

Package: macpan2 (via r-universe)

October 19, 2026

Title Fast and Flexible Compartmental Modelling

Version 3.5.1

Description Tools for building and calibrating compartmental models of infectious disease.

License GPL-3

Depends R (>= 4.1.0)

Imports TMB (>= 1.9.0), oor, MASS, memoise, stats, utils

Suggests jsonlite, covr, knitr, rmarkdown, dplyr, tidyr, ggplot2, cowplot, lubridate, stringr, testthat (>= 3.0.0), htmltools, visNetwork, kableExtra, broom.mixed, DEoptim, outbreaks, tmbstan, numDeriv, parallel, macpan2helpers, iiddata.api, graph, Rgraphviz, Rcpp, deSolve

LinkingTo TMB, RcppEigen, Rcpp

VignetteBuilder knitr

BugReports <https://github.com/canmod/macpan2/issues>

Additional_repositories <https://canmod.r-universe.dev>

Remotes canmod/oor, canmod/macpan2helpers,
iidda.api=canmod/iidda-tools/R/iidda.api

Config/testthat/edition 3

Config/Needs/website canmod/macpan2

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL <https://canmod.github.io/macpan2/>,
<https://github.com/canmod/macpan2>

Repository <https://canmod.r-universe.dev>

Date/Publication 2026-03-12 12:12:28 UTC

RemoteUrl <https://github.com/canmod/macpan2>

RemoteRef HEAD

RemoteSha 13837d8225781b9c9522f218654bb69012dc3858

Contents

BinaryOperator	4
comparison	5
distribution	6
dot_layout	8
empty_matrix	9
empty_trajectory	10
engine_eval	10
engine_functions	11
find_all_paths	25
fit_distr_params	26
LedgerDefinition	27
mp_absolute_flow	28
mp_adjacency	28
mp_aggregate	29
mp_cartesian	29
mp_change_frame	30
mp_default	31
mp_dynamic_model	32
mp_dynamic_simulator	32
mp_effects_descr	33
mp_expand	34
mp_extract	35
mp_factors	35
mp_final	36
mp_flow_frame	36
mp_forecaster	37
mp_functions_used	38
mp_group	38
mp_index	39
mp_initial	41
mp_join	41
mp_kronecker_operator	43
mp_labels	44
mp_layout_grid	44
mp_layout_paths	45
mp_ledgers	46
mp_linear	47
mp_lookup	47
mp_model_docs	48
mp_model_starter	48
mp_opt_attempted	49

mp_optimize	49
mp_optimized_spec	52
mp_optimizer_output	53
mp_par	53
mp_parameterization	54
mp_per_capita_flow	54
mp_positions	56
mp_print_obj_fn	57
mp_print_spec	57
mp_rbf	58
mp_read_rds	59
mp_reference	59
mp_rename	60
mp_show_models	60
mp_sim_bounds	61
mp_sim_offset	62
mp_simulator	63
mp_slices	63
mp_square	64
mp_state_dependence_frame	64
mp_structured_vector	65
mp_subset	66
mp_symmetric	66
mp_time_scale	67
mp_tmb	67
mp_tmb_calibrator	68
mp_tmb_coef	70
mp_tmb_expr_list	70
mp_tmb_fixef_cov	72
mp_tmb_insert	72
mp_tmb_insert_backtrans	75
mp_tmb_insert_glm_timevar	76
mp_tmb_insert_log_linear	77
mp_tmb_insert_reports	79
mp_tmb_insert_trans	80
mp_tmb_library	81
mp_tmb_model_spec	82
mp_tmb_objective	84
mp_tmb_profile	85
mp_tmbstan_coef	86
mp_traj	86
mp_trajectory	87
mp_triangle	90
mp_uncertainty_estimated	90
mp_union	91
mp_vars	91
mp_version	93
mp_version_update	93

mp_zero_vector	94
names_and_labels	94
nlist	97
rbf	97
Reader	98
si_example	98
simple_sims	99
sparse_matrix_notation	100
state_updates	101
StringData	102
to_positions	104
to_string	104
Transform	105
transform_distr_param	105

Index	107
--------------	------------

BinaryOperator	<i>Binary Operator</i>
----------------	------------------------

Description

Convert a function that represents an elementwise binary operator into one that is consistent with the C++ engine. This function is intended to clarify how **macpan2** treats binary operators, which is a little different from base **R**. The difference is clarified in vignette("elementwise_binary_operators"), and BinaryOperator is primarily used as a resource for that vignette.

Usage

```
BinaryOperator(operator)
```

```
mp_binary_operator(operator)
```

Arguments

operator	A binary operator. Valid binary operations have the following characteristics.
	- They are functions.
	- They have exactly two arguments.
	- They do not have any ... arguments

Value

A binary operator consistent with the C++ engine.

Functions

- BinaryOperator(): Synonym of mp_binary_operator.

Examples

```

set.seed(1L)
A = matrix(abs(rnorm(6)), 3, 2) # 3 by 2 matrix
x = matrix(abs(rnorm(3)))      # 3 by 1 matrix
y = t(abs(rnorm(2)))          # 1 by 2 matrix
times = BinaryOperator(`*`)
pow = BinaryOperator(`^`)
identical(times(A, x), times(x, A)) ## TRUE
identical(pow(A, y), pow(y, A)) ## FALSE

```

comparison

Comparison Functions

Description

Comparison Functions

Usage

```

all_equal(x, y)

all_consistent(x, y)

not_all_equal(x, y)

all_not_equal(x, y)

```

Arguments

x [character](#) object

y [character](#) object

Functions

- `all_equal()`: Is it true that all corresponding elements of x and y are equal, have the same shape, and have no missing values?
- `all_consistent()`: Is it true that all corresponding elements of x and y are either equal or at least one is a blank string, have the same shape, and have no missing values?
- `not_all_equal()`: Complement of `all_equal`.
- `all_not_equal()`: Do not know yet. Currently unused; should we remove?

distribution

*Distributions***Description**

Distributions which can be used to specify prior or likelihood components in model calibration.

- Uniform Distribution (Improper), only appropriate for prior components - `mp_unif`
- Normal Distribution - `mp_norm`
- Log-Normal Distribution - `mp_log_norm`
- Logit-Normal Distribution - `mp_logitnorm`
- Poisson Distribution - `mp_pois`
- Negative Binomial Distribution - `mp_nbinom`

Usage

```
mp_unif(trans_distr_param = list())
```

```
mp_uniform(trans_distr_param = list())
```

```
mp_norm(
  location = mp_distr_param_null("location"),
  sd,
  trans_distr_param = list(location = mp_identity, sd = mp_log)
)
```

```
mp_normal(
  location = mp_distr_param_null("location"),
  sd,
  trans_distr_param = list(location = mp_identity, sd = mp_log)
)
```

```
mp_lnorm(
  location = mp_distr_param_null("location"),
  sd,
  trans_distr_param = list(location = mp_identity, sd = mp_identity)
)
```

```
mp_log_normal(
  location = mp_distr_param_null("location"),
  sd,
  trans_distr_param = list(location = mp_identity, sd = mp_identity)
)
```

```

mp_logitnorm(
  location = mp_distr_param_null("location"),
  sd,
  trans_distr_param = list(location = mp_identity, sd = mp_identity)
)

mp_logit_normal(
  location = mp_distr_param_null("location"),
  sd,
  trans_distr_param = list(location = mp_identity, sd = mp_identity)
)

mp_pois(
  location = mp_distr_param_null("location"),
  trans_distr_param = list(location = mp_identity)
)

mp_poisson(
  location = mp_distr_param_null("location"),
  trans_distr_param = list(location = mp_identity)
)

mp_nbinom(
  location = mp_distr_param_null("location"),
  disp,
  trans_distr_param = list(location = mp_identity, disp = mp_log)
)

mp_neg_bin(
  location = mp_distr_param_null("location"),
  disp,
  trans_distr_param = list(location = mp_identity, disp = mp_log)
)

```

Arguments

trans_distr_param	Named list of transformations for each distributional parameter. See transform_distr_param for a list of available transformations.
location	Location parameter. Specifying the location parameter is only necessary when the distribution is used as a prior distribution. If it is used as a likelihood component the location parameter will be taken as the simulated variable being fitted to data, and so this location parameter should be left to the default.
sd	Standard deviation parameter.
disp	Dispersion parameter.

