Package ‘nlmixr2auto’

February 6, 2026

Type Package

Title Automated Population Pharmacokinetic Modeling

Version 1.0.0

Description Automated population pharmacokinetic modeling framework for data-driven initialisation, model evaluation, and metaheuristic optimization. Supports genetic algorithms, ant colony optimization, tabu search, and stepwise procedures for automated model selection and parameter estimation within the nlmixr2 ecosystem.

License GPL (>= 3)

URL https://github.com/ucl-pharmacometrics/nlmixr2auto

BugReports https://github.com/ucl-pharmacometrics/nlmixr2auto/issues

Depends R (>= 4.1.0), nlmixr2data

Imports nlmixr2, nlmixr2est, nlmixr2autoinit, rxode2, dplyr, progressr, processx (> 3.8.0), withr, crayon

Encoding UTF-8

Language en-US

NeedsCompilation no

RoxygenNote 7.3.1

Author Zhonghui Huang [aut, cre],
Joseph Standing [ctb],
Matthew Fidler [ctb],
Frank Kloprogge [ctb]

Maintainer Zhonghui Huang <huangzhonghui22@gmail.com>

Repository CRAN

Date/Publication 2026-02-06 19:40:02 UTC

Contents

`.twoBitCode` *2-bit code helper*

Description

Internal utility used by decodeBinary() and encodeBinary() to convert between a 2-bit representation and categorical values via lookup tables.

Usage

```
.twoBitCode(
  mode = c("decode", "encode"),
  value,
  bit2 = NULL,
  decode_map = NULL,
  encode_map = NULL,
  param_name = "param"
)
```

Arguments

`mode`	Character, either "decode" or "encode".
`value`	For decode: the first bit (0/1). For encode: the categorical value to encode.
`bit2`	For decode: the second bit (0/1). Ignored for encode.
`decode_map`	Numeric vector of length 4 used for decoding, in the order corresponding to 00, 01, 10, 11. Use NA for illegal codes.
`encode_map`	Named integer/numeric vector mapping categorical values (as names) to integer codes 0..3 (corresponding to 00..11).
`param_name`	Character, parameter name used in error messages.

Details

The helper supports two modes:

- decode: converts (bit1, bit2) to a categorical value using a length-4 lookup table decode_map corresponding to 00, 01, 10, 11.
- encode: converts a categorical value to a 2-bit code using a named lookup table encode_map mapping values to integer codes 0..3 (corresponding to 00..11).

Value

For decode: a single numeric categorical value. For encode: an integer vector length 2 containing bits c(bit1, bit2).

Author(s)

Zhonghui Huang

See Also

decodeBinary, encodeBinary

Examples

```
# Decode example (00/01 map to level 1).
.twoBitCode("decode", 0, 0, decode_map = c(1, 1, 2, 3))  # 1
.twoBitCode("decode", 0, 1, decode_map = c(1, 1, 2, 3))  # 1
.twoBitCode("decode", 1, 0, decode_map = c(1, 1, 2, 3))  # 2
.twoBitCode("decode", 1, 1, decode_map = c(1, 1, 2, 3))  # 3

# Encode example (level 1 emits 01).
encode_map <- stats::setNames(c(1, 2, 3), c(1, 2, 3))
.twoBitCode("encode", 1, encode_map = encode_map)  # c(0, 1)
.twoBitCode("encode", 2, encode_map = encode_map)  # c(1, 0)
.twoBitCode("encode", 3, encode_map = encode_map)  # c(1, 1)

# Decode 4-level example (00..11 map to 1..4).
.twoBitCode("decode", 0, 0, decode_map = c(1, 2, 3, 4))  # 1
```

---

aco.operator *ACO operator for model selection*

Description

Implements an ant colony optimization algorithm to explore model space and identify the best-performing model given pre-defined fitness function.

Usage

```
aco.operator(
  dat,
  param_table = NULL,
  search.space = c("ivbase", "oralbase"),
  no.cores = NULL,
  aco.control = acoControl(),
  penalty.control = penaltyControl(),
  precomputed_results_file = NULL,
  foldername = NULL,
  filename = "test",
  seed = 1234,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`aco.control`	A list of ACO control parameters defined by `acoControl()`. Includes: • nants - number of ants per iteration. • niter - maximum number of iterations. • rho - pheromone evaporation rate. • phi0 - initial pheromone level. • phi_min, phi_max - bounds for pheromone levels. • alpha - pheromone weight exponent. • elite - proportion of best solutions preserved each iteration. • prob.min - minimum sampling probability. • diff_tol - threshold for significant fitness difference.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`foldername`	Character string specifying the folder name for storing intermediate results. If `NULL` (default), `tempdir()` is used for temporary storage. If specified, a cache directory is created in the current working directory.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`seed`	Integer. Random seed controlling the random sampling steps of the ant colony optimization operator for reproducible runs. Default is 1234.
`.modEnv`	Optional environment used internally to store model indices, cached parameter tables, and results across steps.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to `mod.run()`.

Details

The ACO approach uses a colony of "ants" to stochastically sample models, evaluate their fitness, and update pheromone trails that guide future searches. This iterative process balances exploration of new models with exploitation of promising candidates.

Value

An object of class `"acoOperatorResult"`, containing:

- `` $`Final Selected Code` `` - Vector representation of the best model.
- `` $`Final Selected Model Name` `` - Human-readable name of the selected model.
- `` $`Model Run History` `` - Data frame of model runs across iterations.
- `` $`Node Run History` `` - History of pheromone probabilities for each iteration.

Author(s)

Zhonghui Huang

See Also

`acoControl`, `penaltyControl`, `auto_param_table`, `mod.run`, `ppkmodGen`

Examples

```
# Example usage with phenotype dataset
outs <- aco.operator(
  dat = pheno_sd,
  param_table = NULL,
  search.space = "ivbase",
  aco.control = acoControl(),
  saem.control = nlmixr2est::saemControl(
    seed = 1234,
    nBurn = 200,
    nEm   = 300,
    logLik = TRUE
  )
)
print(outs)
```

acoControl *Create control parameters for the ACO algorithm*

Description

Creates a list of control settings for the `aco.operator` function.

Usage

```
acoControl(
  nants = 15,
  niter = 20,
  Q = 1,
  rho = 0.5,
  phi0 = 2,
  phi_min = 1,
  phi_max = Inf,
  alpha = 1,
  elite = 0,
  prob_min = 0.2,
  diff_tol = 1
)
```

Arguments

`nants`	Integer. Number of ants (candidate solutions) generated at each iteration. Defaults to 15.
`niter`	Integer. Maximum number of ACO iterations. Defaults to 20.
`Q`	A positive numeric value. Pheromone scaling constant controlling the amount of pheromone deposited by high-quality solutions during each iteration. Defaults to 1.
`rho`	Numeric in (0, 1). Pheromone evaporation rate. Higher values increase evaporation, encouraging exploration. Defaults to 0.5.
`phi0`	A non-negative numeric value. Initial pheromone value assigned to all nodes at the start of the search. Defaults to 2.
`phi_min`	A non-negative numeric value. Lower bound for pheromone values, preventing premature convergence. Defaults to 1.
`phi_max`	A non-negative numeric value. Upper bound for pheromone values, limiting excessive reinforcement. Defaults to Inf.
`alpha`	A non-negative numeric value. Exponent controlling the influence of pheromone values on the probability of selecting a component during solution construction. Defaults to 1.
`elite`	Numeric. Elitism rate between 0 and 1. Specifies the proportion of elite ants whose solutions are preserved and directly propagated to the next iteration. Defaults to 0.

`prob_min`	Numeric. Minimum probability floor between 0 and 1. Applied during solution construction to avoid zero-probability choices. Defaults to 0.2.
`diff_tol`	Numeric. Significance difference threshold used for ranking. Values within this threshold are considered equal and receive the same rank. Default is 1.

Value

A named list containing all ACO control parameters.

Author(s)

Zhonghui Huang

Examples

```
acoControl()
```

`add_covariate` *Add a covariate effect to a parameter model*

Description

Automates the creation of covariate effects in pharmacometric models by generating appropriate beta coefficients and modifying model expressions. Supports both standard allometric scaling rules and custom covariate effects.

Usage

```
add_covariate(
  param_name,
  covariate_var,
  param_model,
  beta_value = NULL,
  existing_betas = c(),
  use_fix = TRUE
)
```

Arguments

`param_name`	Character. Target parameter name (e.g., "cl", "vc").
`covariate_var`	Character. Covariate variable name (e.g., "WT", "BMI").
`param_model`	Character. Current parameter model expression (e.g., "cl = exp(tcl)").
`beta_value`	Numeric. Optional fixed beta value. If NULL, uses built-in rules.
`existing_betas`	Character vector. Existing beta definitions to append to.
`use_fix`	Logical. Use `fix()` for beta values? Default TRUE.

Details

Automatic beta selection rules:

- Standard covariates ("wt"/"ffm"/"bmi"/"bsa"):
  - 0.75 for clearance parameters (cl/q/q2)
  - 1.0 for volume parameters (vc/vp/vp2)
- Other covariates: Default beta = -0.1 with message

Value

List with two elements:

- betas - Updated character vector of beta definitions
- mod - Modified model expression with covariate term

Author(s)

Zhonghui Huang

Examples

```
# Add weight effect to clearance
 add_covariate( "cl", "WT", "cl = exp(tcl)")

# Custom beta value for BMI effect
add_covariate(
  "vc", "BMI", "vc = exp(tvc)",
  beta_value = -0.2, use_fix = FALSE
)
```

`add_variability` *Add inter-individual variability to a parameter*

Description

Defines a model string for a parameter, optionally adding inter-individual variability.

Usage

```
add_variability(param_name, eta_flag, param_table, param.type = 1)
```

Arguments

`param_name`	Character. The name of the parameter.
`eta_flag`	Integer. If 1, inter-individual variability is added; otherwise, it is not.
`param_table`	Data frame. A table containing parameter details with columns `Name`, `init`, and optionally bounds like `lb` and `ub`.
`param.type`	Integer. Transformation type: 1=Exponential, 2=Logistic. Defaults to 1.

Value

A list containing:

`mod`	Character. The model string for the parameter.
`eta_init`	Character. The initialization string for the variability parameter (if applicable).

Author(s)

Zhonghui Huang

Examples

```
param_table <- initialize_param_table()
add_variability("cl", 1, param_table)
```

`applyParamDeps` *Apply parameter dependency rules*

Description

Applies dependency constraints among structural and statistical flags in a model-code parameter list to produce a feasible combination.

Usage

```
applyParamDeps(params)
```

Arguments

`params`	Named list of model-code parameters. Elements are typically scalar categorical values or 0/1 flags. Unknown elements are ignored.

Details

Corrections are applied in the following groups:

- Compartment rules: disable peripheral IIV terms when "no.cmpt" implies they are not used.
- Michaelis-Menten rules: enable or disable "eta.vmax", "eta.km", and "eta.cl" based on "mm".
- Oral absorption rules: enable or disable oral-related terms based on "abs.delay", "abs.type", and "abs.bio".
- Correlation rules: disable "mcorr" when too few IIV terms are present.
- IIV requirement: ensure at least one IIV term is present by enabling a default term consistent with "mm".

Value

A named list with corrected parameter values.

Author(s)

Zhonghui Huang

Examples

```
params <- list(
  no.cmpt = 1, mm = 0, mcorr = 1,
  eta.vc = 1, eta.cl = 0, eta.vp = 1, eta.q = 1
)
applyParamDeps(params)

params2 <- list(
  no.cmpt = 2, mm = 1,
  eta.vmax = 0, eta.km = 0, eta.cl = 1
)
applyParamDeps(params2)
```

`auto_param_table` *Automatically generate a parameter table with initial estimates*

Description

Constructs a parameter table for nlmixr2 model fitting. It supports:

- Direct use of a user-provided parameter table.
- Automatic initialization of parameters from data using `getPPKinits()`.
- Fallback to a default parameter table created by `initialize_param_table()`.

Usage

```
auto_param_table(
  dat = NULL,
  param_table = NULL,
  nlmixr2autoinits = TRUE,
  foldername = NULL,
  filename = "test",
  out.inits = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing observed data (required if `nlmixr2autoinits` = `TRUE`).
`param_table`	Optional. A user-provided parameter table (if provided, all other logic is skipped).
`nlmixr2autoinits`	Logical. Whether to automatically estimate initial values using `getPPKinits()`. Default is `TRUE`.

`foldername`	Character string specifying the folder name for storing `nlmixr2autoinits` outputs. If `NULL` (default), `tempdir()` is used for temporary storage. If specified, a cache directory is created in the current working directory.
`filename`	Character string specifying the base name for model output files generated during evaluation.
`out.inits`	Logical flag indicating whether the results returned by the automated initialization procedure should be saved to an RDS file. When TRUE, the output of the initialization step is written to disk for reproducibility or debugging purposes.
`...`	Additional arguments passed to `getPPKinits()`.

Details

When `nlmixr2autoinits = TRUE`, this function estimates initial values from data, applies a name mapping to internal model parameters, performs log transformations where appropriate, and replaces problematic log values (e.g. log(0) or `NA`) with `log(0.01)` for numerical stability.

