducibility because if that external code is changed, the staged code will remain fixed.

In practice, this function will rarely need to be used directly. The easiest way to bring code is via the "code library" RStudio 'Addin' shiny app.

Usage

```
stage(
  files,
  root_dir,
  overwrite = FALSE,
  silent = FALSE,
  find_replace_dir_names = TRUE
)
```

Arguments

`files`	Character vector. path of files to stage.
`root_dir`	Character path to root directory of `files`. Staged files relative to `staging` directory will be same as `files` to `root_dir`. If this is not specified, will guess based on presence of `nm_default_dirs`
`overwrite`	Logical (default = FALSE).

`silent`	Logical (default = FALSE).
`find_replace_dir_names`	Logical (default = TRUE). Will attempt to find replace strings in scripts to reflect `nm_default_dirs()`.

Value

A `tibble` with staged file information.

See Also

`code_library()`, `import()`

Examples

```
## requires NMproject directory structure
## Not run:

ls_code_library("Models/ADVAN2.mod") %>%
  stage()

## End(Not run)
```

`status` *Get status of NONMEM runs*

Description

[Stable]

Usage

```
status(x)
```

Arguments

`x`	An nm object.

Value

A character with the status of the run with values `"non started"`, `"running"`, `"finished"`, or `"error"`

See Also

`status_table()`.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

status(m1)  ## not run
```

`status_table` *Get status of multiple runs in form of table*

Description

[Stable]

A more friendly version of `status()` for vector valued nm objects. Useful after bootstraps, or stepwise covariate method steps, or any situation dealing with groups of NONMEM runs.

Usage

```
status_table(m)
```

Arguments

`m`	An nm object.

Value

A `tibble` object.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

c(m1, m1) %>% status_table()  ## both not started
```

subroutine *Subroutine*

Description

[Experimental]

Makes the necessary code changes to go from one ADVAN (and TRANS) to another.

Usage

```
subroutine(m, advan = NA, trans = 1, recursive = TRUE)
```

Arguments

m	An nm object.
advan	Character. desired ADVAN.
trans	Character. desired TRANS.
recursive	Logical (default = TRUE). Internal argument, do not modify.

Details

Can only switch between subroutines listed in available_advans.

Value

An nm object with modified ctl_contents field.

See Also

advan()

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))


advan(m1) ## 2
trans(m1) ## 1

m1 <- m1 %>% subroutine(advan = 2, trans = 2)

ds <- .available_advans %>%
  dplyr::filter(oral) %>%
  dplyr::mutate(
    m = m1 %>% child(run_id = label) %>%
      subroutine(advan = advan, trans = trans)
  )

ds

ds$m %>% dollar("PK")
```

`system_cmd` *System/shell command wrapper*

Description

[Stable]

Will run `getOption("system_cmd")`. A OS agnostic interface to the system terminal. Most of the time this will be the same as `system_nm` except when the PsN/NONMEM execution server is location in a different location to the RStudio server.

Usage

```
system_cmd(cmd, dir = ".", ...)
```

Arguments

`cmd`	Character. Command to send to shell.
`dir`	Optional character. Directory to run command in (default = current working directory)
`...`	Other arguments passed to system command.

Value

The return value of `getOption("system_cmd")`.

Examples

```
system_cmd("pwd")
```

system_nm — *System command for NONMEM execution*

Description

[Stable]

Not intended to be used directly in most cases. This is the function used by `run_nm()`. It can also be used directly to launch other PsN commands like `sumo`.

Usage

```
system_nm(cmd, dir = nm_dir("models"), ...)
```

Arguments

`cmd`	Character. System call to be sent to the terminal.
`dir`	Character. Directory (relative path) to run command in. By default this will be the "models" directory (`nm_dir("models")`).
`...`	Additional arguments to be passed to `system()` or `shell()`.

Value

The return value of `getOption("system_nm")`.

See Also

`run_nm()`

Examples

```
system_nm("hostname")

## requires NONMEM to be installed
## Not run:

system_nm("psn --versions")
system_nm("sumo run1.mod")

## End(Not run)
```

temp_files *Remove temporary NONMEM files*

Description

[Stable]

NONMEM produces a lot of temporary files which can add up to a lot of disk space. One strategy to remove this is to use the `clean` option in the PsN command. However, this can automatically remove files as soon as the run finishes that may be useful for debugging. `ls_tempfiles()` allows you to list the paths of all temporary files, for a single run or for all runs for inspection and deletion. `clean_tempfiles()` is a wrapper function that runs `ls_tempfiles()` and deletes everything returned. For safety is limited to only deleting files associated with `nm` objects though.

Usage