Details

All distributional parameter arguments can be specified either as a numeric value, a character string giving the parameter name, or a distributional parameter object (See [fit_distr_params](#)).

dot_layout

Create a Graph from a Model Specification

Description

Convert a compartmental model specification into a graph (using the graph package) of the box diagram for the model.

Usage

```
mp_dot_layout(
  spec,
  include_inout = FALSE,
  action = c("render", "layout", "define"),
  define_args = list(edgemode = "directed"),
  layout_args = list(attrs = list(graph = list(rankdir = "LR"), node = list(shape =
    "rectangle"))),
  render_args = list(edgemode = "directed")
)

dot_layout(spec, include_inout = FALSE)
```

Arguments

spec	A model specification object (for example, created using <code>mp_tmb_model_spec()</code>).
include_inout	(logical) include nodes defined by mp_per_capita_inflow and mp_per_capita_outflow ?
action	What actions should be taken? The default, "render", will define, layout, and render the graph of the model spec. Rendering means that the graph will be rendered in the current graphics device and that the returned object will contain both layout and rendering information (see layoutGraph renderGraph). If action is "layout", then the graph will not be rendered, but the returned object will contain layout information and can therefore be rendered later using the renderGraph function. If action is "define", then the returned object will contain the definition of the graph but not any layout or rendering information.
define_args	List of additional arguments to pass to the graphAM constructor function.
layout_args	List of additional arguments to pass to the layoutGraph function (only applied if action is either "layout" or "render").
render_args	List of additional arguments to pass to the renderGraph function (only applied if action is "render").

Details

In order to create these graph objects you need to install the graph package, and to plot them on the current graphics device you need to install the Rgraphviz package.

Value

A `graphAM` object.

Functions

- `dot_layout()`: Deprecated in favour of `mp_dot_layout`, which both plots and returns the `graphAM` object.

Examples

```
("macpan_base"
  |> mp_official_library()
  |> mp_dot_layout()
)
```

empty_matrix

Empty Matrix

Description

Empty matrices are useful when defining matrices that do not need to be initialized because they will get computed before they are required by other expressions. They can also provide a useful placeholder for matrices that should only have a value after a certain phase in the simulation.

Usage

```
empty_matrix
```

Format

A `numeric matrix` with zero rows and zero columns.

Examples

```
spec = mp_tmb_model_spec(during = list(x ~ time_step(0)))
identical(spec$empty_matrices()[x, empty_matrix]) ## TRUE
```

empty_trajectory	<i>Empty Trajectory</i>
------------------	-------------------------

Description

Output of `mp_trajectory` if nothing is simulated.

Usage

```
empty_trajectory
```

Format

A data frame with zero rows and the following columns: matrix, time, row, col, value.

engine_eval	<i>Engine Evaluation</i>
-------------	--------------------------

Description

Evaluate an expression in the TMB-based C++ simulation and objective function engine. This function is useful for trying out the `engine_functions` that can be used to define **macpan2** models.

Usage

```
engine_eval(
  .expr,
  ...,
  .matrix_to_return,
  .tmb_cpp = getOption("macpan2_dll"),
  .structure_labels = NullLabels()
)
```

Arguments

<code>.expr</code>	Expression as a one-sided formula, the right-hand-side of which is treated as the expression to be evaluated.
<code>...</code>	Named objects that can be coerced into numeric matrices.
<code>.matrix_to_return</code>	Optional name of one of the matrices given in <code>...</code> to be returned. If this argument is missing, the matrix that will be returned is the matrix returned by the expression on the right-hand-side of the formula.
<code>.tmb_cpp</code>	Name of a C++ program defining the engine. Typically you just want to use the default, which is <code>macpan2</code> , unless you are extending the engine yourself.
<code>.structure_labels</code>	Deprecated.

Value

Matrix being produced on the right-hand-side or matrix given in `.matrix_to_return` if it is provided.

Examples

```
engine_eval(~ exp(y), y = pi) # ~ 23.14069

# It is not currently possible to assign values to a subset of
# a matrix in a natural way (e.g. you cannot do things like x[1] = exp(y)),
# but you can achieve this functionality using the assign function.
engine_eval(~ assign(x, 1, 0, exp(y))
, x = rep(0, 2)
, y = pi
, .matrix_to_return = "x"
)
```

engine_functions

Functions Available in the Simulation Engine

Description

Functions currently supported by the C++ TMB engine for constructing expressions for defining model simulations.

Details

The quickest way to experiment with these functions is to use the [engine_eval](#) function, as in the following example that calculates a force of infection.

```
engine_eval(~ beta * I / N
, beta = 0.25
, I = 1e3
, N = 1e7
)
```

To produce a dynamical simulation that iteratively evaluates expression involving these functions, use [simple_sims](#).

```
simple_sims(
  iteration_exprs = list(x ~ x - 0.9 * x),
  time_steps = 5,
  mats = list(x = 1)
)
```

Here, $x = 0.9 * x$ is assigned to x at each of five iterations of a simulation loop.

If these expressions involve matrices with more than one element, You can control which elements in the evaluation of the right hand side go to which elements on the left hand side. This technique involves using either square brackets or the `c` function on the left hand side. For more information on assignment, please see the section on Assignment below.

Elementwise Binary Operators:

Elementwise binary operators take two matrix-valued arguments and apply a binary operator (e.g. $+$, $*$) to each set of corresponding elements, and return the corresponding matrix-valued output containing the resulting elements. What does 'corresponding' mean? If the two matrix-valued arguments have the same shape (same number of rows and columns), then two elements correspond if they occur in the same row and column position in the two matrices. If the two matrices are not of the same shape but there is one row and/or one column in either matrix, then the singleton rows and columns are recycled sufficiently many times so that they match the shape of the other matrix. If after recycling singleton rows and columns the matrices are still of different shape, then an error is thrown and the matrices are said to be incompatible.

Functions:

- $x + y$
- $x - y$
- $x * y$
- x / y
- $x ^ y$

Arguments:

- x : Any matrix with dimensions compatible with y .
- y : Any matrix with dimensions compatible with x .

Return:

- A matrix with the binary operator applied elementwise after any necessary recycling of rows and/or columns.

Examples:

```
engine_eval(~ 1 + 2)
engine_eval(~ y * z, y = 1:3, z = matrix(1:6, 3, 2))
engine_eval(~ 1 / (1 - y), y = 1/4)
```

Elementwise Math:

Functions:

- $\log(x)$: Natural logarithm.
- $\exp(x)$: Exponential function.
- $\cos(x)$: Cosine function.
- $\sin(x)$: Sine function.
- $\sqrt{x}$: Squareroot function.
- $\text{invlogit}(x)$: Inverse logit function, $1/(1 + \exp(-x))$.
- $\text{logit}(x)$: Logit function, $\log(x/(1-x))$.

Arguments:

- x : Any numeric matrix.

Return:

- A matrix with the same dimensions as x , containing the results of applying the function to each element of x .

Examples:

```
engine_eval(~ log(y), y = c(2, 0.5))
```

Limiting Values:

Functions:

- proportions(x , limit, eps)
- limit(x , limit, eps)
- clamp(x , eps, limit)

Arguments:

- x : Any matrix
- limit : numeric value to return elementwise from proportions if $\text{sum}(x) < \text{eps}$
- eps : numeric tolerance for $\text{sum}(x)$

Return:

- matrix of $x / \text{sum}(x)$ or $\text{rep}(\text{limit}, \text{length}(x))$ if $\text{sum}(x) < \text{eps}$.

Examples:

```
engine_eval(~ proportions(y, 0.5, 1e-8), y = c(2, 0.5))
```

Details:

The return value depends on the function.