Value

A `data.frame` representing the parameter table with initial estimates, ready for use in `nlmixr2()`.

Author(s)

Zhonghui Huang

See Also

`getPPKinits`, `initialize_param_table`

Examples

```
auto_param_table(dat = pheno_sd)
```

`base_model` *Create a base model code for single-start model search algorithms*

Description

Constructs a named numeric vector defining the initial structural and inter-individual variability model configuration used in single-start automated PK model search algorithms.

Usage

```
base_model(search.space = "ivbase")
```

Arguments

`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".

Details

Two search spaces are supported: "ivbase" and "oralbase". A user-specified initial model code can be provided via the custom_base argument. The input is validated for numerical type and expected length, and standardized element names are applied before returning. The function is currently used in stepwise selection and tabu search routines, where a single starting model is iteratively updated.

Value

For search.space = "ivbase": a named integer vector of length 9 containing:

- no.cmpt - Number of compartments
- eta.km - IIV flag for K_m
- eta.vc - IIV flag for V_c
- eta.vp - IIV flag for V_p
- eta.vp2 - IIV flag for V_{p2}
- eta.q - IIV flag for Q
- eta.q2 - IIV flag for Q_2
- mm - Michaelis–Menten term flag
- mcorr - Correlation flag among ETAs
- rv - Residual error model code

For search.space = "oralbase": a named integer vector of length 11, including all fields above plus:

- eta.ka - IIV flag for k_a (oral absorption rate constant)

Author(s)

Zhonghui Huang

Examples

```
base_model("ivbase")
base_model("oralbase")
```

`build_odeline` *Build ODE model lines for pharmacokinetic modeling*

Description

Constructs a system of ordinary differential equations (ODEs) for pharmacokinetic modeling with various configurations including different absorption models, compartmental structures, and elimination kinetics.

Usage

```
build_odeline(
  mm = 0,
  no.cmpt = 1,
  route = "bolus",
  abs.bio = 0,
  abs.type = 1,
  abs.delay = 0
)
```

Arguments

`mm`	Michaelis-Menten elimination flag. 0 = linear elimination (default), 1 = Michaelis-Menten elimination.
`no.cmpt`	Number of compartments. Supported values: 1 (central only), 2 (central + peripheral), or 3 (central + 2 peripheral).
`route`	Administration route. Options: "bolus" (IV), "oral", or "mixed_iv_oral".
`abs.bio`	Bioavailability estimation flag. 0 = no bioavailability estimation (default), 1 = include bioavailability parameter (F1).
`abs.type`	Absorption type for oral route: • 1 = First-order absorption (default) • 2 = Zero-order absorption • 3 = Sequential zero-first order absorption • 4 = Dual zero-first order absorption
`abs.delay`	Absorption delay type: • 0 = No delay (default) • 1 = Lag time (tlag) • 2 = Transit compartment model

Details

Parameter Constraints: The function includes error checking for incompatible parameter combinations:

- abs.bio=1 cannot be used with abs.type=4 or abs.delay=3
- Dual absorption (abs.type=4) not supported for mixed routes

Value

A character vector containing ODE equations for the specified configuration. Includes differential equations for drug compartments, absorption models, and derived parameters.

Author(s)

Zhonghui Huang

Examples

```
# Two-compartment model with first-order absorption
build_odeline(no.cmpt = 2, route = "oral")

# One-compartment IV model with Michaelis-Menten elimination
build_odeline(mm = 1, route = "bolus")
```

`create.pop` *Create an initial GA population*

Description

Generates an initial population for a genetic algorithm (GA). Each individual is a binary chromosome represented by a numeric vector containing 0 and 1.

Usage

```
create.pop(npop, nbits)
```

Arguments

`npop`	Integer. Number of individuals (chromosomes) in the population.
`nbits`	Integer. Number of bits in each chromosome.

Details

Bits are sampled independently. Each bit takes the value 0 or 1 with equal probability.

Value

A numeric matrix with npop rows and nbits columns containing only 0 and 1. Each row corresponds to one chromosome.

Author(s)

Zhonghui Huang

Examples

```
create.pop(npop = 10, nbits = 12)
```

createAnts *Create ant population for ACO*

Description

Generate a population of ants (candidate models) for use in an ant colony optimization algorithm for pharmacometric model search.

Usage

```
createAnts(
  search.space = "ivbase",
  nants = 15,
  init = FALSE,
  nodeslst = NULL,
  custom_config = NULL,
  fixed = NULL
)
```

Arguments

`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`nants`	Integer. Number of ants (candidate solutions) generated at each iteration. Defaults to 15.
`init`	Logical. If TRUE, a subset of ants is initialized as fixed base models and the remaining ants are generated by probabilistic sampling. If FALSE, all ants are generated by probabilistic sampling.
`nodeslst`	Data frame containing pheromone information used for probabilistic sampling. It must include node identifiers and associated sampling probabilities. This argument is required whenever random ants are generated.
`custom_config`	Optional named list defining a custom parameter structure. If provided, the parameter names are taken from the names of this list. If NULL, a default parameter structure is used based on the selected search space.
`fixed`	Optional list specifying fixed ants for initialization. The list may contain the following elements: • n: number of fixed ants. • mat: optional matrix specifying fixed ant encodings, with parameters in rows and ants in columns.

Details

Each ant is represented as a column vector encoding discrete structural model decisions, including the number of compartments, inclusion of random effects, Michaelis–Menten elimination, correlation structures, and residual error models. The set of parameters included in the encoding depends on the selected search space.

Ants are generated using a combination of fixed initialization and pheromone-guided probabilistic sampling. When fixed initialization is enabled, a subset of ants corresponds to predefined base models, such as one- to three-compartment structures with different residual error models. The remaining ants are sampled according to probability distributions derived from pheromone weights stored in the node list.

Structural dependencies between parameters are enforced during generation. For example, parameters associated with peripheral compartments are only active when the number of compartments is sufficient, and parameters related to Michaelis–Menten elimination are only sampled when the corresponding mechanism is selected. Parameters that are not applicable for a given structure are encoded with a value of -1.

Value

A numeric matrix in which rows correspond to model parameters and columns correspond to individual ants. Column names identify ants sequentially.

Parameter values are encoded as integers. Binary indicators represent exclusion or inclusion of model components, categorical values represent multi-level structural choices, and the value -1 indicates that a parameter is not applicable for the given model structure.

Author(s)

Zhonghui Huang

See Also

`initNodeList`, `aco.operator`

Examples

```
# Example 1: Use defaults (6 fixed base models)
nodes <- initNodeList(search.space = "ivbase", phi0 = 1)
createAnts(
  search.space = "ivbase",
  nants = 15,
  init = TRUE,
  nodeslst = nodes
)

# Example 2: Custom number of fixed ants
createAnts(
  search.space = "ivbase",
  nants = 20,
  init = TRUE,
  nodeslst = nodes,
  fixed = list(n = 10)  # 10 fixed, mat = NULL (auto-generate)
)

# Example 3: Custom fixed models
my_models <- matrix(
  c(1, -1, 0, -1, -1, -1, -1, 0, 0, 1,
    2, -1, 1,  0, -1,  0, -1, 0, 0, 2,
    3, -1, 1,  1,  0,  1,  0, 1, 1, 3),
  nrow = 10, ncol = 3
)
rownames(my_models) <- c("no.cmpt", "eta.km", "eta.vc", "eta.vp",
                         "eta.vp2", "eta.q", "eta.q2", "mm", "mcorr", "rv")

createAnts(
  search.space = "ivbase",
  nants = 10,
  init = TRUE,
  nodeslst = nodes,
  fixed = list(n = 3, mat = my_models)
)
```

`decodeBinary` *Decode binary encoding to categorical encoding*

Description

Converts a binary-encoded GA chromosome (0/1 vector) into a categorical parameter vector.

Usage

```
decodeBinary(binary_string, search.space = "ivbase", custom_config)
```

Arguments

`binary_string`	Numeric vector of bits (0/1). Length must match the expected layout for the selected search.space.
`search.space`	Character string specifying which search space to use. Options are "ivbase", "oralbase", or "custom". Default is "ivbase".
`custom_config`	Optional named list defining a custom parameter structure. If provided, the parameter names are taken from the names of this list. If NULL, a default parameter structure is used based on the selected search space.

Details

Supported search spaces:

- "ivbase": binary string has 12 bits and decodes to 10 values.
- "oralbase": binary string has 13 bits and decodes to 11 values.
- "custom": binary string has 29 bits and decodes to 24 values.

Legacy layout ("ivbase" and "oralbase"):

- The first two bits encode the number of compartments (no.cmpt) as 00 or 01 for 1, 10 for 2, and 11 for 3.
- The middle parameters are copied directly and preserve values such as 0, 1, and -1.

- The last two bits encode the residual error model (rv) as 00 or 01 for 1, 10 for 2, and 11 for 3.

Custom layout ("custom"):

- Multi-level parameters are stored as 2-bit fields (for example, no.cmpt, absorption type, absorption delay, residual error model, and allometric scaling).
- Binary flags are stored as single bits, including absolute bioavailability (abs.bio), the Michaelis-Menten indicator (mm), and the correlation indicator (mcorr).
- Inter-individual variability indicators (eta.*) are stored as 16 single-bit flags in a fixed order.

For "custom", the categorical output order is:

```
no.cmpt, abs.type, abs.delay, abs.bio,
eta.vmax, eta.km, eta.cl, eta.vc, eta.vp, eta.vp2, eta.q, eta.q2,
eta.ka, eta.tlag, eta.D2, eta.F1, eta.Fr, eta.mtt, eta.n, eta.bio,
mm, mcorr, rv, allometric_scaling
```

Value

Numeric vector of categorical parameter values.

Author(s)

Zhonghui Huang

See Also

`.twoBitCode`, `encodeBinary`

Examples

```
binary_iv <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1)
decodeBinary(binary_iv, "ivbase")

binary_oral <- c(0, 1, rep(0, 9), 1, 1)
decodeBinary(binary_oral, "oralbase")

binary_custom <- c(
  0, 0, 0, 1, 1, 1, 1, rep(0, 16), 0, 1, 1, 0, 0, 1
)
decodeBinary(binary_custom, "custom")
```

`detect_move` *Detect the primary move between two model codes*

Description

Compares a previous model code with a new one and identifies the primary intended change. If an `original_neighbor` is provided, this is used to determine the intended change, ignoring any secondary modifications introduced by validation.

Usage

```
detect_move(prev_string, new_string, original_neighbor = NULL)
```

Arguments

`prev_string`	A named numeric vector: the starting model code.
`new_string`	A named numeric vector: the validated model code.
`original_neighbor`	Optional named numeric vector: the original neighbor before validation. If provided, this is used to identify the primary change.

Value

A list with `element`, `from`, and `to` describing the primary change.

Author(s)

Zhonghui Huang

Examples

```
prev <- c(no.cmpt = 2, eta.vc = 1)
orig <- c(no.cmpt = 3, eta.vc = 1)  # original neighbor
new  <- c(no.cmpt = 3, eta.vc = 0)  # validated neighbor (extra fix)
detect_move(prev, new, original_neighbor = orig)
```

`encodeBinary` *Encode categorical encoding to binary encoding*

Description

Converts a categorical parameter vector into a binary-encoded GA chromosome.

Usage

```
encodeBinary(categorical_string, search.space = "ivbase", custom_config)
```

Arguments

`categorical_string`	Numeric vector of categorical parameter values. Length must match the expected layout for the selected `search.space`.
`search.space`	Character string specifying which search space to use. Options are "ivbase", "oralbase", or "custom". Default is "ivbase".
`custom_config`	Optional named list defining a custom parameter structure. If provided, the parameter names are taken from the names of this list. If NULL, a default parameter structure is used based on the selected search space.

Details

This function converts a vector of categorical parameter values into a 0/1 bit string (a GA chromosome). The required input layout and the encoding rules depend on the selected search space.

Built-in search spaces: ivbase / oralbase

- Expected input length:
  - ivbase: 10 categorical values
  - oralbase: 11 categorical values
- Structure: the first value is no.cmpt and the last value is rv. All middle values are copied as-is (they are expected to be 0/1 flags). Specifically, ivbase copies indices 2..9 and oralbase copies indices 2..10.
- no.cmpt and rv use the legacy 3-level 2-bit encoding:
  - 1 -> 01
  - 2 -> 10
  - 3 -> 11

  The code 00 is not used in this mapping.
- Output length:
  - ivbase: 12 bits (2 + 8 + 2)
  - oralbase: 13 bits (2 + 9 + 2)

Custom search space: custom

For search.space = "custom", the function reads categorical values in the order given by params. If custom_config$params is provided and non-empty, that vector defines the parameter names and order; otherwise, the default 24-parameter layout is used:

```
no.cmpt, abs.type, abs.delay, abs.bio,
eta.vmax, eta.km, eta.cl, eta.vc, eta.vp, eta.vp2, eta.q, eta.q2,
eta.ka, eta.tlag, eta.D2, eta.F1, eta.Fr, eta.mtt, eta.n, eta.bio,
mm, mcorr, rv, allometric_scaling
```

- Parameters encoded with 2 bits: no.cmpt, abs.type, abs.delay, rv, and allometric_scaling.
- All other parameters must be single-bit flags (0 or 1) and are appended directly to the chromosome.