```
ls_tempfiles(
  object = ".",
  output_loc = c("run_dir", "base"),
  run_files = NA_character_,
  include_slurm_files = TRUE,
  ctl_extension = "mod",
  include_psn_exports = FALSE
)

clean_run(m, output_loc = c("run_dir", "base"), include_slurm_files = TRUE)

clean_tempfiles(
  object = ".",
  output_loc = c("run_dir", "base"),
  include_slurm_files = TRUE
)
```

Arguments

`object`	Either an nm object or path to project (default = `"."`). If a path is specified, the function will look for all runs in the directory (including subdirectories).
`output_loc`	Optional character for locating files. Either `"run_dir"` (default) for PsN execution or `"base"` for "nmfe" execution.
`run_files`	Optional character vector. Search amongst only these files instead. Default value `NA` searches based on `object`.
`include_slurm_files`	Logical (default = `TRUE`). Include files generated by Slurm.
`ctl_extension`	Character. Extension of control file (default = `"mod"`)
`include_psn_exports`	Logical (default = `FALSE`). Considers files that PsN exports to the `run_in` directory as temporary
`m`	An nm object

Details

Setting `include_psn_exports` = `TRUE` will break 'Pirana' and 'xpose' capability as these software use exported files.

Value

A `character` vector of temporary file paths

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

ls_tempfiles(m1) ## if no files, will be empty

m1 %>%
  ls_tempfiles() %>%
  unlink() ## delete all m1 temp files

## above line is equivalent to:
clean_tempfiles(m1)

ls_tempfiles() ## display all temp files in analysis project

ls_tempfiles() %>% unlink() ## remove all temp files in analysis project
```

`test_relations` *Generate tibble of covariate relations to test*

Description

[Stable]

The goal of NMproject's covariate modelling functions is to provide a stepwise covariate method *with manual decision* making. This important to ensure that the full model selection/evaluation criteria (should be defined in statistical analysis plans) can be applied at every step rather than just log likelihood ratio testing, where the most significant model may be unstable, may worsen model predictions or may only be slightly more significant than a more physiologically plausible covariate relationship.

The functions `test_relations()`, `covariate_step_tibble()`, `bind_covariate_results()` together comprise NMproject stepwise covariate method with manual decision. The goal is to be part way between PsN's SCM and completely manual process at each forward and backward elimination step. The syntax of how covariates are included is the same as PsN's SCM routine - See PsN documentation for more information.

Usage

```
test_relations(dtest, param, cov, state, continuous)
```

Arguments

`dtest`	Optional existing `dtest` to append (from an previous use `test_relations()`).
`param`	Character. Name of parameter(s).
`cov`	Character. Name of covariate(s).
`state`	Numeric or character. Number/name of state (see details).
`continuous`	Logical (default = `TRUE`). If `FALSE`, will treat the covariate as categorical.

Details

Setting vector values for `param`, `cov`, and `state`, will expand the grid to test each value with every other value greedily. This is similar to `expand.grid()` available states (see also `add_cov()`):

"2" or "linear" PARCOV= (1 + THETA(1)*(COV -median))

"3" or "hockey-stick" IF(COV.LE.median) PARCOV = (1 + THETA(1)*(COV - median)) IF(COV.GT.median) PARCOV = (1 + THETA(2)*(COV - median))

"4" or "exponential" PARCOV= EXP(THETA(1)*(COV - median))

"5" or "power" PARCOV= ((COV/median)**THETA(1))

"power1" PARCOV= ((COV/median))

"power0.75" PARCOV= ((COV/median)**0.75)

"6" or "log-linear" PARCOV= (1 + THETA(1)*(LOG(COV) - log(median)))

Value

A `tibble` describing relationships to test.

See Also

`add_cov()`, `covariate_step_tibble()`, `bind_covariate_results()`

Examples

```
dtest <- test_relations(param = c("KA", "K", "V"),
                        cov = c("LIN1", "LIN2", "LIN3", "RND1", "RND2", "RND3"),
                        state = c("linear", "power"),
                        continuous = TRUE) %>%
         test_relations(param = c("KA", "K", "V"),
                        cov = "BN1",
                        state = "linear",
                        continuous = FALSE)

dtest
```

`update_parameters` *Update initial estimates to final estimates*

Description

[Stable]

Usage

```
update_parameters(ctl, from)
```

Arguments

`ctl`	An nm object.
`from`	Optional nm object. The completed object from which to extract results. If not specified, `from` will be taken to be `ctl`.