- proportions : matrix of $x / \text{sum}(x)$ or $\text{rep}(\text{limit}, \text{length}(x))$ if $\text{sum}(x) < \text{eps}$.
- limit :
- clamp :

Details:

The divide_safe() function conducts elementwise division of the first two arguments, and then allows two other arguments:

- limit : Value to return if the $\text{sqrt}(\text{denominator}^2)$ is

Details:

The clamp function smoothly clamps the elements of a matrix so that they do not get closer to 0 than a tolerance, eps, with a default of 1e-12. This clamp function is the following modification of the [squareplus function](#).

$$f(x) = \epsilon_- + \frac{(x - \epsilon_-) + \sqrt{(x - \epsilon_-)^2 + (2\epsilon_0 - \epsilon_-)^2 - \epsilon_-^2}}{2}$$

Where the two parameters are defined as follows.

$$\epsilon_0 = f(0)$$

$$\epsilon_- = \lim_{x \rightarrow -\infty} f(x)$$

This function is differentiable everywhere, monotonically increasing, and $f(x) \approx x$ if x is positive and not too close to zero. By modifying the parameters, you can control the distance

between $f(x)$ and the horizontal axis at two 'places' – 0 and $-\infty$. See [issue #93](#). for more information.

For `clamp` the arguments specifically mean.

- `x` : A matrix with elements that should remain positive.
- `eps` : A small positive number, $\epsilon_0 = f(0)$, giving the value of the function when the input is zero. The default value is 1e-11
- `limit` : A small positive number,

$$\epsilon_- = \lim_{x \rightarrow -\infty} f(x)$$

, giving the value of the function as the input goes to negative infinity. The default is `limit` = 1e-12. This `limit` should be chosen to be less than `eps` to ensure that `clamp` is twice differentiable.

Integer Sequences:

Functions:

- `from:to` : Inclusive and ordered sequence of integers between two bounds.
- `seq(from, length, by)` : Ordered sequence of integers with equal spacing between adjacent values.

Arguments:

- `from` : Scalar integer giving the first integer in the sequence.
- `to` : Scalar integer giving the last integer in the sequence.
- `length` : Number of integers in the sequence.
- `by` : Scalar giving the difference between adjacent values in the sequence.

Return:

- Column vector with a sequence of integers.

Details:

The colon operator works much like the base R version `:`. It takes two scalar-valued integers and returns a column vector with all integers between the two inputs.

The `seq` function is a little different from the base R default, [seq](#), in that it allows the user precise control over the length of the output through the `length` argument. The base R function gives the user this option, but not as the default.

Examples:

```
engine_eval(~ 1:10)
engine_eval(~ seq(1, 10, 2))
```

Replicate Elements

Functions:

- `rep(x, times)` : Replicate a column vector a number of times, by repeatedly stacking it on top of itself.
- `recycle(x, rows, cols)` : Recycle rows and columns of `x` to produce a matrix with rows rows and cols columns.

Arguments:

- `x` : A scalar-valued variable to repeat.
- `times` : A scalar-valued integer variable giving the number of times to repeat `x`.

- rows : Number of rows in the output of recycle.
- cols : Number of columns in the output of recycle.

Return:

- Column vector with times copies of x stacked on top of each other.

Examples:

```
engine_eval(~ rep(1, 10))
engine_eval(~ recycle( 1:3, 3, 4))
engine_eval(~ recycle(t(1:4), 3, 4))
```

Matrix Multiplication:

Functions:

- `x %*% y` : Standard matrix multiplication.
- `x %x% y` : Kronecker product
- `sparse_mat_mult(x, i, j, y, z)` : Matrix multiplication when the left matrix is represented as a column vector, x, of non-zero elements and integer vectors of row, i, and column, j, indices. The right matrix and the resulting matrix are not represented as sparse matrices.

Arguments:

- x : A matrix.
- y : A matrix.
- i : Integer vector the same length as x giving zero-based row indices for sparse matrix representation.
- j : Integer vector the same length as x giving zero-based column indices for sparse matrix representation.
- z : A matrix with dimensions equal to the result of the sparse matrix multiplication (see details).

Return:

- The matrix product of x and y.

Examples:

```
engine_eval(~ (1:10) %*% t(1:10))
engine_eval(~ (1:10) %x% t(1:10))
```

Details:

For standard matrix multiplication, `x %*% y`, the number of columns of x equals the number of rows of y.

Think about `sparse_mat_mult(x, i, j, y, z)` as similar to `z ~ x %*% y`, where x is represented differently. In particular, the argument x is a column vector containing the non-zero elements of the left matrix, i contains the zero-based row indices associated with each element in x, and j contains the zero-based column indices associated with each element in x.

Parenthesis:

The order of operations can be enforced in the usual way with round parentheses, `(`.

Reshaping and Combining Matrices:

Functions:

- `c(...)` : Stack columns of arguments into a single column vector.
- `cbind(...)` : Create a matrix containing all of the columns of a group of matrices with the same number of rows.
- `rbind(...)` : Create a matrix containing all of the rows of a group of matrices with the same number of columns.
- `matrix(x, rows, cols)` : Reshape a matrix to have `rows` rows and `cols` columns. The input `x` must have `rows * cols` elements.
- `t(x)` : Standard matrix transpose.

Arguments:

- `...` : Any number of dimensionally consistent matrices. The definition of dimensionally consistent depends on the function.
- `x` : Can be any matrix for `t`, but for `matrix` it must have `rows * cols` elements.
- `rows` : Scalar integer giving the number of rows in the output.
- `cols` : Scalar integer giving the number of columns in the output.

Return:

- A combined or reshaped matrix.

Details:

Any number of column vectors can be combined into a bigger column vector.

Column and row vectors of the same length can be combined using the `cbind` and `rbind` functions respectively

The `matrix` function can be used to redefine the numbers of rows and columns to use for arranging the values of a matrix. It works similarly to the base R `matrix` function in that it takes the same arguments. On the other hand, this function differs substantially from the base R version in that it must be filled by column and there is no `byrow` option.

Matrices can be transposed with the usual function, `t`.

Examples:

```
engine_eval(~ c(a, b, c), a = 1, b = 10:13, c = matrix(20:25, 3, 2))
engine_eval(~ cbind(a, 10 + a), a = 0:3)
engine_eval(~ rbind(a, 10 + a), a = t(0:3))
engine_eval(~ matrix(1:12, 4, 3))
engine_eval(~ t(1:3))
```

Matrix Diagonals:

Functions:

- `to_diag(x)` : Create a diagonal matrix by setting the diagonal to a column vector, `x`.
- `from_diag(x)` : Extract the diagonal from a matrix, `x`, and return the diagonal as a column vector.

Arguments:

- `x` : Any matrix (for `from_diag`) or a column vector (for `to_diag`). It is common to assume that `x` is square for `from_diag` but this is not required.

Return:

- `to_diag(x)` : Diagonal matrix with `x` on the diagonal.
- `from_diag(x)` : Column vector containing the diagonal of `x`. A value is considered to be on the diagonal if it has a row index equal to the column index.

Details:

The `to_diag` function can be used to produce a diagonal matrix by setting a column vector equal to the desired diagonal. The `from_diag` does (almost) the opposite, which is to get a column vector containing the diagonal of an existing matrix.

Examples:

```
engine_eval(~from_diag(matrix(1:9, 3, 3)))
engine_eval(~to_diag(from_diag(matrix(1:9, 3, 3))))
engine_eval(~from_diag(to_diag(from_diag(matrix(1:9, 3, 3)))))
```

Summarizing Matrix Values:*Functions:*

- `sum(...)` : Sum all of the elements of all of the matrices passed to ...
- `col_sums(x)` : Row vector containing the sums of each column.
- `row_sums(x)` : Column vector containing the sums of each row.
- `group_sums(x, f, n)` : Column vector containing the sums of groups of elements in `x`. The groups are determined by the integers in `f` and the order of the sums in the output is determined by these integers.
- `mean(x)` : Arithmetic average of all elements in matrix `x`.
- `sd(x)` : Sample standard deviation of all elements in matrix `x`.

Arguments:

- ... : Any number of matrices of any shape.
- `x` : A matrix of any dimensions, except for `group_sums` that expects `x` to be a column vector.
- `f` : A column vector the same length as `x` containing integers between 0 and `m-1`, given `m` unique groups. Elements of `f` refer to the indices of `x` that will be grouped and summed.
- `n` : A column vector of length `m`. If `f` does not contain group `k` in `[0, m-1]`, `group_sums` skips this group and the output at index `k+1` is `n[k+1]`.