2-bit encoding rules

- no.cmpt (1..3): 1->01, 2->10, 3->11 (00 unused)
- abs.type (1..4): 1->00, 2->01, 3->10, 4->11
- abs.delay (0..2): 0->00, 1->01, 2->10 (11 unused)
- rv (1..4): 1->00, 2->01, 3->10, 4->11
- allometric_scaling (0..3): 0->00, 1->01, 2->10, 3->11

Value

Numeric vector of bits (0/1).

Author(s)

Zhonghui Huang

See Also

.twoBitCode, decodeBinary

Examples

```
# ivbase: 10 categorical -> 12 bits
cat_iv <- c(1, rep(0, 8), 3)
encodeBinary(cat_iv, "ivbase")

# Custom: 24 categorical -> 29 bits
cat_custom <- c(
  1, 2, 0, 1,
  rep(0, 16),
  0, 1, 3, 1
)
encodeBinary(cat_custom, "custom")
```

fitness	*Evaluate fitness of a population pharmacokinetic model*

Description

Evaluates the quality of a fitted model based on parameter bounds and diagnostic thresholds.

Usage

```
fitness(
  search.space = "ivbase",
  fit = NULL,
  dat = NULL,
  penalty.control = penaltyControl(),
  objf = "BIC"
)
```

Arguments

`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`fit`	Data frame. Model summary from tools such as `get.mod.lst()`, with parameter estimates and diagnostics.
`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`penalty.control`	List created using `penaltyControl()`, including: **penalty.value** Numeric. Default penalty multiplier used in binary violations. **step.penalties** Numeric vector or list. Penalties applied to step violations (mild, severe). **bounds** List of parameter lower/upper bounds, typically from `param.bounds()`. **thresholds** Named list of diagnostic constraints (e.g., RSE, shrinkage). Each contains a method ("binary" or "step") and the corresponding threshold or step levels. **penalty.terms** Character vector of constraint categories to penalize. Valid terms include "theta", "rse", "omega", "shrinkage", "sigma", "correlation", "covariance", and "total".
`objf`	Character. Column name in fit used as the base objective function (e.g., "AIC", "BIC", "OBJFV").

Value

A data frame extending fit with the following:

- flag.* columns: indicators of constraint violations (0 = no violation, 1 = mild, 2 = severe).
- count.constraint.* columns: number of violations per constraint type.
- fitness: penalized objective function value, computed from the specified objf plus applicable penalties.

Author(s)

Zhonghui Huang

See Also

penaltyControl(), param.bounds().

Examples

```
# Fit a model (using nlmixr2)
pheno <- function() {
  ini({
    tcl <- log(0.008)
    tv <-  log(0.6)
    eta.cl + eta.v ~ c(1,
                       0.01, 1)
    add.err <- 0.1
  })
  model({
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    ke <- cl / v
    d/dt(A1) = - ke * A1
    cp = A1 / v
    cp ~ add(add.err)
  })
}
fit <- nlmixr2est::nlmixr2(pheno, pheno_sd, "saem", control = list(print = 0),
              table = list(cwres = TRUE, npde = TRUE))
Store. <- get.mod.lst(fit.s = fit, 1)
 fitness(fit = Store.,dat = pheno_sd)
```

`ga.crossover` *Crossover operator (one- or two-point) for binary chromosomes*

Description

Apply one- or two-point crossover to a selected population of binary chromosomes.

Usage

```
ga.crossover(sel.population, pcross, npop, nbits)
```

Arguments

`sel.population`	Numeric matrix of dimension npop by nbits. Each row is a chromosome and is expected to contain binary values (0/1).
`pcross`	Single numeric value in $[0, 1]$ giving the probability of applying crossover to each parent pair.
`npop`	Single positive even integer giving the population size.
`nbits`	Single positive integer giving the chromosome length.

Details

Parents are paired sequentially (1 with 2, 3 with 4, etc.). For each pair, crossover is applied with probability `pcross`; otherwise the parents are copied unchanged. Crossover points are sampled from 0:`nbits`. If the sampled points do not yield a valid crossover, no crossover is performed for that pair.

Value

Numeric matrix containing the children population after crossover.

Author(s)

Zhonghui Huang

Examples

```
sel.population <- matrix(sample(0:1, 100, replace = TRUE), nrow = 10)
ga.crossover(sel.population = sel.population, pcross = 0.7, npop = 10, nbits = 10)
```

`ga.mutation` *Mutation operator for binary genetic algorithms*

Description

Mutate a binary population by flipping bits with probability pmut.

Usage

```
ga.mutation(children.cross, pmut)
```

Arguments

`children.cross`	Numeric matrix containing the child population. Rows are individuals and columns are bits. Values are expected to be 0/1.
`pmut`	Single numeric value in $[0, 1]$ giving the per-bit mutation probability.

Details

Mutation is applied independently to each bit (gene). For each position, a Bernoulli trial with success probability pmut determines whether the bit is flipped (0 becomes 1, 1 becomes 0).

Value

Numeric matrix containing the mutated population.

Author(s)

Zhonghui Huang

Examples

```
children.cross <- matrix(sample(0:1, 120, replace = TRUE), nrow = 10)
ga.mutation(children.cross, pmut = 0.1)
```

`ga.operator` *Genetic algorithm operator for model selection*

Description

Run a genetic algorithm to search for an optimal PK model structure within a predefined search space using nlmixr2-based model fitting and penalties.

Usage

```
ga.operator(
  dat,
  param_table = NULL,
  search.space = c("ivbase", "oralbase"),
  no.cores = NULL,
  ga.control = gaControl(),
  penalty.control = penaltyControl(),
  precomputed_results_file = NULL,
  foldername = NULL,
  filename = "test",
  seed = 1234,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`ga.control`	A list of GA control parameters, generated by `gaControl()`. Includes: • npop - number of individuals (chromosomes) per generation. • niter - maximum number of generations. • pcross - crossover probability. • pmut - per-bit mutation probability. • diff_tol - significance difference threshold used for ranking. • nls - frequency (in generations) of running local exhaustive search around the best current model.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`foldername`	Character string specifying the folder name for storing intermediate results. If `NULL` (default), `tempdir()` is used for temporary storage. If specified, a cache directory is created in the current working directory.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`seed`	Integer. Random seed controlling the random sampling steps of the genetic algorithm operator for reproducible runs. Default is 1234.
`.modEnv`	Optional environment used to store run state and cached results. If NULL, a new environment is created.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to `mod.run()`.

Details

The algorithm evolves a population of binary-encoded candidate models over multiple generations using tournament selection, crossover, mutation, and local search. Candidate encodings are validated and then evaluated by fitting models and applying user-defined penalties. The best individual is carried forward to the next generation.

Value

An object of class "gaOperatorResult" containing:

- Final Selected Code: data frame with the best binary encoding.

- Final Selected Model Name: model identifier for the best encoding.
- Model Run History: data frame of fitted models and fitness values.
- Selection History: list of per-generation results.

Author(s)

Zhonghui Huang

See Also

mod.run, gaControl, penaltyControl, auto_param_table, spaceConfig, create.pop, validStringbinary, decodeBinary, parseName, rank_new, runlocal, ga.sel.tournament, ga.crossover, ga.mutation

Examples

```
# Example usage with phenotype dataset
outs <- ga.operator(
  dat = pheno_sd,
  param_table = NULL,
  search.space = "ivbase",
  ga.control = gaControl(),
  saem.control = nlmixr2est::saemControl(
    seed = 1234,
    nBurn = 200,
    nEm   = 300,
    logLik = TRUE
  )
)
print(outs)
```

ga.sel.tournament *Tournament selection*

Description

Select individuals for the next generation using tournament selection.

Usage

```
ga.sel.tournament(data.pop, npop, nbits)
```

Arguments

`data.pop`	A data.frame containing the current population. The first nbits columns must be the chromosome (typically coded as 0/1). The data frame must also contain a column named `rank`, where smaller values indicate better individuals (e.g., rank 1 is best).
`npop`	Integer. Number of individuals to select for the next generation.
`nbits`	Integer. Number of bits (genes) per chromosome; i.e., the number of columns taken from `data.pop` to form the chromosome matrix.

Value

A matrix of dimension npop x nbits containing the selected chromosomes for the next generation.

Author(s)

Zhonghui Huang

Examples

```
data.pop <- data.frame(fitness = stats::runif(10), rank = rank(stats::runif(10)))
population <- matrix(sample(0:1, 100, replace = TRUE), nrow = 10)
ga.sel.tournament(data.pop=cbind(as.data.frame(population), data.pop), npop=10, nbits=10)
```

`gaControl` *Control parameters for genetic algorithm*

Description

Creates a list of control settings for the `ga.operator()` function.

Usage

```
gaControl(
  npop = 20,
  niter = 20,
  pcross = 0.7,
  pmut = 0.1,
  diff_tol = 1,
  nls = 3
)
```

Arguments

`npop`	Integer. The number of individuals (chromosomes) in the population for each generation.
`niter`	Integer. The maximum number of generations to run the GA.
`pcross`	Numeric in $[0, 1]$. Probability of performing crossover between two selected parents.
`pmut`	Numeric in $[0, 1]$. Probability of mutating each bit in a chromosome.
`diff_tol`	A numeric value specifying the significance difference threshold. Values within this threshold are considered equal and receive the same rank. Default is 1.
`nls`	Integer. Frequency (in generations) of running local exhaustive search around the best current model.

Value

A named list containing all GA control parameters.

Author(s)

Zhonghui Huang

See Also

`ga.operator`, `rank_new`, `runlocal`

Examples

```
# Default settings
gaControl()
```

`generate_neighbors_df` *Generate neighbor models*

Description

Generates a set of neighbor models from a given current model code. The neighborhood is defined as all single-variable changes (1-bit modifications).

Usage

```
generate_neighbors_df(
  current_string,
  search.space = c("ivbase", "oralbase"),
  nsize = NULL
)
```

Arguments

`current_string`	A named numeric vector representing the current model code. Names correspond to model features (e.g. "no.cmpt", "eta.vc", "rv"), and values to their current states.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`nsize`	Integer (optional). Maximum number of neighbors to return. If NULL (default), the full neighborhood is returned. If specified, a random subset of this size is sampled.

Details

For each neighbor, both the original (pre-validation) and the validated (post-validation) codes are retained. This allows downstream functions (e.g. `detect_move()`) to distinguish between the intended primary modification and any secondary adjustments introduced by validation.

Optionally, the function can restrict the number of neighbors by random sampling (candidate list strategy).

Value

A list with two components:

original Neighbors generated by single-variable flips, before validation.

validated Neighbors after validation, representing feasible models.

Author(s)

Zhonghui Huang

Examples

```
current_string <- c(no.cmpt = 2, eta.km = 0, eta.vc = 1,
                    eta.vp = 0, eta.vp2 = 0, eta.q = 1,
                    eta.q2 = 0, mm = 0, mcorr = 1, rv = 2)
neighbors <- generate_neighbors_df(current_string, search.space = "ivbase")
head(neighbors$original)   # raw neighbors (pre-validation)
head(neighbors$validated)  # validated neighbors (post-validation)
```

`get.mod.lst` *Summarize parameter estimates and run information from an nlmixr2 fit*

Description

Extracts fixed effects, between-subject variability, residual variability, estimation precision, confidence intervals, covariance structure, shrinkage, and key runtime metrics from a fitted model produced by nlmixr2.

Usage

```
get.mod.lst(fit.s, modi)
```

Arguments

`fit.s`	A model object generated using nlmixr2.
`modi`	A numeric identifier used to label the model results, for example when multiple models are evaluated in sequence.

Details

The function checks for the presence of each element before extraction to ensure robust handling of incomplete estimation or missing covariance results.

Value

A data.frame with parameter summaries, model fit criteria (AIC, BIC, objective function value, log-likelihood, number of estimated parameters) and computation timings extracted from the fitted object.

Author(s)

Zhonghui Huang

Examples

```
pheno <- function() {
  ini({
    tcl <- log(0.008) # typical value of clearance
    tv <-  log(0.6)   # typical value of volume
    eta.cl + eta.v ~ c(1,
                       0.01, 1) ## cov(eta.cl, eta.v), var(eta.v)
    add.err <- 0.1    # residual variability
  })
  model({
    cl <- exp(tcl + eta.cl) # individual value of clearance
    v <- exp(tv + eta.v)    # individual value of volume
    ke <- cl / v            # elimination rate constant
    d/dt(A1) = - ke * A1    # model differential equation
    cp = A1 / v             # concentration in plasma
    cp ~ add(add.err)       # define error model
  })
}

# Fit the model using nlmixr2
fit <- nlmixr2est::nlmixr2(pheno, pheno_sd, est="saem", nlmixr2est::saemControl(print=0))

# Extract model results
model_results <- get.mod.lst(fit,1)
print(model_results)
```

`initialize_param` *Initialize model parameters from parameter table*

Description

Generates parameter initialization code based on a parameter table, handling both fixed and estimated parameters.

Usage