Value

An nm object with modified `ctl_contents` field.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))


m1 %>% dollar("THETA")

## requires NONMEM to be installed
## Not run:
m1 %>% run_nm() %>% wait_finish()
m1 <- m1 %>% update_parameters()
m1 %>% dollar("THETA")

## End(Not run)
```

`wait_finish` *Wait for runs to finish*

Description

[Stable]

Blocks subsequent r execution until run(s) are finished. This is useful for when subsequent relies on outputs from completed NONMEM jobs. It is normally a good idea to include this in post processing R markdown templates, to ensure they wait for runs to complete before executing.

Usage

```
wait_finish(r, timeout = NA)
```

Arguments

`r`	An nm object.
`timeout`	Numeric seconds to wait before timeout.

Value

Invisibly returns `r` unmodified. Called for side effects.

Examples

```
## requires NONMEM to be installed

## Not run:
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

m1 %>%
  run_nm() %>%
  wait_finish()

## following requires run to be completed.
covariance_plot(m1)

## End(Not run)
```

wait_for — *Wait for statement to be TRUE*

Description

[Stable]

Will block R console until an expression evaluates to be `TRUE`.

Usage

```
wait_for(x, timeout = NULL, interval = 1)
```

Arguments

`x`	Boolean expression to evaluate.
`timeout`	Numeric. Maximum time (in seconds) to wait.
`interval`	Numeric. The polling interval in seconds (default=1).

Value

Invisibly returns `TRUE` indicating value of `x` after waiting for `x` to be `TRUE`.

See Also

`wait_finish()`.

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

## requires NONMEM to be installed
## Not run:

## the following are identical
m1 %>% run_nm() %>% wait_finish()

wait_for(is_finished(m1)) ## wait_finish is a more convenient form of this

## End(Not run)
```

`wipe_run` *Wipe previous run files*

Description

[Stable]

Will remove all the output files generated by a previously completed run. This is run by `run_nm()` prior to launching any jobs to ensure that output files from old runs do not get mistaken for up-to-date runs.

Usage

```
wipe_run(r)
```

Arguments

`r`	An nm object.

Value

No return value, called for side effects.

`write_derived_data` *Write derived data file*

Description

[Stable]

Will write a dataset and an .RDS version of it to the (by default) "DerivedData" directory. The main benefit of the .RDS dataset is that functions like `input_data()` and `output_table()` can use it for rapid reading speeding up overall function.

Usage

```
write_derived_data(d, name, ...)
```

Arguments

`d`	A data.frame. Data frame to be saved.
`name`	Character. Name of file (with or without extension). If not a path, will save to DerivedData directory.
`...`	Additional arguments to be passed to `utils::write.csv()`.

Details

If there is no "DerivedData" data directory and you are using a different structure the argument `name` must be a (relative) path to an existing directory where you want your NONMEM ready dataset to be stored.

Value

No return value, called for side effects.

See Also

`read_derived_data()`, `input_data()`, `exclude_rows()`

Examples

```
## requires NMproject directory structure to operate in
## Not run:

## read a dataset that's been copie into SourceData
d <- read.csv("SourceData/orig_data.csv")

## modify it
d <- d[d$ID < 10, ]

d %>% write_derived_data("DerivedData/data.csv")

## load it again either with
d <- read_derived_data("data")

## or more commonly if it is associated with run (e.g. m1),
## you can use input_data() to load it via the nm object

d <- input_data(m1)

## End(Not run)
```

`%f>%` *Function pipe for nm objects*

Description

[Experimental]

Pipe an nm object object to a list of functions. Although this enables multiple NONMEM runs to be handled simultaneously, it does make your code less readable.

Usage

```
lhs %f>% rhs
```

Arguments

`lhs`	An nm object.
`rhs`	A list of functions. Must be same length as `lhs`.

Value

A modified nm object.

See Also

`child()` for creating multiple child NONMEM objects

Examples

```
# create example object m1 from package demo files
exdir <- system.file("extdata", "examples", "theopp", package = "NMproject")
m1 <- new_nm(run_id = "m1",
             based_on = file.path(exdir, "Models", "ADVAN2.mod"),
             data_path = file.path(exdir, "SourceData", "THEOPP.csv"))

temp_data_file <- paste0(tempfile(), ".csv")

## dataset has missing WTs so create a new one and assign this to the run
input_data(m1) %>%
  dplyr::group_by(ID) %>%
  dplyr::mutate(WT = na.omit(WT)) %>%
  write_derived_data(temp_data_file)

m1 <- m1 %>% data_path(temp_data_file)

mWT <- m1 %>% child(c("m2", "m3", "m4")) %f>%
list(
 . %>% add_cov(param = "V", cov = "WT", state = "linear"),
 . %>% add_cov(param = "V", cov = "WT", state = "power"),
 . %>% add_cov(param = "V", cov = "WT", state = "power1")
)

mWT %>% dollar("PK")

unlink(temp_data_file)
```

Index