Return:

- A matrix containing sums of subsets of the inputs.

Details:

The `row_sums` and `col_sums` are similar to the base R `rowSums` and `colSums` functions, but with slightly different behaviour. In particular, the `row_sums` function returns a column vector and the `col_sums` function returns a row vector. If a specific shape is required then the transpose `t` function must be explicitly used.

Examples:

```
x = 1
y = 1:3
A = matrix(1:12, 4, 3)
engine_eval(~ sum(y), y = y)
engine_eval(~ sum(x, y, A), x = x, y = y, A = A)
engine_eval(~ col_sums(A), A = A)
engine_eval(~ row_sums(A), A = A)
engine_eval(~ group_sums(x, f, n), x = 1:10, f = rep(0:3, 1:4), n = c(1:4))
```

Sweeping Matrix Elements:

Functions:

- `cumsum(x)` : Return a matrix with columns containing the cumulative sum of the columns in `x`.

Arguments:

- `x` : A matrix.

Return:

A matrix the same size as `x` but with columns containing the cumulative sum of the columns in `x`.

Extracting Matrix Elements:*Functions:*

- `x[i, j]` : Return a matrix containing a subset of the rows and columns of `x`.
- `block(x, i, j, n, m)` : Return a matrix containing a contiguous subset of rows and columns of `x` https://eigen.tuxfamily.org/dox/group__TutorialBlockOperations.html.
- `last(x)` : The last element of a matrix (i.e., the lower-right element).

Arguments:

- `x` : Any matrix.
- `i` : An integer column vector (for `[]`) or integer scalar (for `block`) containing the indices of the rows to extract (for `[]`) or the index of the first row to extract (for `block`).
- `j` : An integer column vector (for `[]`) or integer scalar (for `block`) containing the indices of the columns to extract (for `[]`) or the index of the first column to extract (for `block`). If `j` is missing in a call to `[]`, it is assumed to be `j = 0` although we might change this default to be the vector of all column indices.
- `n` : Number of rows in the block to return.
- `m` : Number of columns in the block to return.

Return:

- A matrix containing a subset of the rows and columns in `x`.

Details:

Note that zero-based indexing is used so the first row/column gets index, 0, etc. The `block` function is expected to be more efficient than `[]` when the elements to be extracted are contiguous.

Examples:

```
engine_eval(~ A[c(3, 1, 2), 2], A = matrix(1:12, 4, 3))
engine_eval(~ block(x, i, j, n, m), x = matrix(1:12, 4, 3), i=1, j=1, n=2, m=2)
engine_eval(~ last(A), A = matrix(1:12, 4, 3))
```

Accessing Past Values in the Simulation History:

For matrices with their simulation history saved, it is possible to bind the rows or columns of past versions of such matrices into a single matrix.

Functions:

- `rbind_lag(x, lag, t_min)` : Bind the rows of versions of `x` that were recorded at the end of all simulation iterations corresponding to time lags given by integers in `lag`.
- `rbind_time(x, t, t_min)` : Bind the rows of versions of `x` that were recorded at the end of all simulation iterations corresponding to integers in `t`.

- `cbind_lag(x, lag, t_min)` : Bind the columns of versions of `x` that were recorded at the end of all simulation iterations corresponding to time lags given by integers in `lag`. (TODO – `cbind_lag` is not developed yet)
- `cbind_time(x, t, t_min)` : Bind the columns of versions of `x` that were recorded at the end of all simulation iterations corresponding to integers in `t`. (TODO – `cbind_lag` is not developed yet)

Arguments:

- `x` : Any matrix with saved history such that the number of columns (for `rbind_*`) or rows (for `cbind_*`) does not change throughout the simulation.
- `lag` : Integer vector giving numbers of time steps before the current step to obtain past values of `x`.
- `t` : Integer vector giving time steps at which to obtain past values of `x`.
- `t_min` : Integer giving the minimum time step that is allowed to be accessed. All time-steps in `t` or implied by `lag` that are before `t_min` are ignored.

Return:

- A matrix containing values of `x` from past times.

Time Indexing:

Get or update the index of the current or lagged time step or the index of the current time group. A time group is a contiguous set of time steps defined by two change points.

Functions:

- `time_step(lag)`: Get the time-step associated with a particular lag from the current time-step. If the lagged time-step is less than zero, the function returns zero.
- `time_group(index, change_points)`: Update the index associated with the current time group. The current group is defined by the minimum of all elements of `change_points` that are greater than the current time step. The time group index is the index associated with this element. Please see the examples below, they are easier to understand than this explanation.
- `time_var(x, change_points)`: An improvement to `time_group`. Returns values in `x` at time steps in `change_points`, return value remains constant between `change_points`.

Arguments:

- `x`: Column vector representing a time series. `time_var` will return the value of `x` corresponding to element in `change_points` that contains the current time.
- `lag`: Number of time-steps to look back for the time-step to return.
- `change_points`: Increasing column vector of time steps giving the lower bound of each time group.

Return:

A 1-by-1 matrix with the time-step `lag` steps ago, or with zero if `t+1 < lag`

Examples:

```
simple_sims(
  iteration_exprs = list(x ~ time_step(0)),
  time_steps = 10,
  mats = list(x = empty_matrix)
)
```

```

sims = simple_sims(
  iteration_exprs = list(
    j ~ time_group(j, change_points),
    time_varying_parameter ~ time_variation_schedule[j]
  ),
  mats = list(
    j = 0,
    change_points = c(0, 4, 7),
    time_variation_schedule = c(42, pi, sqrt(2)),
    time_varying_parameter = empty_matrix
  ),
  time_steps = 10,
)
set.seed(1L)
change_points = c(0,2,5)
x_val = rnorm(length(change_points))
simple_sims(
  iteration_exprs = list(x ~ time_var(x_val,change_points))
  , int_vecs = list(change_points = change_points)
  , mats = list(x = empty_matrix, x_val=x_val)
  , time_steps = 10
)

```

Convolution:

You can take the convolution of each element in a matrix, x , over simulation time using a kernel, k .

Functions:

- `convolution(x, k)`

Arguments:

- x : The matrix containing elements to be convolved.
- k : A column vector giving the convolution kernel.

Return:

A matrix the same size as x but with the convolutions, y_{ij} , of each element, x_{ij} . The value of y_{ij} at time $t = 1, \dots, T$ is given by the following.

$$y_{ij}(t) = \sum_{\tau=0}^{\min(t,m)-1} x_{ij}(t-\tau)k_{\tau}$$

Where:

- $x_{ij}(t)$: value of x_{ij} at time step t .
- $y_{ij}(t)$: value of y_{ij} at time step t .
- $t = 1, \dots, T$: the time step.
- $\tau = 0, \dots, m-1$: index of the time lag for a kernel of length m .
- k_{τ} : value of the kernel associated with lag τ .

Details:

If any empty matrices are encountered when looking back in time, they are treated as matrices with all zeros. The convolution of a matrix of all positive values will be biased low for all time steps less than the length of the kernel (i.e., for all time steps such that $t < m$), and therefore one should only compare observed data with a convolution (e.g., when calibrating) for time steps less than m .

Examples:

```
simple_sims(
  list(
    x ~ 3 * x * (1 - x),
    y ~ convolution(x, rep(1/10, 10))
  ),
  time_steps = 50,
  mats = list(x = 0.5, y = empty_matrix)
)
```

Probability Densities:

All probability densities have the same first two arguments.

- observed
- simulated

The simulated argument gives a matrix of means for the observed values at which the densities are being evaluated. Additional arguments are other distributional parameters such as the standard deviation or dispersion parameter. All densities are given as log-densities, so if you would like the density itself you must pass the result through the exp function.

If the simulated matrix or the additional parameter matrices have either a single row or single column, these singleton rows and columns are repeated to match the number of rows and columns in the observed matrix. This feature allows one to do things like specify a single common mean for several values.

Functions:

- `dpois(observed, simulated)` : Log of the Poisson density based on this `dpois` TMB function.
- `dnbinom(observed, simulated, over_dispersion)` : Log of the negative binomial density based on this `dnbinom` TMB function. To get the variance that this function requires we use this expression, `simulated + simulated^2/over_dispersion`, following p.165 in this [book](#)
- `dnorm(observed, simulated, standard_deviation)` : Log of the normal density based on this `dnorm` TMB function.
- `dbinom(observed, size, probability)` : Log of the binomial density based on the `dbinom` TMB function.