```
initialize_param(param_name, param_table)
```

Arguments

`param_name`	Character, name of the parameter to initialize (without "l" prefix)
`param_table`	Dataframe containing parameter specifications, must include: • `Name`: Character parameter names with "l" prefix (e.g., "lka" corresponds to param_name="ka") • `init`: Numeric initial values • `fixed`: Integer flag (0/1) indicating fixed status (1 = fixed)

Value

Character vector containing generated initialization code line. Format:

- Fixed parameters: `<param_name> <- fix(initial_value)`
- Estimated parameters: `l<param_name> <- initial_value`

Author(s)

Zhonghui Huang

Examples

```
# Create sample parameter table
param_table <- initialize_param_table()

# Generate initialization code
initialize_param("ka", param_table)  # Returns "ka <- fix(0.500)"
initialize_param("cl", param_table)  # Returns "lcl <- 1.200"
```

initialize_param_table

Generate initial parameter table for pharmacometric model estimation

Description

Creates a structured parameter table containing initial estimates with constraints for base parameters, inter-individual variability (ETA), residual errors (SIGMA), and correlation terms (OMEGA) to initialize nonlinear mixed-effects model fitting.

Usage

```
initialize_param_table()
```

Details

This table includes:

- Base PK parameters (absorption, clearance, volumes, etc.) in log-scale
- Michaelis-Menten kinetics parameters (vmax, km)
- Absorption parameters including zero-order, mixed-order, and transit compartment models
- Residual variability components (additive and proportional error)
- Inter-individual variability (ETA) terms with variance parameters
- Correlation parameters between ETA terms in two blocks:
  - Block 1: vmax and km parameters
  - Block 2: clearance, volumes, and inter-compartmental clearance

Parameters are organized with:

- Name: Parameter name following standard nomenclature
- init: Initial estimate for model fitting
- lb/ub: Lower/upper bounds for parameter estimation
- fixed: Flag indicating fixed parameters (1) vs estimated (0)
- Description: Plain-text explanation of parameter meaning

Value

A data.frame with 29 columns containing parameter specifications. The structure includes:

Name Parameter name (e.g., "lcl", "eta.vc", "cor.eta_cl_vc")

init Numeric initial value for parameter estimation

lb Lower bound constraint (use -Inf for unconstrained)

ub Upper bound constraint (use Inf for unconstrained)

fixed Integer flag indicating whether parameter should be fixed (1) or estimated (0)

Description Text description of parameter's biological/pharmacometric meaning

Author(s)

Zhonghui Huang

Examples

```
# Generate default parameter table
initialize_param_table()
```

`initNodeList` *Initialize node list for ACO search space*

Description

Construct the initial edge list used in model structure search based on ant colony optimization.

Usage

```
initNodeList(search.space, phi0)
```

Arguments

`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`phi0`	A non-negative numeric value. Initial pheromone value assigned to all nodes at the start of the search. Defaults to 2.

Value

A data.frame in which each row represents an edge in the ACO path-construction graph, with the following columns:

travel Integer. Travel counter associated with the edge, initialized to zero.

node.no Integer. Decision node identifier corresponding to a model feature.

node.name Character. Semantic label of the decision node.

edge.no Integer. Global edge index.

local.edge.no Integer. Index of the edge within the corresponding decision node.

edge.name Character. Semantic label of the edge (model component choice).

phi Numeric. Initial pheromone value associated with the edge.

delta_phi Numeric. Change in pheromone level, initialized to zero.

p Numeric. Initial selection probability of the edge.

Author(s)

Zhonghui Huang

Examples

```
initNodeList(search.space = "ivbase", phi0 = 1)
initNodeList(search.space = "oralbase", phi0 = 1)
```

`is_move_tabu` *Check if a move is tabu*

Description

Given a move (variable, from-value, to-value) and a tabu list, this function checks whether the move is currently forbidden by the tabu list.

Usage

```
is_move_tabu(move, tabu_list, policy = c("attribute", "move"))
```

Arguments

`move`	A list as returned by `detect_move`, containing `element`, `from`, and `to`.
`tabu_list`	Data frame of tabu elements, with columns: `elements` (variable name), `elements.value` (forbidden value), and `tabu.iteration.left` (remaining tabu tenure).
`policy`	Character scalar. Tabu restriction type: `"attribute"` (default) or `"move"`.

Value

Logical scalar: TRUE if the move is tabu, FALSE otherwise.

Author(s)

Zhonghui Huang

Examples

```
move <- list(element = "no.cmpt", from = 2, to = 3)
tabu_list <- data.frame(
  elements = c("no.cmpt", "eta.vc"),
  elements.value = c(3, 1),
  tabu.iteration.left = c(2, 1)
)
is_move_tabu(move, tabu_list)
```

mod.run *Run population pharmacokinetic model with pre-defined search space*

Description

Fits a population PK model using nlmixr2 with configurable search spaces. Supports pre-defined model structures (IV, oral) and custom configurations for advanced modeling scenarios.

Usage

```
mod.run(
  string = NULL,
  dat = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  penalty.control = penaltyControl(),
  param_table = NULL,
  nlmixr2autoinits = TRUE,
  reuse_cache = 1,
  precomputed_results_file = NULL,
  foldername = NULL,
  filename = "test",
  save_fit_rds = FALSE,
  save_csv = TRUE,
  .modEnv = NULL,
  verbose = TRUE,
  custom_config,
  ...
)
```

Arguments

`string`	Numeric vector of parameter values. The length and interpretation depends on the search.space configuration: • "ivbase": 10 values in order: (no.cmpt, eta.km, eta.vc, eta.vp, eta.vp2, eta.q, eta.q2, mm, mcorr, rv) • "oralbase": 11 values in order: (no.cmpt, eta.km, eta.vc, eta.vp, eta.vp2, eta.q, eta.q2,eta.ka, mm, mcorr, rv) • "custom": Length determined by custom_config$params in the order specified The meaning of each element name is defined in `ppkmodGen()`.
`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`search.space`	Character string specifying which search space to use. Options are "ivbase", "oralbase", or "custom". Default is "ivbase".

`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`nlmixr2autoinits`	Logical; if TRUE, use automatic initial estimates from nlmixr2. Default is TRUE.
`reuse_cache`	Integer; if 1, attempt to reuse cached results from previous runs. Default is 1.
`precomputed_results_file`	Character string; path to a CSV file containing pre-computed model results for caching
`foldername`	Character string specifying the folder name for storing model files and results. If `NULL` (default), `tempdir()` is used for temporary storage. If specified, a cache directory is created in the current working directory.
`filename`	Character string; base name for output files (without extension). Required parameter with no default.
`save_fit_rds`	Logical; if TRUE, save the fitted model object as an RDS file. Default is FALSE.
`save_csv`	Logical; if TRUE, save results to a CSV file. Default is TRUE.
`.modEnv`	Environment for storing state across multiple model runs. If NULL, a new environment will be created.
`verbose`	Logical; if TRUE, print progress messages. Default is TRUE.
`custom_config`	List; custom search space configuration for use with search.space = "custom". Must contain four elements: route, params, param_dependencies, and fixed_params. See Details and Examples.
`...`	Additional arguments passed to nlmixr2 control functions (e.g., saem.control, table.control, max_wall_time)

Details

This function implements a flexible framework for fitting population PK models with different structural configurations. It uses a configuration-driven approach where the search.space parameter determines how the string vector is interpreted and which model structure is generated.

Search Space Configurations: The function supports three types of search spaces:

ivbase Pre-defined IV bolus model with 11 parameters. Supports 1-3 compartment models with linear or Michaelis-Menten elimination.

oralbase Pre-defined oral model with 12 parameters (adds eta.ka for first-order absorption). Same features as ivbase plus absorption kinetics.

custom User-defined model structure requiring custom_config argument. Allows any combination of parameters supported by `ppkmodGen()`. Supported parameters include: no.cmpt, abs.bio, abs.type, abs.delay, eta.ka, eta.vc, eta.vp, eta.vp2, eta.q, eta.q2, mm, eta.km, eta.tlag, eta.n, eta.mtt, eta.bio, eta.D2, eta.F1, eta.Fr, mcorr, rv, and allometric_scaling. Note: eta.cl and eta.vmax are mutually exclusive and cannot be placed in the search space simultaneously; NLME models must include either eta.cl (when mm = 0) or eta.vmax (when mm = 1) to ensure at least one random effect on elimination. For advanced model parameters not covered by `nlmixr2autoinit()`, initial estimates default to 1 before any transformation. Users can provide custom initial estimates through the param_table argument.

Custom Configuration Structure: When using search.space = "custom", the custom_config argument must be provided as a list with four required elements:

route Character string: "bolus", "oral", or "mixed_iv_oral"

params Character vector of parameter names expected in string, in the exact order they appear. Length of this vector must match length of string.

param_dependencies Named list of functions where each function computes a parameter value based on other parameters. Example: eta.vmax = function(mm) if (mm == 0) 0 else 1. Use empty list if no dependencies exist.

fixed_params Named list of parameters with fixed values that are NOT in string. These parameters are automatically passed to the model generator. Use empty list if no fixed parameters exist.

Using fixed_params: The fixed_params element specifies parameter values that remain constant and do not appear in the string vector. This mechanism serves to:

- Define model structure (e.g., compartment count, absorption type)
- Fix certain parameters at specific values across all model runs
- Keep the string vector shorter and focused on variable parameters

Caching System: A two-level caching system avoids re-fitting identical models:

- In-memory cache: Results stored in .modEnv during current session
- File-based cache: Results loaded from CSV file specified by filename

To enable caching, set reuse_cache = 1 (default) and use consistent filename across runs. Pass the same .modEnv object to subsequent calls to maintain in-memory cache between model evaluations.

Value

Numeric value representing the fitness score of the fitted model

Author(s)

Zhonghui Huang

See Also

`spaceConfig` for search space configuration details. `parseParams` for parameter parsing. `ppkmodGen` for model generation. `penaltyControl` for penalty control settings.

Examples

```
# Example 1: IV model with pre-defined search space
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(0.008)
param_table$init[param_table$Name == "lvc1cmpt"] <- log(0.6)
result <- mod.run(
  string = c(1, 0, 0, 0, 0, 0, 0, 0, 0, 1),
  dat = pheno_sd,
  search.space = "ivbase",
  param_table = param_table,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=200,nEm=300)
)

# Example 2: Oral model with pre-defined search space
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(2.72)
param_table$init[param_table$Name == "lvc1cmpt"] <- log(31.5)
result <- mod.run(
  string = c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1),
  dat = theo_sd,
  search.space = "oralbase",
  param_table = param_table,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=200,nEm=300)
)

# Example 3: Simplified 1-compartment model with allometric scalling on
# Fix no.cmpt=1 and mcorr=0, vary only CL, Vc, and residual error
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(0.008 *((70/3.5)^0.75))
param_table$init[param_table$Name == "lvc1cmpt"] <- log(0.6 *((70/3.5)))
simple_config <- list(
  route = "bolus",
  params = c("eta.vc", "mcorr", "rv"),
  param_dependencies = list(),
  fixed_params = list(
    no.cmpt = 1,
    eta.cl = 1,
    allometric_scaling = 1
  )
)
dat<-pheno_sd
dat$LOGWT<-log(dat$WT/70)
result <- mod.run(
  string = c(1, 1, 1),  # Only 3 values needed
  dat = dat,
  search.space = "custom",
  custom_config = simple_config,
  param_table = param_table,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=200,nEm=300)
)
```

omega_block *Generate omega block Code for nlmixr2 model*

Description

Generates the code for the omega block matrix in nlmixr2 syntax, supporting both independent variance terms and correlated covariance structures.

Usage

```
omega_block(param_list, mcorr, eta_table)
```

Arguments

`param_list`	A character vector of parameter names requiring inter-individual variability (IIV) terms.
`mcorr`	Integer flag indicating covariance structure: • `0`: Generate independent variance terms only • `1`: Generate full block covariance structure
`eta_table`	A data frame containing eta initialization values and correlation coefficients. Must contain columns: • `Name`: Parameter names (format "eta.X" for variances, "cor.eta_X_Y" for correlations) • `init`: Initialization values for variance/covariance terms

Value

A character string containing nlmixr2 omega matrix specification code.

- When `mcorr = 0`: Returns individual variance terms in formula syntax
- When `mcorr = 1`: Returns covariance block structure in matrix syntax

Author(s)

Zhonghui Huang

Examples

```
# Example eta table structure
eta_table <- initialize_param_table()

# Generate independent terms
omega_block(c("eta.cl", "eta.vc"), mcorr = 0, eta_table)

# Generate covariance block
omega_block(c("eta.cl", "eta.vc"), mcorr = 1, eta_table)
```

`p.calculation` *Calculate selection probabilities for each node*

Description

Calculates the probability of selecting each node in an ant colony optimization search, based on pheromone levels ϕ.

Usage

```
p.calculation(nodeslst, prob_min = NULL)
```

Arguments

`nodeslst`	A data frame of nodes, including columns: **phi** Current pheromone level ϕ **node.no** Group ID for the decision step **p** Probability of selection (to be calculated)
`prob_min`	Numeric scalar. Minimum probability each node is allowed to have within its decision group. Set to NULL or 0 to disable smoothing.

Details

Within each decision group G, selection probabilities are computed from pheromone levels ϕ as:

$$p_i = \frac{\phi_i}{\sum_{j \in G} \phi_j}$$

If prob_min is enabled and any calculated probability falls below this value, the algorithm:

1. Sets all probabilities below prob_min to prob_min.
2. Redistributes the remaining probability mass proportionally among the other nodes in the same group.

This acts as a probability smoothing mechanism, preventing premature convergence by ensuring all nodes retain some chance of being explored.

Value

The updated node list with recalculated p values.

Author(s)

Zhonghui Huang

Examples

```
node.list <- initNodeList(search.space = "ivbase", phi0 = 1)
p.calculation(nodeslst = node.list, prob_min = 0.2)
```

`param.bounds` *Define Parameter Bounds for PK Models*

Description

Utility function to generate lower and upper bounds for pharmacokinetic model parameters, including fixed effects (theta), random effects variances (omega), residual error (sigma), and correlation constraints.

Usage

```
param.bounds(
  theta = list(lower = NULL, upper = NULL),
  omega = list(lower = NULL, upper = NULL),
 sigma = list(add = list(lower = 0.001, upper = Inf), prop = list(lower = 0.001, upper =
    Inf)),
  correlation = list(lower = 0.1, upper = 0.8)
)
```

Arguments

`theta` A list with optional elements:

- **lower** Named list of lower bounds for fixed effects. Defaults to -Inf for all parameters.
- **upper** Named list of upper bounds for fixed effects. Defaults to 10^9 for all parameters.

`omega` A list with optional elements:

- **lower** Named list of lower bounds for variance terms. Defaults to 10 for all parameters.
- **upper** Named list of upper bounds for variance terms. Defaults to Inf for all parameters.

`sigma` A list with two elements (each itself a list of bounds):

- **add** Lower and upper bounds for additive error component. Defaults to 0.001 and Inf.
- **prop** Lower and upper bounds for proportional error component. Defaults to 0.001 and Inf.

`correlation` A list with elements lower and upper giving the bounds for correlation terms. Defaults to 0.1 and 0.8.

Details

Default theta bounds use -Inf for lower limits and 10^9 for upper limits to avoid allowing unrealistically large fixed effect estimates while still providing flexibility during model estimation.

Value

A named list with four components:

theta List of parameter-specific lower and upper bounds for fixed effects.

omega List of lower and upper bounds for variance terms.

sigma List with additive (add) and proportional (prop) error bounds.

correlation List with lower and upper correlation bounds.

Author(s)

Zhonghui Huang

Examples

```
# Use all default bounds
 param.bounds()

# Customize only omega lower bounds
param.bounds(omega = list(lower = list(cl = 5, vc = 2)))

# Adjust sigma proportional error bounds
param.bounds(
  sigma = list(
    add = list(lower = 0.001, upper = 1),
    prop = list(lower = 0.01, upper = 0.05)
  )
)
```

`parseName` *Parse model coding vector to model name*

Description

Converts an ordinal model-coding vector into a single pharmacokinetic model name string, using a search space configuration.

Usage

```
parseName(modcode, search.space = NULL, custom_config = NULL)
```

Arguments

`modcode`	Numeric vector of model-coding flags/values in the order defined by the search space configuration.
`search.space`	Character string specifying which search space to use. Options are "ivbase", "oralbase", or "custom". Default is "ivbase".

`custom_config`	Optional named list defining a custom parameter structure. If provided, the parameter names are taken from the names of this list. If NULL, a default parameter structure is used based on the selected search space.

Details

The function selects a configuration (either a custom configuration when search.space is "custom", or the predefined configuration from spaceConfig(search.space) otherwise). It then decodes modcode with parseParams() and assembles an underscore-separated model name.

The name is built from these blocks in order: prefix, compartments, optional absorption, ETA block, elimination, correlation, residual error, and optional allometric scaling.

Key rules:

- The prefix is taken from config$prefix when available; otherwise from config$route; otherwise from search.space.
- The absorption block is included when any absorption option is present. If abs.type, abs.delay, and abs.bio are all missing/NA, the absorption block is included only for oral or mixed routes using FO_abs; otherwise it is omitted.
- The ETA block includes all eta.* terms equal to 1 (NA values are ignored), and also forces Vmax when mm equals 1; otherwise it forces CL.
- Elimination uses FO_elim when mm is 0 or NA, and MM_elim when mm is 1. Correlation uses uncorrelated when mcorr is 0 or NA, and correlated when mcorr is 1. Residual error uses add, prop, or combined.
- Allometric scaling is omitted when allometric_scaling is 0 or NA; otherwise it appends "asWT", "asBMI", or "asFFM".

Value

A character string representing the constructed model name.

Author(s)

Zhonghui Huang

See Also

`spaceConfig()`, `parseParams()`

Examples

```
# Example 1: Parse IV base model name
parseName(c(1, 0, 0, 0, 0, 0, 0, 0, 0, 1), "ivbase")

# Example 2: Parse oral base model name
parseName(c(2, 1, 1, 0, 0, 1, 1, 1, 0, 1, 3), "oralbase")

# Example 3: Parse custom configuration model name
custom_config <- list(
  prefix = "custom",
  route  = "oral",
  params = c("no.cmpt", "eta.cl", "eta.vc", "mm", "mcorr", "rv"),
  param_dependencies = list(),
  fixed_params = list()
)
parseName(c(2, 1, 0, 0, 1, 2), search.space = "custom",
custom_config = custom_config)
```

`parseParams` *Parse string vector to model parameters*

Description

Converts a numeric vector of parameter values into a named list of model parameters based on the search space configuration.

Usage

```
parseParams(string, config)
```

Arguments

`string`	Numeric vector containing parameter values in the order specified by the search space configuration
`config`	List object returned by `spaceConfig()`, containing parameter definitions and dependencies

Details

This function performs three main operations:

1. Maps the input vector to named parameters
2. Computes dependent parameter values using defined functions
3. Adds fixed parameters and route information

Value

A named list containing:

- All parameters specified in config$params with their values
- Computed dependent parameters based on param_dependencies
- Fixed parameters from `fixed_params`
- Administration route from config$route

Author(s)

Zhonghui Huang

See Also

`spaceConfig()`, `mod.run()`

Examples

```
# Example 1: Parse IV base model parameters
config_iv <- spaceConfig("ivbase")
parseParams(c(2, 1, 1, 0, 0, 1, 1, 0, 1, 1), config_iv)

# Example 2: Parse oral base model parameters
config_oral <- spaceConfig("oralbase")
parseParams(c(2, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1), config_oral)

# Example 3: Parse custom configuration parameters
custom_config <- list(route = "oral", params = c("no.cmpt", "eta.cl", "eta.vc"),
                      param_dependencies = list(), fixed_params = list(mm = 0))
parseParams(c(1, 1, 1), custom_config)
```

`penaltyControl` *Configure penalty settings for model evaluation*

Description

Defines rules governing penalty assignment during model adequacy evaluation.

Usage

```
penaltyControl(
  penalty.value = 10000,
 step.penalties = list(rse = c(10, 10000), shrinkage = c(10, 10000), bsv = c(10, 10000),
   sigma = list(add = c(10, 10000), prop = c(10, 10000)), correlation = c(10, 10000)),
  bounds = param.bounds(),
  thresholds = list(),
  penalty.terms = c("total")
)
```

Arguments

`penalty.value`	Numeric. Constant penalty assigned to binary violations and bound constraints.
`step.penalties`	A named list defining penalty magnitudes used in step-wise procedures. Each element must contain a numeric vector of length two representing penalty levels for moderate and critical deviations.
`bounds`	A list specifying lower and upper parameter limits, as returned by param.bounds(). The structure can include limits for theta, omega, sigma, and correlation terms.
`thresholds`	A named list describing evaluation rules for RSE and shrinkage. Each component must include a field named method, with value binary or step, together with the corresponding limit definition:

- If method = binary: a single cutoff value stored in threshold
- If method = step: two deviation boundaries stored in step.levels

`penalty.terms` Character vector specifying which components are considered when penalties are reported. Recognized entries include: rse, shrinkage, theta, omega, sigma, correlation, covariance, and total. If total is included, penalties are aggregated across all components and any other entries are ignored.

Details

Penalization may be triggered by exceeding predefined parameter bounds (fixed-effect and variance-covariance elements) or by surpassing thresholds for relative standard error (RSE) or shrinkage criteria. Binary and step-wise penalty procedures are supported.

Value

A list containing the full penalty configuration for use in fitness().

Author(s)

Zhonghui Huang

See Also

`param.bounds()`, `fitness()`.

Examples

```
# Default configuration
penaltyControl()

# Custom bounds for selected fixed-effect parameters
penaltyControl(bounds = param.bounds(
  theta = list(lower = list(cl = 0.01, vc = 0.01))
))

# Binary penalty method for RSE
penaltyControl(thresholds = list(
  rse = list(method = "binary", threshold = 40)
))
```

`perturb_2bit` *Apply 2-bit perturbation to escape local optimum*

Description

Randomly flips two parameters ("2-bit change") in the current model string to generate a perturbed candidate.

Usage

```
perturb_2bit(prev_string, search.space, max.try = 1000)
```

Arguments

`prev_string`	A named numeric vector representing the current model.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`max.try`	Maximum number of attempts to generate a valid perturbed model.

Details

The function returns both:

- `original_neighbor`: the raw 2-bit flip before validation
- `validated_neighbor`: the corrected version after validation

This allows downstream functions (e.g. `detect_move()`) to identify which parameters were intentionally changed (primary moves), while still using a valid model code for evaluation.

Value

A `list` with two named numeric vectors:

`original_neighbor`	raw 2-bit flip (may be invalid)
`validated_neighbor`	validated and usable model code

Author(s)

Zhonghui Huang

Examples

```
prev <- c(no.cmpt = 2, eta.km = 0, eta.vc = 1,
          eta.vp = 0, eta.vp2 = 0, eta.q = 1,
          eta.q2 = 0, mm = 0, mcorr = 1, rv = 2)
perturb <- perturb_2bit(prev, search.space = "ivbase")
perturb$original_neighbor   # original 2-bit flip
perturb$validated_neighbor  # validated model
```

phi.calculate *Update pheromone levels for each decision node*

Description

Compute pheromone increments (`delta_phi`) for each node in the ant colony optimization search tree and update the global pheromone levels (`phi`) based on the ants' paths in the current round.

Usage

```
phi.calculate(
  r,
  search.space = "ivbase",
  fitness_history = NULL,
  nodeslst.hist = NULL,
  Q = 1,
  alpha = 1,
  rho = 0.5,
  diff_tol = 1,
  phi0 = 2,
  phi_min = 1,
  phi_max = Inf
)
```

Arguments

`r`	Integer. Current optimization round.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`fitness_history`	Data frame. History of ants' fitness values and decision variable selections across rounds.
`nodeslst.hist`	Data frame. History of node-level pheromone values across previous rounds.
`Q`	A positive numeric value. Pheromone scaling constant controlling the amount of pheromone deposited by high-quality solutions during each iteration. Defaults to 1.
`alpha`	A non-negative numeric value. Exponent controlling the influence of pheromone values on the probability of selecting a component during solution construction. Defaults to 1.
`rho`	Numeric in (0, 1). Pheromone evaporation rate. Higher values increase evaporation, encouraging exploration. Defaults to 0.5.
`diff_tol`	Numeric. Tolerance threshold controlling when differences in fitness values are treated as meaningful during pheromone updates. Defaults to 1.
`phi0`	A non-negative numeric value. Initial pheromone value assigned to all nodes at the start of the search. Defaults to 2.

`phi_min`	A non-negative numeric value. Lower bound for pheromone values, preventing premature convergence. Defaults to 1.
`phi_max`	A non-negative numeric value. Upper bound for pheromone values, limiting excessive reinforcement. Defaults to Inf.

Details

The update proceeds as follows:

- Initialize the node list for the given search space with $phi = 0$.
- Subset the ants from the current round in `fitness_history`.
- Compute rank-based weights so better-performing ants contribute more:

$$\Delta\phi \propto 1/\mathrm{rank}^{\alpha}.$$

- Extract the decision columns and attach the computed weights to form a working table of ant paths and contributions.
- Map local decision indices to global node numbers using `node.no` and `local.edge.no` from the node list.
- For each node, sum contributions from ants that selected the node to obtain $\Delta\phi$, then update pheromone with evaporation:

$$\phi_{\mathrm{new}} = (1-\rho)\,\phi_{\mathrm{prev}} + \Delta\phi.$$

- Clamp updated ϕ to be between `phi_min` and `phi_max`.

Value

A data frame (node list) with updated `phi` and `delta_phi` for each node.

Author(s)

Zhonghui Huang

See Also

initNodeList, rank_new

Examples