Arguments:

- `observed` : Matrix of observed values at which the density is being evaluated.
- `simulated` : Matrix of distributional means, with singleton rows and columns recycled to match the numbers of rows and columns in observed.
- `over_dispersion` : Over-dispersion parameter given by $(\text{simulated}/\text{standard_deviation})^2 - \text{simulated}$.
- `standard_deviation` : Standard deviation parameter.

- `size` : Number of Bernoulli trials.
- `probability` : Probability of a successful Bernoulli trial.

Pseudo-Random Number Generators:

All random number generator functions have mean as the first argument. Subsequent arguments give additional distributional parameters. Singleton rows and columns in the matrices passed to the additional distributional parameters are recycled so that all arguments have the same number of rows and columns. All functions return a matrix the same shape as mean but with pseudo-random numbers deviating from each mean in the mean matrix.

Functions:

- `rpois(mean)` : Pseudo-random Poisson distributed values.
- `rnbinom(mean, over_dispersion)` : Pseudo-random negative binomially distributed values.
- `rnorm(mean, standard_deviation)` : Pseudo-random normal values.
- `rbinom(size, prob)` : Pseudo-random binomial values.
- `reulermultinom(size, rate, dt)` : Pseudo-random **Euler-multinomial** values.

Arguments:

- `mean` : Matrix of means about which to simulate pseudo-random variation.
- `over_dispersion` : Matrix of over-dispersion parameters given by $(\text{simulated}/\text{standard_deviation})^2 - \text{simulated}$.
- `standard_deviation` : Matrix of standard deviation parameters.
- `size` : Matrix of numbers of trials.
- `prob` : Matrix of probabilities of success.
- `rate` : Matrix of rates, used to compute the probabilities in a multinomial distribution. The probability associated with the i th rate, r_i , is $(1 - \exp(-\sum(r * dt))) * (r_i / r)$, where r is the sum of the rates. This is not a typical multinomial distribution in that if you sum these probabilities up you do not get 1 but rather $(1 - \exp(-\sum(r * dt)))$. See details below for more on the Euler-multinomial distribution
- `dt` : Optional parameter specifying the length of the time step. See details below for more on the Euler-multinomial distribution.

Details:

The Euler-multinomial distribution is used to model how many individuals move from one compartment to a set of other compartments in a single time step of length `dt`. The rate of moving to each of these compartments is characterized by the associated element in the rate matrix. The reason why the probabilities do not sum to 1, is that not all individuals have to change compartments in a time step.

Cumulative Distribution Functions:

Lower-tail cumulative distribution functions.

Functions:

- `pgamma(q, shape, scale)` : Cumulative distribution function of the Gamma distribution. This is a lite wrapper for the **pgamma function in TMB**.
- `pnorm(q, mean, sd)` : Cumulative distribution function of the normal distribution. This is a lite wrapper for the **pnorm function in TMB**.

Arguments:

- `q` : Matrix of Quantiles.
- `shape` : Matrix of shape parameters of the Gamma distribution.
- `scale` : Matrix of scale parameters of the Gamma distribution.
- `mean` : Matrix of mean parameters of the normal distribution.
- `sd` : Matrix of standard deviation parameters of the normal distribution.

Rounding:*Functions:*

`round(x)` : Round elements of a matrix to the nearest integer.

Arguments:

- `x` : Matrix to be rounded.

Details:

Be careful if you are using rounding in a model to be calibrated. Rounding will break differentiability of the objective function if `x` depends, either directly or indirectly, on parameters being calibrated. This will lead to incorrect gradients potentially being passed to an optimizer. To be safe, do not round in models being calibrated.

Debugging Instrumentation:

Functions to use when you are trying to figure stuff out.

Functions:

- `print(x)` : Print out the value of a matrix.
- `check_finite(x)` : Stop the simulations and return an error if `x` has any non-finite values.

Arguments:

- `x` : Name of a matrix in the model.

Return:

An `empty_matrix`.

Examples:

```
simple_sims(
  list(dummy ~ print(x), x ~ x / 2)
  , time_steps = 10
  , mats = list(x = 2)
)
engine_eval(~ 1/0) ## returns Inf
engine_eval(~ check_finite(1/0)) ## returns nothing and throws an error
```

Assign (deprecated):

Assign values to a subset of the elements in a matrix.

Functions:

- `assign(x, i, j, v)`

Arguments:

- `x` : Matrix with elements that are to be updated by the values in `v`.
- `i` : Column vector of row indices pointing to the elements of `x` to be updated. These indices are paired with those in `v`. If the length of `i` does not equal that of `v`, then it must have a single index that gets paired with every element of `v`. Indices are zero-based, `i=0` corresponds to the first row.

- *j* : Column vector of column indices pointing to the elements of *x* to be updated. These indices are paired with those in *v*. If the length of *j* does not equal that of *v*, then it must have a single index that gets paired with every element of *v*. Indices are zero-based, *j*=0 corresponds to the first column.
- *v* : Column vector of values to replace elements of *x* at locations given by *i* and *j*.

Return:

The `assign` function is not called for its return value, which is an `empty_matrix`, but rather to modify *x* but replacing some of its components with those in *v*.

Examples:

```
x = matrix(1:12, 3, 4)
engine_eval(~ x + 1, x = x)
engine_eval(~ x + 1, x = x, .matrix_to_return = "x")
engine_eval(~ assign(x, 2, 1, 100), x = x, .matrix_to_return = "x")
engine_eval(~ assign(x
  , c(2, 1, 0)
  , 0
  , c(100, 1000, 10000)
), x = x, .matrix_to_return = "x")
```

Unpack (deprecated):

Unpack elements of a matrix into smaller matrices.

Functions:

- `unpack(x, ...)`

Arguments:

- *x* : Matrix with elements to be distributed to the matrices passed through . . .
- . . . : Matrices with elements to be replaced by the values of elements in *x* in column-major order. These matrices must be named matrices and not computed on the fly using expressions. Note that even subsetting (e.g. `unpack(x, y[0], y[3])`) counts as an expression. This use-case would require the `assign` function `assign(y, c(0, 3), 0, x)`.

Return:

The `unpack` function is not called for its return value, which is an `empty_matrix`, but rather to modify the matrices in . . . by replacing at least some of its components with those in *x*.

Examples:

Here we fill a matrix with integers from 1 to 12 and then unpack them one-at-a-time into two column vectors, *x* and *y*. By returning *y* we see the integers after the first three were used up by *x*.

```
engine_eval(~unpack(matrix(1:12, 3, 4), x, y)
  , x = rep(0, 3)
  , y = rep(1, 5)
  , .matrix_to_return = "y"
)
```

Assignment:

The left-hand-side of formulas sent to the simulation engine determine assignment works.

Functions:

- $y \sim x$: Assign x to y .
- $y[i] \sim x$: Assign the first column of x to those rows in the first column of y that are indexed by i .
- $y[i, j] \sim x$: Assign each element, $x[k, l]$, in x , to element, $y[i[k], j[l]]$, in y .
- $c(\dots) \sim x$: Assign the elements of the columns of x (stacked on top of each other) to the matrices in $\dots$ in the order in which they appear. If the number of columns in x equals the number of matrices in $\dots$, and if these matrices are vectors (i.e., have only a single column or a single row), then the columns of x become assigned to the vectors in $\dots$.

Arguments:

- x : Matrix containing the result of the expression on the right-hand-side.
- y : Matrix with elements that will be assigned the elements of x .
- i : Integer vector giving zero-based row indexes describing the rows in x that get the

find_all_paths

*Find all Paths Through Compartments***Description**

Find all paths through a compartmental model.

Usage

```
find_all_paths(edges_df, start_node_guesses = character(0L))
```

Arguments

`edges_df` A data frame with a `from` and a `to` column, which can be extracted from a model specification object using the [mp_flow_frame](#).

`start_node_guesses` Optional guesses for nodes that start paths. This is useful for models that are not directed acyclic graphs (DAGs).