```
# Define search space
search.space <- "ivbase"
# Example fitness_history from round 1
fitness_history <- data.frame(
  round   = rep(1, 8),
  mod.no  = 1:8,
  no.cmpt = c(1, 1, 2, 2, 3, 3, 2, 2),
  eta.km  = c(0, 0, 0, 0, 0, 0, 0, 0),
  eta.vc  = c(0, 0, 0, 0, 0, 0, 1, 1),
  eta.vp  = c(0, 0, 0, 0, 0, 0, 0, 1),
  eta.vp2 = c(0, 0, 0, 0, 0, 0, 0, 0),
  eta.q   = c(0, 0, 0, 0, 0, 0, 0, 0),
  eta.q2  = c(0, 0, 0, 0, 0, 0, 0, 0),
  mm      = c(0, 0, 0, 0, 0, 0, 1, 0),
  mcorr   = c(0, 0, 0, 0, 0, 0, 0, 0),
  rv      = c(1, 2, 1, 2, 1, 2, 1, 1),
  fitness = c(1243.874, 1200.762, 31249.876, 31202.200,
              51259.286, 51204.839, 61032.572, 41031.825),
  allrank = c(2, 1, 4, 3, 7, 6, 8, 5)
)

# Example node list history
nodeslst.hist <- initNodeList(
  search.space = search.space,
  phi0 = 2
)

 phi.calculate(
  r = 1,
  search.space = search.space,
  fitness_history = fitness_history,
  nodeslst.hist = nodeslst.hist
)
```

ppkmodGen — *Generate a Pharmacokinetic (PK) Model for nlmixr2*

Description

Constructs a PK model based on specified parameters, absorption characteristics, variability components, and residual error models. The model is generated as a text file compatible with nlmixr syntax. The function handles various absorption types, multi-compartment models, Michaelis-Menten kinetics, and different residual variability structures.

Usage

```
ppkmodGen(
  modi = 1,
  route = "bolus",
  no.cmpt = 1,
  abs.bio = 0,
  abs.type = 1,
  abs.delay = 0,
  eta.ka = 0,
  eta.cl = 0,
  eta.vc = 0,
  eta.vp = 0,
  eta.vp2 = 0,
  eta.q = 0,
  eta.q2 = 0,
  mm = 0,
  eta.vmax = 0,
  eta.km = 0,
  eta.tlag = 0,
  eta.n = 0,
  eta.mtt = 0,
  eta.bio = 0,
  eta.D2 = 0,
  eta.F1 = 0,
  eta.Fr = 0,
  mcorr = 0,
  rv = 1,
  allometric_scaling = 0,
  param_table = NULL,
  return.func = FALSE,
  out.dir = NULL,
  verbose = TRUE
)
```

Arguments

`modi`	Model identification number (default: 1). Used for generating unique model filenames.
`route`	Administration route. Valid options: "bolus", "oral", "mixed_iv_oral" (default: "bolus").
`no.cmpt`	Number of compartments in the model (1, 2, or 3) (default: 1).
`abs.bio`	Bioavailability flag (0 = no bioavailability, 1 = with bioavailability) (default: 0).
`abs.type`	Absorption type (1 = first-order, 2 = zero-order, 3 = sequential first-order and zero-order absorption, 4 = dual first-order and zero-order absorption) (default: 1).
`abs.delay`	Absorption delay type (0 = none, 1 = lag time, 2 = transit compartments) (default: 0).
`eta.ka`	Variability flag for absorption rate (ka) (0 = no variability, 1 = include variability).
`eta.cl`	Variability flag for clearance (CL) (0 = no variability, 1 = include variability).
`eta.vc`	Variability flag for central volume (Vc) (0 = no variability, 1 = include variability).
`eta.vp`	Variability flag for peripheral volume (Vp) in multi-compartment models.
`eta.vp2`	Variability flag for second peripheral volume (Vp2) in 3-compartment models.
`eta.q`	Variability flag for intercompartmental clearance (Q) in multi-compartment models.
`eta.q2`	Variability flag for second intercompartmental clearance (Q2) in 3-compartment models.

`mm`	Michaelis-Menten kinetics flag (0 = linear kinetics, 1 = Michaelis-Menten kinetics).
`eta.vmax`	Variability flag for Vmax when using Michaelis-Menten kinetics.
`eta.km`	Variability flag for Km when using Michaelis-Menten kinetics.
`eta.tlag`	Variability flag for lag time (tlag) when abs.delay=1.
`eta.n`	Variability flag for number of transit compartments when abs.delay=2.
`eta.mtt`	Variability flag for mean transit time when abs.delay=2.
`eta.bio`	Variability flag for bioavailability when abs.delay=2.
`eta.D2`	Variability flag for zero-order duration (D2) when abs.type=2 or 3.
`eta.F1`	Variability flag for bioavailability fraction (F1) when abs.bio=1.
`eta.Fr`	Variability flag for absorption fraction (Fr) when abs.type=4.
`mcorr`	Correlation flag for omega blocks (0 = no correlation, 1 = include correlations).
`rv`	Residual variability type (1 = additive, 2 = proportional, 3 = combined, 4 = log-normal).
`allometric_scaling`	Allometric scaling type (0 = none, 1 = weight, 2 = BMI, 3 = FFM).
`param_table`	Data frame containing parameter initial values and variability components. Should contain columns: Name (parameter name), init (initial value), eta (TRUE/FALSE for variability inclusion), cov (covariate relationships).
`return.func`	Logical, whether to return a compiled function (default `FALSE` returns model code as text).
`out.dir`	Directory where model files and results are written. Defaults to the current working directory when not provided.
`verbose`	Logical; if `TRUE`, progress messages are printed.

Value

Generates a text file ('modX.txt' where X = modi) containing the nlmixr-compatible model code. The file is written to the current working directory. No explicit return value. If `return.func = TRUE`, returns a compiled model function object.

Author(s)

Zhonghui Huang

Examples

```
withr::with_dir(tempdir(), {
#' # Create a 1-compartment oral model with first-order absorption
 ppkmodGen( no.cmpt = 1, abs.type = 1,return.func = TRUE,param_table = initialize_param_table())
})
```

`print.acoOperatorResult` *Print method for ACO operator results*

Description

Print ACO operator results.

Usage

```
## S3 method for class 'acoOperatorResult'
print(x, ...)
```

Arguments

`x`	An "acoOperatorResult" object.
`...`	Additional arguments (currently ignored).

Value

Invisibly returns x.

Author(s)

Zhonghui Huang

See Also

`aco.operator`

`print.gaOperatorResult` *Print method for gaOperatorResult objects*

Description

Custom print method for results returned by the GA operator. Displays only:

- Final selected model code
- Final selected model name

Usage

```
## S3 method for class 'gaOperatorResult'
print(x, ...)
```

Arguments

`x`	An object containing GA operator output (class gaOperatorResult).
`...`	Additional arguments (currently unused).

Value

Invisibly returns the input object.

`print.sfOperatorResult` *Print method for sfOperatorResult objects*

Description

Defines a custom print method for objects of class 'sfOperatorResult'.

Usage

```
## S3 method for class 'sfOperatorResult'
print(x, ...)
```

Arguments

`x`	An object of class 'sfOperatorResult'.
`...`	Further arguments passed to or from other methods (currently unused).

Value

Invisibly returns x.

`print.tabuOperatorResult` *Print method for tabu operator results*

Description

Print tabu operator results.

Usage

```
## S3 method for class 'tabuOperatorResult'
print(x, ...)
```

Arguments

`x`	A "tabuOperatorResult" object.
`...`	Additional arguments (currently ignored).

Value

Invisibly returns x.

See Also

tabu.operator

`rank_new` *Ranking with significance difference threshold*

Description

Performs a custom ranking of a numeric vector,and ajusts the ranks of values that differ by less than a specified threshold, ensuring they receive the same rank.

Usage

```
rank_new(x1, diff_tol)
```

Arguments

`x1`	A numeric vector to be ranked.
`diff_tol`	A numeric value specifying the significance difference threshold. Values within this threshold are considered equal and receive the same rank.

Value

A numeric vector representing the adjusted ranks of the input values.

Author(s)

Zhonghui Huang

Examples

```
x1 <- c(10, 20, 20.5, 30)
diff_tol <- 1
ranked_list <- rank_new(x1, diff_tol)
print(ranked_list)
```

runlocal *Perform 1-bit local search*

Description

Runs a 1-bit neighbourhood local search on a binary-coded model string and returns a data frame of candidate models with their computed fitness (and ranks).

Usage

```
runlocal(
  dat,
  param_table = NULL,
  search.space = c("ivbase", "oralbase"),
  no.cores = NULL,
  start.string = NULL,
  diff_tol = 1,
  penalty.control = penaltyControl(),
  precomputed_results_file = NULL,
  foldername = NULL,
  filename = "test",
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`start.string`	Optional numeric/integer vector of 0 or 1 values giving the starting binary code.
`diff_tol`	A numeric value specifying the significance difference threshold. Values within this threshold are considered equal and receive the same rank. Default is 1.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.

`foldername`	Character string specifying the folder name for storing intermediate results. If `NULL` (default), `tempdir()` is used for temporary storage. If specified, a cache directory is created in the current working directory.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`.modEnv`	Optional environment used to persist state across calls (e.g., cached parameter tables and precomputed results). When `NULL`, a new environment is created.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to `mod.run()`.

Details

For each position in the starting binary code, `runlocal()` constructs a candidate by flipping that single bit (a 1-bit flip proposal). Some model components are encoded by linked two-bit schemes (e.g., "no.cmpt1"/"no.cmpt2" and "rv1"/"rv2"); when a proposal targets the second bit of a linked pair, a feasibility rule is applied to maintain a valid encoding.

Each candidate is then canonicalised/validated using `validStringbinary` before evaluation. Fitness is obtained by calling `mod.run` for each candidate and results are ranked using `rank_new`.

If ".modEnv" is supplied and contains the GA iteration counter ".modEnv$r", local search does not advance this counter; implementations may decrement ".modEnv$r" (with a lower bound of 1) so that local search does not consume a GA "round".

Value

A data frame where each row corresponds to a unique candidate model. Columns include the binary encoding (one column per bit), the computed "fitness", and the resulting "rank".

Author(s)

Zhonghui Huang

See Also

`mod.run`, `auto_param_table`, `validStringbinary`, `penaltyControl`, `rank_new`

Examples

```
  dat <- pheno_sd
# Example best model binary code
  current_code <- c(1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0)
  param_table <- initialize_param_table()
  param_table$init[param_table$Name == "lcl"] <- log(0.008)
  param_table$init[param_table$Name == "lvc"] <- log(0.6)
# Run local search
  result_local <- runlocal(
  dat                         = dat,
  search.space                = "ivbase",
  start.string                = current_code,
  filename                    = "local_search_test",
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
print(result_local)
```

run_model_in_subprocess

Run an nlmixr2 model in an isolated subprocess

Description

Executes an nlmixr2 model fitting procedure in a separate background R session using the **processx** backend. Running the model in an isolated subprocess prevents the main R session from crashing and allows monitoring errors, wall-time limits, and controlled output.

Usage

```
run_model_in_subprocess(
  modi,
  dat,
  f,
  saem.control = NULL,
  table.control = NULL,
  max_errors = 100,
  max_wall_time = 86400,
  temp_path = NULL,
  cleanup = TRUE,
  verbose = TRUE
)
```

Arguments

`modi`	Integer. A model index used to generate unique temporary filenames.
`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format for model fitting.
`f`	An nlmixr2 model function (e.g., generated by `ppkmodGen(..., return.func = TRUE)`).
`saem.control`	A `saemControl()` object providing estimation settings.
`table.control`	A `tableControl()` object controlling table output behavior.
`max_errors`	Integer. Maximum number of detected error messages before forcibly terminating the subprocess. Default is 100.
`max_wall_time`	Numeric (seconds). Maximum allowed real (wall-clock) time for the subprocess before termination. Default is 86400 (24 hours).
`temp_path`	Character. Directory where temporary files will be written. If NULL (default), the system temporary directory `tempdir()` is used. If a non-NULL path is supplied but does not exist, the function aborts with an error.

`cleanup`	Logical. Whether to delete all temporary files upon completion. Default is TRUE. If FALSE, generated temporary files are preserved for debugging/troubleshooting.
`verbose`	Logical. If TRUE, progress and diagnostic messages are printed during subprocess monitoring. Default is TRUE.

Details

The model fitting is executed in an isolated background R process (via **processx**) to prevent the main R session from crashing due to instabilities in long-running nlmixr2/SAEM estimation routines or poorly specified models. Output and error streams are monitored in real time, and the subprocess is automatically terminated if either the error count (`max_errors`) or the wall-time limit (`max_wall_time`) is exceeded.

Temporary files used to pass data and retrieve results are stored only in the session-specific temporary directory (`tempdir()`) and are removed upon completion, ensuring that no files are created in or left behind in the user's working directory.

Value

A list with:

fit.s The fitted nlmixr2 object, or NULL if the subprocess failed.

loadError Logical indicating whether an error occurred (including timeout or crash).

Author(s)

Zhonghui Huang

Examples

```
# Example: run a simple nlmixr2 model
pheno <- function() {
  ini({
    tcl <- log(0.008)     # typical clearance
    tv  <- log(0.6)       # typical volume
    eta.cl + eta.v ~ c(1,
                       0.01, 1)  # interindividual variability
    add.err <- 0.1        # residual variability
  })

  model({
    cl <- exp(tcl + eta.cl)
    v  <- exp(tv  + eta.v)
    ke <- cl / v
    d/dt(A1) = -ke * A1
    cp = A1 / v
    cp ~ add(add.err)
  })
}
 run_model_in_subprocess(
    modi = 1,
    dat = pheno_sd,
    f = pheno,
    saem.control = nlmixr2est::saemControl(
      seed = 1234,
      nBurn = 100,
      nEm = 100,
      logLik = TRUE
    )
  )
```

`sf.operator` *Stepwise model building operator for model selection*

Description

Implements automated stepwise model selection for structural and statistical components of nonlinear mixed-effects models, evaluating the number of compartments, elimination type, inter-individual variability, correlation structures, and residual error models.

Usage

```
sf.operator(
  dat,
  start.mod = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  param_table = NULL,
  steps = 123567,
  dynamic_fitness = TRUE,
  penalty.control = penaltyControl(),
  precomputed_results_file = NULL,
  foldername = NULL,
  filename = "test",
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.

`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`steps`	Numeric or character vector defining the sequence of steps to be executed. Each digit corresponds to a specific step: **1** Number of compartments **2** Elimination type **3** IIV on Km **4** IIV on Ka **5** Forward selection of structural IIV **6** Correlation between random effects **7** Residual error model
`dynamic_fitness`	Logical; if `TRUE`, the set of penalty terms may change dynamically across steps.
`penalty.control`	An object created by `penaltyControl()` defining penalty terms used in the fitness calculation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`foldername`	Character string specifying the name of the folder to be created in the current working directory to store intermediate results. If NULL, a name is generated automatically.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`.modEnv`	Optional environment used internally to store model indices, cached parameter tables, and results across steps.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to `mod.run()`.

Details

The stepwise procedure iterates over the specified steps in order. At each step, only a single component of the model is modified, while all other structural and statistical elements remain unchanged. Model comparison is based on a scalar fitness criterion returned by the estimation routine.

The order and inclusion of steps are controlled by the user via a numeric step code sequence. Steps that are not applicable to the current model configuration may be skipped automatically.

The final best model is defined as the model with the minimum fitness value in the last completed estimation round.

Value

An object of class `"sfOperatorResult"` with the following elements:

- `"Final Best Code"`: Named integer vector of the selected model code.
- `"Final Best Model Name"`: Character string identifying the best model.

- `"Stepwise Best Models"`: Data frame summarizing the best model selected at each executed step.
- `"Stepwise History"`: Named list containing full results for each step using descriptive step names.
- `"Model Run History"`: Data frame containing all model runs performed during the procedure.

Author(s)

Zhonghui Huang

See Also

`step_compartments`, `step_elimination`, `step_iiv_km`, `step_iiv_f`, `step_correlation`, `step_rv`

`auto_param_table`, `base_model`, `penaltyControl`, `mod.run`, `ppkmodGen`, `step_compartments`, `step_elimination`, `step_iiv_km`, `step_iiv_ka`, `step_iiv_f`, `step_correlation`, `step_rv`

Examples

```
out<-sf.operator(
  dat = pheno_sd,
  steps = 1234,
  search.space = "ivbase",
  saem.control = nlmixr2est::saemControl(
    seed = 1234,
    nBurn = 200,
    nEm   = 300,
    logLik = TRUE
  )
)
print(out)
```

`spaceConfig` *Get search space configuration*

Description

Retrieve the configuration for a specified search space.

Usage

```
spaceConfig(search.space = c("ivbase", "oralbase"))
```

Arguments

`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".

Details

Pre-defined search spaces:

- "ivbase": IV bolus model, 11 parameters, supports 1 to 3 compartments.
- "oralbase": Oral model, 12 parameters (adds eta.ka), supports 1 to 3 compartments.

For "ivbase" and "oralbase", param_dependencies handle the relationship between Michaelis-Menten elimination (mm) and the associated variability parameters (eta.vmax, eta.cl).

Value

A list with four elements:

- route: Administration route ("bolus", "oral", or NULL).
- params: Character vector of parameter names expected in the string vector.
- param_dependencies: Named list of functions that compute dependent parameters.
- fixed_params: Named list of fixed parameter values.

Author(s)

Zhonghui Huang

See Also

mod.run for the main function that uses these configurations. parseParams for parameter parsing using configurations.

Examples

```
# Get IV base configuration
config <- spaceConfig("ivbase")
config$params

# Get oral base configuration
config <- spaceConfig("oralbase")
config$params
```

`step_compartments` *Screen number of compartments*

Description

Runs candidate models with one, two, and three compartments by modifying only the compartment setting in the current model code.

Usage

```
step_compartments(
  dat,
  start.mod = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  param_table = NULL,
  penalty.control = NULL,
  precomputed_results_file = NULL,
  filename = "test",
  foldername = NULL,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`foldername`	Character string specifying the folder name for storing intermediate results. If `NULL` (default), `tempdir()` is used for temporary storage. If specified, a cache directory is created in the current working directory.
`.modEnv`	Internal environment used to store model indices and cached results across steps.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to mod.run. These may include custom_base, which is used to initialize the baseline model when no best_code is present in start.mod.

Details

Three candidate models are created by modifying only the number of compartments in the starting model code. The candidate codes are evaluated sequentially, and a results table containing model names, model codes, Fitness values, and information criteria is returned for logging and decision making.

Value

A list with the following elements:

- results_table: a data frame with one row per candidate model, including model description and fit statistics
- best_code: named integer vector corresponding to the best candidate
- best_row: one-row data frame containing the best candidate summary

Author(s)

Zhonghui Huang

See Also

`mod.run`, `base_model`, `penaltyControl`

Examples

```
dat <- pheno_sd
string <- c(1, 0, 0, 0, 0, 0, 0, 0, 0, 1)
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(0.008)
param_table$init[param_table$Name == "lvc"] <- log(0.6)
penalty.control = penaltyControl()
penalty.control$penalty.terms = c("rse", "theta", "covariance")
step_compartments(
  dat = dat,
  search.space = "ivbase",
  param_table = param_table,
  filename = "step_cmpt_test",
  penalty.control = penalty.control,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
```

`step_correlation` *Evaluate inclusion of ETA correlation structure*

Description

Evaluates whether correlation between inter-individual random effects (ETA correlation) should be included in the model.

Usage

```
step_correlation(
  dat,
  start.mod = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  param_table = NULL,
  penalty.control = NULL,
  precomputed_results_file = NULL,
  filename = "test",
  foldername = NULL,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`foldername`	Character string specifying the name of the folder to be created in the current working directory to store intermediate results. If NULL, a name is generated automatically.
`.modEnv`	Optional environment used to store model indices and cached results across steps.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to the model estimation function.

Details

Two candidate models are constructed by toggling the correlation setting of inter-individual random effects in the model code. Model selection is based on comparison of Fitness values returned during estimation.

Value

A list with the following elements:

- results_table: A data frame summarizing the evaluated models,
- best_code: A named integer vector corresponding to the selected model code,
- best_row: A one-row data frame containing the summary of the selected model.

Author(s)

Zhonghui Huang

See Also

`mod.run`, `base_model`, `penaltyControl`

Examples

```
dat <- pheno_sd
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(0.008)
param_table$init[param_table$Name == "lvc"] <- log(0.6)
penalty.control <- penaltyControl()
penalty.control$penalty.terms <-
  c("rse", "theta", "covariance", "shrinkage", "omega")
start.mod <- base_model("ivbase")
start.mod["eta.vc"] <- 1L
step_correlation(
  dat = dat,
  start.mod = start.mod,
  search.space = "ivbase",
  param_table = param_table,
  filename = "step_mcorr_test",
  penalty.control = penalty.control,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
```

`step_elimination` *Screen elimination type (linear vs Michaelis-Menten)*

Description

Runs linear and Michaelis-Menten elimination candidates by modifying only the elimination setting in the current model code.

Usage

```
step_elimination(
  dat,
  start.mod = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  param_table = NULL,
  penalty.control = NULL,
  precomputed_results_file = NULL,
  filename = "test",
  foldername = NULL,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`foldername`	Character string specifying the name of the folder to be created in the current working directory to store intermediate results. If NULL, a name is generated automatically.
`.modEnv`	Optional internal environment used to store model indices and cached results across model-selection steps.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to `mod.run()`.

Details

When mm = 0, any inter-individual variability term for Km (eta.km) present in the model code is automatically set to zero.

Value

A list with the following elements:

- results_table: a data.frame with one row per candidate model, including model description, Fitness, AIC, BIC, and OFV.
- best_code: named integer vector corresponding to the best candidate's model code.
- best_row: one-row data.frame summarizing the best candidate.

Author(s)

Zhonghui Huang

See Also

`mod.run`, `base_model`, `penaltyControl`

Examples

```
dat <- pheno_sd
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(0.008)
param_table$init[param_table$Name == "lvc"] <- log(0.6)
penalty.control = penaltyControl()
penalty.control$penalty.terms = c("rse", "theta", "covariance")
# Initialize start.mod with a base model
 start.mod <- base_model("ivbase")
step_elimination(
  dat = dat,
  start.mod = start.mod,
  search.space = "ivbase",
  param_table = param_table,
  filename = "step_elim_test",
  penalty.control = penalty.control,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
```

`step_iiv_f` *Forward selection of IIV on structural parameters*

Description

Implements a forward selection procedure to assess the inclusion of inter-individual variability on structural pharmacokinetic parameters.

Usage

```
step_iiv_f(
  dat,
  start.mod = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  param_table = NULL,
  penalty.control = NULL,
  precomputed_results_file = NULL,
  filename = "test",
  foldername = NULL,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`foldername`	Character string specifying the name of the folder to be created in the current working directory to store intermediate results. If NULL, a name is generated automatically.
`.modEnv`	Optional environment for storing intermediate results across model runs.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to the model estimation function.

Details

The procedure begins with an initial model and proceeds iteratively. At each step, candidate models are generated by adding exactly one additional IIV (random-effect) term while keeping all other aspects of the model unchanged. If any candidate improves the chosen fitness criterion, the best-improving candidate becomes the new reference model for the next iteration. The algorithm stops when no further improvement is achieved. The set of parameters eligible for IIV depends on the number of compartments:

- One-compartment models: clearance and central volume
- Two-compartment models: clearance, central volume, peripheral volume, and inter-compartmental clearance
- Three-compartment models: clearance, central volume, peripheral volumes, and inter-compartmental clearances

Value

A list with three elements:

- results_table: A data frame summarizing all models evaluated during the forward selection process.
- best_code: A named integer vector corresponding to the selected model.
- best_row: A one-row data frame containing the results of the selected model.

Author(s)

Zhonghui Huang

See Also

`mod.run`, `base_model`, `penaltyControl`

Examples

```
dat <- Bolus_2CPT[Bolus_2CPT$SD==1,]
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(4)
param_table$init[param_table$Name == "lvc2cmpt"] <- log(70)
param_table$init[param_table$Name == "lvp2cmpt"] <- log(40)
param_table$init[param_table$Name == "lq2cmpt"] <- log(4)
penalty.control <- penaltyControl()
penalty.control$penalty.terms <-
  c("rse", "theta", "covariance", "shrinkage", "omega")
start.mod <- base_model("ivbase")
start.mod["no.cmpt"] <- 2L
step_iiv_f(
  dat = dat,
  start.mod = start.mod,
  search.space = "ivbase",
  param_table = param_table,
  filename = "step_eta_test",
  penalty.control = penalty.control,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
```

step_iiv_ka *Evaluate inter-individual variability on Ka*

Description

Runs candidate models with and without IIV on K_a by modifying only the corresponding random-effect setting in the current model code.

Usage

```
step_iiv_ka(
  dat,
  start.mod = NULL,
  search.space = "oralbase",
  no.cores = NULL,
  param_table = NULL,
  penalty.control = NULL,
  precomputed_results_file = NULL,
  filename = "test",
  foldername = NULL,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "oralbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`foldername`	Character string specifying the name of the folder to be created in the current working directory to store intermediate results. If NULL, a temporary path is used via `tempdir()`.

`.modEnv`	An optional environment used to store intermediate results across model runs.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments forwarded to `mod.run()`.

Details

This step is executed only when the search space is "oralbase" and the starting model code does not already include inter-individual variability on K_a. If these conditions are not met, no model comparison is performed.

Value

A list with the following elements:

- results_table: A data.frame summarizing the evaluated models.
- best_code: A named integer vector representing the selected model code.
- best_row: A one-row data.frame corresponding to the selected model.

Author(s)

Zhonghui Huang

See Also

`mod.run`, `base_model`, `penaltyControl`

Examples

```
dat <- theo_sd
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(2)
param_table$init[param_table$Name == "lvc"] <- log(30)
penalty.control <- penaltyControl()
penalty.control$penalty.terms <-
  c("rse", "theta", "covariance", "shrinkage", "omega")
start.mod <- base_model("oralbase")
step_iiv_ka(
  dat = dat,
  start.mod = start.mod,
  search.space = "oralbase",
  param_table = param_table,
  filename = "step_etaka_test",
  penalty.control = penalty.control,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
```

step_iiv_km *Evaluate inter-individual variability on Km*

Description

Runs candidate models with and without IIV on K_m by modifying only the corresponding random-effect setting in the current model code.