Value

List of character vectors of state variable names, each vector describing a path through the model.

Examples

```
spec = mp_tmb_library("starter_models", "macpan_base", package = "macpan2")
spec |> mp_flow_frame() |> find_all_paths()
```

fit_distr_params	<i>Fitting Distributional Parameters</i>
------------------	--

Description

Distributional parameters can be added to the list of parameters that are fit during calibration. By default, distributional parameters in priors and likelihoods are not fit. Use `mp_nofit` to exclude distributional parameters from being fit and `mp_fit` to fit them.

Usage

```
mp_fit(x, trans = DistrParamTransDefault())
```

```
mp_nofit(x, trans = DistrParamTransDefault())
```

Arguments

<code>x</code>	numeric starting value of the distributional parameter to fit, or character name of an existing variable in the model with a default starting value to use.
<code>trans</code>	transformation to apply to the distributional parameter. By default, distributional parameters inherit a default transformation from the associated distribution. For example, the standard deviation parameter <code>sd</code> in the <code>mp_normal</code> distributions has a default log transformation specified using <code>mp_log</code> .

Value

A distributional parameter object.

Examples

```
# First we call the SIR model spec, and generate some data for calibration.
spec = mp_tmb_library("starter_models", "sir", package = "macpan2")
data = mp_simulator(spec, 50, "infection") |> mp_trajectory()

# Suppose we want to specify a Normal prior on the transmission parameter
# beta, and we are interested in estimating the prior standard deviation.
# Here we use `mp_fit` to estimate the standard deviation, `sd`, and we
# provide a numeric starting value for `sd` in the optimization.
cal = mp_tmb_calibrator(
  spec
  , data
  , traj = "infection"
  , par = list(beta = mp_normal(location = 0.35, sd = mp_fit(0.1)))
  , default = list(beta = 0.25)
)

# When viewing the calibration objective function we can see the additional
# prior density term added for beta. The standard deviation parameter has
# been automatically named 'distr_params_log_sd_beta'.
```

```

cal$simulator$tmb_model$obj_fn$obj_fn_expr

# Next we optimize and view the fitted parameters. We can see the
# distributional parameter in the coefficient table with a default value
# equal to the numeric value we provided to `mp_fit` above.
mp_optimize(cal)
if (suppressPackageStartupMessages(require(broom.mixed))) {
  print(mp_tmb_coef(cal))
}

# If instead we want control over the name of the new fitted distributional
# parameter, we can add a new variable to our model specification with the
# default value set to the desired optimization starting value.
updated_spec = spec |> mp_tmb_insert(default = list(sd_var = 0.1))

# In the calibrator, we use the name of this newly added variable, "sd_var",
# as input to `mp_fit`.
cal = mp_tmb_calibrator(
  updated_spec
  , data
  , traj = "infection"
  , par = list(beta = mp_normal(location = 0.35, sd = mp_fit("sd_var")))
  , default = list(beta = 0.25)
)

# We can see this distributional parameter get propagated to the objective
# function and the fitted parameter table.
cal$simulator$tmb_model$obj_fn$obj_fn_expr
mp_optimize(cal)
if (suppressPackageStartupMessages(require(broom.mixed))) {
  print(mp_tmb_coef(cal))
}

```

LedgerDefinition

Ledgers

Description

A ledger is a table with rows that identify specific instances of a functional form used to define a [mp_dynamic_model](#). Ledgers are most commonly created using the [mp_join](#) function as in the examples.

Examples

```

age = mp_index(Age = c("young", "old"))
state = mp_cartesian(
  mp_index(Epi = c("S", "I", "R")),
  age
)
mp_join(

```

```

    from = mp_subset(state, Epi = "S"),
    to = mp_subset(state, Epi = "I"),
    by = list(from.to = "Age")
  )

```

mp_absolute_flow	<i>Specify Absolute Flow Between Compartments (Experimental)</i>
------------------	--

Description

An experimental alternative to [mp_per_capita_flow](#) that allows users to specify flows using absolute rates instead of per-capita rates.

Usage

```
mp_absolute_flow(from, to, rate, flow_name = NULL, rate_name = NULL)
```

Arguments

from	String giving the name of the compartment from which the flow originates.
to	String giving the name of the compartment to which the flow is going.
rate	String giving the expression for the absolute flow rate per time-step.
flow_name	String giving the name for the variable that will store the rate.
rate_name	Deprecated synonym for flow_name. Please use flow_name in all future work.

See Also

[mp_per_capita_flow\(\)](#)

mp_adjacency	<i>Adjacency Matrix</i>
--------------	-------------------------

Description

Get the **adjacency matrix** associated with a model specification.

Usage

```
mp_adjacency(spec, include_inout = FALSE)
```

Arguments

spec	A model specification object (for example, created using mp_tmb_model_spec()).
include_inout	(logical) include nodes defined by mp_per_capita_inflow and mp_per_capita_outflow ?

Value

An adjacency [matrix](#).

mp_aggregate	<i>Aggregate an Index</i>
--------------	---------------------------

Description

Create a one-column ledger (see [LedgerDefinition](#)) with rows identifying instances of an aggregation.

Usage

```
mp_aggregate(index, by = "Group", ledger_column = "group")
```

Arguments

index	An index to aggregate over.
by	A column set label to group by. By default a dummy and constant "Group" column is created.
ledger_column	Name of the column in the output ledger that describes the groups.

See Also

Other functions that return ledgers [mp_join\(\)](#)

mp_cartesian	<i>Cartesian Product of Index Tables</i>
--------------	--

Description

Produce a new index table by taking all possible pairwise combinations of the input tables. This is useful for producing product models that expand model components through stratification.

Usage

```
mp_cartesian(...)
```

Arguments

...	Index tables (see mp_index).
-----	---

See Also

Other functions that return index tables [mp_index\(\)](#), [mp_rename\(\)](#), [mp_subset\(\)](#), [mp_union\(\)](#)

Other functions that take products of index tables and return one index tables [mp_linear\(\)](#), [mp_square\(\)](#), [mp_symmetric\(\)](#), [mp_triangle\(\)](#)

Examples

```

mp_cartesian(
  mp_index(Epi = c("S", "I")),
  mp_index(Age = c("young", "old"))
)

si = mp_index(Epi = c("S", "I"))
age = mp_index(Age = c("young", "old"))
loc = mp_index(City = c("hamilton", "toronto"))
vax = mp_index(Vax = c("unvax", "vax"))
(si
 |> mp_cartesian(age)
 |> mp_cartesian(loc)
 |> mp_cartesian(vax)
)

flow_rates = mp_index(Epi = c("infection", "recovery"))
mp_union(
  mp_cartesian(
    mp_subset(flow_rates, Epi = "infection"),
    age
  ),
  mp_subset(flow_rates, Epi = "recovery")
)

```

mp_change_frame

Data Frame Describing Each Change to Each State Variable

Description

Get a data frame with one row for each change made to each state variable at each time step.

Usage

```
mp_change_frame(spec)
```

Arguments

spec Model specification ([mp_tmb_model_spec](#)).

Value

Data frame with two columns: state and change. Each row describes one change.

Examples

```
( "starter_models"
  |> mp_tmb_library("sir", package = "macpan2")
  |> mp_change_frame()
)
```

mp_default

*Default Values***Description**

Default Values

Usage

```
mp_default(model, include_all = FALSE)
```

```
mp_default_list(model, include_all = FALSE)
```

Arguments

model	A model object from which to extract default values. If model is a calibrator object (see mp_tmb_calibrator) that has been optimized (using mp_optimize), then the values returned by mp_default and mp_default_list are updated to reflect this calibration/optimization process.
include_all	Include all default variables, even those that are not used in the before, during, or after phase of the simulations. When include_all is FALSE, examples of excluded variables are those used by an objective function only or those intended to be used in an extended model specification produced using functions like mp_tmb_insert and mp_tmb_update .

Value

A long-format data frame with default values for matrices required as input to model objects. The columns of this output are matrix, row, col, and value. Scalar matrices do not have any entries in the row or col columns.