Usage

```
step_iiv_km(
  dat,
  start.mod = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  param_table = NULL,
  penalty.control = NULL,
  precomputed_results_file = NULL,
  filename = "test",
  foldername = NULL,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`foldername`	Character string specifying the name of the folder to be created in the current working directory to store intermediate results. If NULL, a name is generated automatically.

`.modEnv`	Optional internal environment used to store model indices and cached results across model-selection steps.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments forwarded to mod.run().

Details

This step is executed only when the starting model code specifies Michaelis–Menten elimination (mm = 1). If mm is not equal to 1 in the starting model, no model comparison is performed.

Value

A list with the following elements:

- results_table: A data.frame summarizing the evaluated models.
- best_code: A named integer vector representing the selected model code.
- best_row: A one-row data.frame corresponding to the selected model.

Author(s)

Zhonghui Huang

See Also

`mod.run`, `base_model`, `penaltyControl`

Examples

```
dat <- pheno_sd
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(0.008)
param_table$init[param_table$Name == "lvc"] <- log(0.6)
penalty.control <- penaltyControl()
penalty.control$penalty.terms <-
  c("rse", "theta", "covariance", "shrinkage", "omega")
start.mod <- base_model("ivbase")
start.mod["mm"] <- 1L
step_iiv_km(
  dat = dat,
  start.mod = start.mod,
  search.space = "ivbase",
  param_table = param_table,
  filename = "step_etakm_test",
  penalty.control = penalty.control,
 saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
```

step_rv *Evaluate residual error model structure*

Description

Evaluates alternative residual error model structures by modifying the residual variability setting in the model code.

Usage

```
step_rv(
  dat,
  start.mod = NULL,
  search.space = "ivbase",
  no.cores = NULL,
  param_table = NULL,
  penalty.control = NULL,
  precomputed_results_file = NULL,
  filename = "test",
  foldername = NULL,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of `ivbase` or `oralbase`. Default is `ivbase`.
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`param_table`	Optional parameter table used during model estimation.
`penalty.control`	Optional penalty control object used for reporting penalty terms in the results table.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`foldername`	Character string specifying the name of the folder to be created in the current working directory to store intermediate results. If NULL, a name is generated automatically.

`.modEnv`	Optional environment used to store model indices and cached results across steps.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to the model estimation function.

Details

Candidate models are constructed by assigning different residual error types to the model code. Each candidate differs only in the residual variability specification, and all other structural and statistical components are kept unchanged. Model selection is based on comparison of Fitness values obtained during estimation.

Value

A list with the following elements:

- results_table: A data frame summarizing the evaluated residual error models and their fit statistics,
- best_code: A named integer vector corresponding to the selected model code,
- best_row: A one-row data frame containing the summary of the selected model.

Author(s)

Zhonghui Huang

See Also

`mod.run`, `base_model`, `penaltyControl`

Examples

```
dat <- pheno_sd
param_table <- initialize_param_table()
param_table$init[param_table$Name == "lcl"] <- log(0.008)
param_table$init[param_table$Name == "lvc"] <- log(0.6)
penalty.control <- penaltyControl()
penalty.control$penalty.terms <-
  c("rse","theta", "covariance","shrinkage","omega","correlation","sigma")
step_rv(
  dat = dat,
  search.space = "ivbase",
  param_table = param_table,
  filename = "step_rv_test",
  penalty.control = penalty.control,
  saem.control = nlmixr2est::saemControl(logLik = TRUE,nBurn=15,nEm=15)
)
```

tabu.operator *Tabu search operator for model selection*

Description

Performs tabu search to explore the pharmacometric model space and identify the best-performing model. Supports both IV and Oral search spaces.

Usage

```
tabu.operator(
  dat,
  param_table = NULL,
  start.mod = NULL,
  search.space = c("ivbase", "oralbase"),
  no.cores = NULL,
  tabu.control = tabuControl(),
  penalty.control = penaltyControl(),
  precomputed_results_file = NULL,
  foldername = NULL,
  filename = "test",
  seed = 1234,
  .modEnv = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`dat`	A data frame containing pharmacokinetic data in standard nlmixr2 format, including "ID", "TIME", "EVID", and "DV", and may include additional columns.
`param_table`	Optional data frame of initial parameter estimates. If NULL, the table is generated by `auto_param_table()`.
`start.mod`	A named integer vector specifying the starting model code. If NULL, a base model is generated using `base_model()`.
`search.space`	Character, one of "ivbase" or "oralbase". Default is "ivbase".
`no.cores`	Integer. Number of CPU cores to use. If NULL, uses `rxode2::getRxThreads()`.
`tabu.control`	A list of Tabu Search control parameters from `tabuControl`:

tenure Integer. Number of iterations a move remains tabu.

niter Integer. Maximum number of search iterations.

start.point Optional initial model code vector.

aspiration Logical. If TRUE, allows aspiration criterion.

policy Character. Tabu restriction policy: `move` or `attribute`. See Details.

nsize Optional integer. Maximum number of neighbors randomly sampled from the full neighborhood (candidate list strategy).

`penalty.control`	A list of penalty control parameters defined by `penaltyControl()`, specifying penalty values used for model diagnostics during fitness evaluation.
`precomputed_results_file`	Optional path to a CSV file of previously computed model results used for caching.
`foldername`	Character string specifying the folder name for storing intermediate results. If `NULL` (default), `tempdir()` is used for temporary storage. If specified, a cache directory is created in the current working directory.
`filename`	Optional character string used as a prefix for output files. Defaults to "test".
`seed`	Integer. Random seed controlling the random sampling steps of the tabu operator for reproducible runs. Default is 1234.
`.modEnv`	Environment for storing intermediate results. If `NULL`, a new environment is created.
`verbose`	Logical. If TRUE, print progress messages.
`...`	Additional arguments passed to `mod.run()`.

Details

This function implements tabu search for pharmacometric model structure optimization. Models are encoded as bit vectors representing structural and statistical components.

Neighbor Generation and Validation

Each iteration generates neighbors by one-bit flips, then validates them using `validStringcat()`. The algorithm maintains both:

- `neighbors_orig`: Original neighbors (before validation) $\rightarrow$ used to detect intended moves
- `neighbors_val`: Validated neighbors (after validation) $\rightarrow$ used for fitness evaluation

This separation is critical because validation may introduce secondary changes. For example, changing `no.cmpt` from 2 to 3 might force `eta.vp = 0` to maintain model legality. The tabu list records only the intended change (`no.cmpt`), not validation side effects (`eta.vp`).

Tabu List Policies

Two restriction policies are available:

- `"move"`: Forbids specific transitions (e.g., `no.cmpt: 2 -> 3` and `3 -> 2`). Stores both forward and reverse moves.
- `"attribute"`: Forbids setting a parameter to a specific value regardless of origin (e.g., any move setting `no.cmpt = 3`).

Both policies use the same data structure (`element`, `from`, `to`, `tabu.iteration.left`). For attribute-based policy, the `from` field is stored for record-keeping but only `to` is used in tabu checking.

Aspiration Criterion When enabled, tabu moves are allowed if they produce a solution better than the global best.

Perturbation If the search returns to a previous starting point (cycling detected), a 2-bit perturbation is applied to escape the local region.

Value

An object of class `tabuOperatorResult`, containing:

`Final Selected Code`
: Vector representation of the best model.

`Final Selected Model Name`
: Selected best model (human-readable).

`Model Run History`
: Data frame of all model evaluations with fitness values.

`Search History`
: List with iteration-level history: `starting.points.history`, `local.best.history`, `tabu.elements.history`, `neighbors.history`.

Author(s)

Zhonghui Huang

See Also

`tabuControl` for control parameters, `detect_move` for move detection, `is_move_tabu` for tabu checking, `perturb_2bit` for perturbation

Examples

```
# Example usage with phenotype dataset
outs <- tabu.operator(
  dat = pheno_sd,
  param_table = NULL,
  search.space = "ivbase",
  tabu.control = tabuControl(),
  saem.control = nlmixr2est::saemControl(
    seed = 1234,
    nBurn = 200,
    nEm   = 300,
    logLik = TRUE
  )
)
print(outs)
```

`tabuControl` *Control Parameters for Tabu Search*

Description

Creates a list of control settings for the `tabu.operator` function.

Usage

```
tabuControl(
  tenure = 3,
  niter = 20,
  aspiration = TRUE,
  nsize = NULL,
  policy = "attribute"
)
```

Arguments

`tenure`	Integer. Number of iterations a move remains tabu.
`niter`	Integer. Maximum number of search iterations.
`aspiration`	Logical. Whether to apply the aspiration criterion. If TRUE, tabu moves are allowed if they yield a solution strictly better than the global best found so far.
`nsize`	Optional integer. If not NULL, restricts neighborhood sear to a random subset of this size (candidate list strategy).
`policy`	Character. Type of tabu restriction: • `"attribute"` — forbid revisiting a variable value (default). • `"move"` — forbid only specific from–to transitions.

Value

A named list containing all tabu control parameters.

Author(s)

Zhonghui Huang

Examples

```
tabuControl()
```

`validStringbinary` *Validate and correct model string for GA*

Description

Validates model parameter strings from genetic algorithms.

Usage

```
validStringbinary(string, search.space = "ivbase", custom_config = NULL)
```

Arguments

`string`	Numeric vector representing binary model encoding (0/1).
`search.space`	Character string specifying which search space to use. Options are "ivbase", "oralbase", or "custom". Default is "ivbase".
`custom_config`	List, configuration for custom search spaces. Required when search.space is "custom".

Details

The input string is a binary chromosome (0/1). The function:

1. Decodes the binary chromosome to a categorical parameter vector using `decodeBinary`.
2. Applies model constraints in categorical space by calling `validStringcat`.
3. Encodes the corrected categorical vector back to binary using `encodeBinary`.

This design keeps all correction rules in `validStringcat` and makes the GA version a thin wrapper around the categorical validator.

Value

Numeric vector of validated and corrected parameters (binary encoding).

Author(s)

Zhonghui Huang

See Also

`validStringcat` for categorical validation used by ACO/TS. `decodeBinary` and `encodeBinary` for encoding conversions.

Examples

```
# Example 1: ivbase, 1 compartment disables peripheral terms.
# Bits 1-2 encode no.cmpt; here 00 maps to 1.
invalid_iv <- c(0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1)
validStringbinary(invalid_iv, "ivbase")

# Example 2: oralbase, mm = 0 forces eta.km to 0.
# Bits 12-13 encode rv for oralbase.
invalid_oral <- c(1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1)
validStringbinary(invalid_oral, "oralbase")

# Example 3: custom, mcorr is cleared when there are not enough IIV terms.
simple_config <- list(
  route = "bolus",
  params = c("eta.vc", "mcorr", "rv"),
  param_dependencies = list(),
  fixed_params = list(no.cmpt = 1, eta.cl = 1, allometric_scaling = 1)
)
```

```
# custom encoding: eta.vc (1 bit), mcorr (1 bit), rv (2 bits)
invalid_custom <- c(0, 1, 1, 1)  # eta.vc=0, mcorr=1, rv=4
validStringbinary(invalid_custom, "custom", custom_config = simple_config)
```

`validStringcat` *Validate and correct model string for ACO/TS*

Description

Validates model parameter strings from ACO or tabu search algorithms.

Usage

```
validStringcat(string, search.space = "ivbase", custom_config = NULL)
```

Arguments

`string`	Numeric vector representing categorical model encoding.
`search.space`	Character string specifying which search space to use. Options are "ivbase", "oralbase", or "custom". Default is "ivbase".
`custom_config`	List, configuration for custom search spaces. Required when search.space is "custom".

Details

The input string is interpreted using the parameter order defined by the selected search space configuration (for "custom", this is custom_config$params). The function:

1. Maps the input vector to named parameters via parseParams().
2. Enforces model constraints via applyParamDeps().
3. Returns only the parameters that belong to the search space, using the same order as space_cfg$params.

This design ensures the returned vector is compatible with downstream model generation and with binary encoding wrappers (for example, validStringbinary()).

Value

Numeric vector of validated and corrected parameters (categorical).

Author(s)

Zhonghui Huang

See Also

`validStringbinary` for the GA wrapper using binary encoding. `parseParams` for mapping vectors to named parameters. `applyParamDeps` for constraint enforcement rules.

Examples

```
# Example 1: ivbase, 1 compartment disables peripheral terms.
invalid_iv <- c(1, 1, 1, 1, 0, 1, 0, 0, 0, 1)
validStringcat(invalid_iv, "ivbase")

# Example 2: oralbase, mm = 0 forces eta.km to 0.
invalid_oral <- c(2, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1)
validStringcat(invalid_oral, "oralbase")

# Example 3: custom, mcorr is cleared when there are not enough IIV terms.
simple_config <- list(
  route = "bolus",
  params = c("eta.vc", "mcorr", "rv"),
  param_dependencies = list(),
  fixed_params = list(no.cmpt = 1, eta.cl = 1, allometric_scaling = 1)
)
invalid_custom <- c(0, 1, 4)
validStringcat(invalid_custom, "custom", custom_config = simple_config)
```

Index