Functions

- `mp_default_list()`: List of the default variables as matrices.

mp_dynamic_model	<i>Dynamic Model</i>
------------------	----------------------

Description

This is an 'old' model specification function that was tested out at a workshop. Currently it still drives the engine-agnostic-grammar vignette, but we plan to replace this function with [mp_tmb_model_spec](#) and other model specification functions.

Usage

```
mp_dynamic_model(
  expr_list = ExprList(),
  ledgers = list(),
  init_vecs = list(),
  unstruc_mats = list()
)
```

Arguments

expr_list	Expression list.
ledgers	Ledgers.
init_vecs	Initial structured vectors.
unstruc_mats	Initial unstructured matrices.

mp_dynamic_simulator	<i>TMB Simulator from Dynamic Model</i>
----------------------	---

Description

This is an 'old' function that was tested out at a workshop. Currently it still drives the engine-agnostic-grammar vignette, but we plan to replace this function with [mp_simulator](#).

Usage

```
mp_dynamic_simulator(
  dynamic_model,
  time_steps = 0L,
  vectors = NULL,
  unstruc_mats = NULL,
  mats_to_save = NULL,
  mats_to_return = NULL,
  params = OptParamsList(),
  random = OptParamsList(),
)
```

```

    obj_fn = ObjectiveFunction(~0),
    log_file = LogFile(),
    do_pred_sdreport = TRUE,
    tmb_cpp = "macpan2",
    initialize_ad_fun = TRUE,
    ...
  )

```

Arguments

dynamic_model	Object product by <code>mp_dynamic_model</code> .
time_steps	Number of time steps to simulate.
vectors	Named list of named vectors as initial values for the simulations that are referenced in the expression list in the dynamic model.
unstruc_mats	= Named list of objects that can be coerced to numerical matrices that are used in the expression list of the dynamic model.
mats_to_save	TODO
mats_to_return	TODO
params	TODO
random	TODO
obj_fn	TODO
log_file	TODO
do_pred_sdreport	TODO
tmb_cpp	TODO
initialize_ad_fun	TODO
...	TODO

mp_effects_descr	<i>Describe Statistical Effects</i>
------------------	-------------------------------------

Description

Additional information that can be joined to the output of the `tidy.TMB` or `tidy.stanfit` functions in the `broom.mixed` package.

Usage

```

mp_effects_descr(model)

mp_add_effects_descr(coef_table, model)

```

Arguments

model	A model in the TMB engine that can be used to compute tables of statistical effects.
coef_table	Coefficient table that was probably generated using mp_tmb_coef or mp_tmbstan_coef , but also perhaps generated directly using the <code>tidy.TMB</code> or the <code>tidy.stanfit</code> methods in the <code>broom.mixed</code> package.

Functions

- `mp_add_effects_descr()`: Convenience function for adding coefficient descriptions from a calibrated model to `coef_tables` generated by [mp_tmb_coef](#) or [mp_tmbstan_coef](#).

mp_expand	<i>Expand Model</i>
-----------	---------------------

Description

Expand a structured model so that it is represented in an unstructured format requiring a more verbose description. Currently, this is only applicable for [mp_tmb_model_spec](#) objects that have explicit flows (e.g. [mp_per_capita_flow](#)). For such models, `mp_expand` produces a model with expression lists composed entirely of plain R formulas.

Usage

```
mp_expand(model)

mp_reduce(model)
```

Arguments

model	A model object.
-------	-----------------

Functions

- `mp_reduce()`: Confusingly, `mp_reduce` and `mp_expand` are synonyms. Please use `mp_expand` in new projects, as `mp_reduce` is available for back-compatibility only.

Examples

```
sir = mp_tmb_library("starter_models", "sir", package = "macpan2")
print(sir)
print(mp_expand(sir))
```

mp_extract	<i>Extract Index</i>
------------	----------------------

Description

Extract the index for a particular dimension in a ledger from a ledger or from an object containing one or more ledgers.

Usage

```
mp_extract(x, dimension_name)
```

Arguments

x Object
dimension_name Name of a dimension used in a ledger.

mp_factors	<i>Factor an Index</i>
------------	------------------------

Description

Factor an Index

Usage

```
mp_factors(index, unpack = c("no", "maybe", "yes"))
```

Arguments

index An index to be factored.
unpack Place factors in the global environment?

mp_final	<i>Final Values</i>
----------	---------------------

Description

Return the values of variables after the simulation loop has finished and the `final` set of expressions have been evaluated.

Usage

```
mp_final(model)
```

```
mp_final_list(model)
```

Arguments

`model` Object that can be used to simulate.

Functions

- `mp_final_list()`: Final values formatted as a list of matrices.

mp_flow_frame	<i>Data Frame Describing Compartmental Model Flows</i>
---------------	--

Description

Get a data frame where each row represents a flow in a model specification.

Usage

```
mp_flow_frame(spec, topological_sort = TRUE, loops = "^$")
```

Arguments

`spec` A [mp_tmb_model_spec](#).

`topological_sort`

Should the states be topologically sorted to respect the main direction of flow?

`loops`

Pattern for matching the names of flows that make the flow model not a DAG, which is a critical assumption when topologically sorting the order of states and flows in the output. This is only relevant if `topological_sort` is used.

Value

A data frame that gives information provided in calls to [mp_per_capita_flow](#) and [mp_per_capita_inflow](#).

Description

A forecaster is an object that can make forecasts. It is constructed from `mp_tmb_calibrator`, which is an object that can be used to calibrate the parameters of a model – typically by fitting the model to data. The main difference between a calibrator and a forecaster is that the latter runs for more time steps, past the last time step of the calibrator. A forecaster object can be used to generate different types of forecasts using the `mp_trajectory` family of functions.

Usage

```
mp_forecaster(
  calibrator,
  forecast_period_time_steps,
  outputs = NULL,
  data = NULL,
  tv = NULL,
  default = list(),
  inits = list()
)
```

Arguments

calibrator	Object made using <code>mp_tmb_calibrator</code> , which can be calibrated to data.
forecast_period_time_steps	The number of time steps to project beyond the period with data.
outputs	An optional character vector of variables in the model to forecast. If this argument is omitted, then the forecaster will inherit the outputs of the calibrator object. Note that if you prefix the name of an output using <code>log_</code> , <code>logit_</code> , or <code>sqrt_</code> then the transformed version of the outputs will be simulated. This technique is useful for confidence intervals produced by <code>mp_trajectory_sd</code> that go out of the valid range of values (e.g., use <code>log_</code> for confidence intervals of negative state variables).
data	An optional data frame containing the data that were fitted to. Typically this argument will be unused, because by default the data are stored in the calibrator unless you want to avoid making too many copies of the data.
tv	An optional replacement for the <code>tv</code> parameter in the <code>mp_tmb_calibrator</code> function.
default	An optional list of default model variables (e.g., parameters initial values of state variables) to override calibrated values.
inits	An optional list of initial values for the state variables. These initial values can be added to the <code>default</code> list with identical results, but adding them to <code>inits</code> is better practice because it makes it clear that they are initial values that will change as the state updates.

mp_functions_used	<i>Functions Used by an Object for Communicating with a Computational Engine</i>
-------------------	--

Description

Functions Used by an Object for Communicating with a Computational Engine

Usage

```
mp_functions_used(object)

mp_generates_randomness(object)
```

Arguments

object An object for communicating with a computational engine.

Value

Character vector of names of functions that are used by object.

Functions

- mp_generates_randomness(): Does the object use functions that generate randomness?

mp_group	<i>Group an Index</i>
----------	-----------------------

Description

Create a new index with fewer columns to create names for an aggregated vector that is labelled by the input index.

Usage

```
mp_group(index, by)
```

Arguments

index Index to group rows.
by Column set label to group by.

mp_index	<i>Model Quantity Index Table</i>
----------	-----------------------------------

Description

Make an index table to enumerate model quantity labels by category. These objects generalize and wrap `data.frames`, where each column is a label category and each row is an index. Indices must contain only letters, numbers, and underscores. Blank empty string entries are allowed, but missing values (NAs) are not.

Usage

```
mp_index(..., labelling_column_names)

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

## S3 method for class 'Index'
names(x)
```

Arguments

<code>...</code>	Character vectors to combine to produce an index. Alternatively, any number of data frames of character-valued columns. If data frames are supplied, their rows will be bound and the result converted to an index if possible.
<code>labelling_column_names</code>	A <code>character</code> vector of the names of the index that will be used to label the model components (i.e. rows) being described. The <code>labelling_column_names</code> cannot have duplicates and must contain at least one name. The index given by the <code>labelling_column_names</code> must uniquely identify each row. The default <code>NULL</code> gives the set of columns, in order starting with the first column, that are required to uniquely identify each row.
<code>x</code>	An index.

Details

Dots are not allowed in indices so that the labels can be inverted to reproduce the original index table (provided that the column names can be retrieved).

It is recommended to use UpperCamelCase for the columns of index tables and single uppercase characters ("S", "I"), all lowercase character strings ("gamma"), and/or snake_case strings ("aging_rate") for indices. This convention helps when reading code that contains references to both column names and indices.

Functions

- `print(Index)`: Print an index.
- `names(Index)`: Get the names of the columns of an index.

See Also[mp_structured_vector\(\)](#)[mp_set_numbers\(\)](#)Other functions that return index tables [mp_cartesian\(\)](#), [mp_rename\(\)](#), [mp_subset\(\)](#), [mp_union\(\)](#)**Examples**

```

state = mp_index(
  Epi = c("S", "I", "S", "I"),
  Age = c("young", "young", "old", "old")
)
print(state)
labels(state)
mp_cartesian(state, mp_index(City = c("hamilton", "toronto")))

# The following index table describes the state variables of the
# model:
sir = mp_index(Epi = c("S", "I", "R"))
print(sir)

# Here, the column `Epi` denotes that the category of these labels is
# epidemiological. There is nothing special about this specific choice of
# category name; we could have also used another name like `Compartment`.

# However, in more complicated models, it is good to think carefully about
# choosing descriptive category names. For example, in an age-structured SIR
# model, we could add an `Age` column to generate an index table as follows:
sir_age = mp_index(
  Epi = rep(c("S", "I", "R"), 2),
  Age = rep(c("young", "old"), each = 3)
)
print(sir_age)

# Here, having the first column in the index table labeled `Compartment` would
# be somewhat misleading, as the compartments aren't actually just "S", "I",
# and "R", they are each of the epidemiological states stratified by the age
# groups "young" and "old".

# This index table could also be generated by first specifying individual index
# tables for the `Epi` and `Age` columns, and then using a `macpan2` product
# function that combines the tables into a single index table:
sir = mp_index(Epi = c("S", "I", "R"))
age = mp_index(Age = c("young", "old"))
prod = mp_cartesian(sir, age)
prod

# The mp_cartesian() function will produce a table with entries that are all
# possible combinations of the individual index tables. The "See Also" section
# of the mp_cartesian() help page catalogues all available product functions.

# We can produce the full labels of model quantities, which are simply

```

```
# dot-concatenated indices, one for each entry in the index table, using the
labels(prod)
```

mp_initial	<i>Initial Values of Variables Immediately Before the Simulation Loop</i>
------------	---

Description

Return a data frame containing the values of variables at the end of the before phase, right before the simulation loop begins (i.e. right before the during phase).

Usage

```
mp_initial(model)

mp_initial_list(model)
```

Arguments

model	A model specification object or model simulator object from which to extract initial values.
-------	--

Value

A long-format data frame with initial values for matrices. The columns of this output are matrix, time, row, col, and value. Scalar matrices do not have any entries in the row or col columns. The before phase corresponds to a time value of 0.

Functions

- `mp_initial_list()`: List of the initial variables as matrices.

mp_join	<i>Join Indexes</i>
---------	---------------------

Description

Join two or more index tables (see [mp_index](#)) to produce a ledger (see [LedgerDefinition](#)).

Usage

```
mp_join(..., by = empty_named_list())
```

Arguments

- ... Named arguments giving indexes created by `mp_index` or another function that manipulates indexes. Each argument will become a position vector used to subset or expand numeric vectors in archetype formulas.
- by What columns to use to join the indexes. See below on how to specify this argument.

Details

When two index tables are passed to ..., `mp_join` behaves very much like an ordinary **inner join**. When more than two tables are passed to ..., `mp_join` iteratively joins pairs of tables to produce a final ledger. For example, if index tables A B, and C are passed to `mp_join`, an inner join of A and B is performed and the result is joined with C. In each of these successive internal joins. The properties of inner joins ensures that the order of tables does not affect the set of rows in the final table (SW states without proof!).

See Also

Other functions that return ledgers `mp_aggregate()`

Examples

```
# When two index tables are passed to `...`, the `by` argument is just a
# character vector of column names on which to join (as in standard R functions
# for joining data frames), or the dot-concatenation of these column names.
# For example,
state = mp_index(
  Epi = c("S", "I", "S", "I"),
  Age = c("young", "young", "old", "old")
)
mp_join(
  from = mp_subset(state, Epi = "S"),
  to = mp_subset(state, Epi = "I"),
  by = "Age"
)
# If there are more than two tables then the `by` argument must be a named
# list of character vectors, each describing how to join the columns of
# a pair of tables in `...`. The names of this list are dot-concatenations
# of the names of pairs of tables in `...`. For example,
rates = mp_index(
  Epi = c("lambda", "lambda"),
  Age = c("young", "old")
)
mp_join(
  from = mp_subset(state, Epi = "S"),
  to = mp_subset(state, Epi = "I"),
  rate = mp_subset(rates, Epi = "lambda"),
  by = list(
    from.to = "Age",
    from.rate = "Age"
  )
)
```

```

)
# If the `by` columns have different names in two tables, then you can
# specify these using formula notation where the left-hand-side
# is a dot-concatenation of columns in the first table and the
# right-hand-side is a dot-concatenation of the columns in the second
# table. For example,
contact = mp_index(
  AgeSusceptible = c("young", "young", "old", "old"),
  AgeInfectious = c("young", "old", "young", "old")
)
mp_join(
  sus = mp_subset(state, Epi = "S"),
  inf = mp_subset(state, Epi = "I"),
  con = contact,
  by = list(
    sus.con = "Age" ~ "AgeSusceptible",
    inf.con = "Age" ~ "AgeInfectious"
  )
)

```

mp_kronecker_operator *Kronecker Operator*

Description

Convert a scalar binary operator into one that performs a Kronecker product with more convenient row and column names for use with `macpan2`. The result is numerically identical to base R's `%x%` and `kronecker()`, but with cleaner naming of rows and columns.

Usage

```
mp_kronecker_operator(operator)
```

Arguments

`operator` A scalar binary operator.

Value

A Kronecker operator convenient for use with `macpan2`.

mp_labels

Index Labels

Description

Return a character vector of labels for each row of an index (or a ledger?? FIXME: what does this mean for ledgers??).

Usage

```
mp_labels(x, labelling_column_names)
```

Arguments

x Object

labelling_column_names

What index columns should be used for generating the labels. If missing then defaults will be used. (FIXME: clarify how the defaults are used.)

mp_layout_grid

Flow Diagram Grid Layout (experimental)

Description

Create a grid on which to layout the flow diagram of a model specification.

Usage

```
mp_layout_grid(
  spec,
  east = "",
  south = "^$",
  north = "^$",
  west = "^$",
  loops = north,
  x_gap = 0.3,
  y_gap = 0.3,
  north_south_sep = 0,
  east_west_sep = 0,
  trim_states = character()
)
```

Arguments

spec	A model specification made with <code>mp_tmb_model_spec</code> or related function.
east	Regular expression for matching the names of flows that will be connected eastward in the layout.
south	Regular expression for matching the names of flows that will be connected southward in the layout.
north	Regular expression for matching the names of flows that will be connected northward in the layout.
west	Regular expression for matching the names of flows that will be connected westward in the layout.
loops	Regular expression for matching the names of flows that cause loops in the flow model, and so should be ignored when building the layout.
x_gap	Size of the gap to the left and right of the 1-by-1 space provided for a node.
y_gap	Size of the gap above and below the 1-by-1 space provided for a node.
north_south_sep	Horizontal separation between north and south flow arrows.
east_west_sep	Vertical separation between east and west flow arrows.
trim_states	List of states to remove from the diagram

mp_layout_paths	<i>Flow Diagram Grid Layout (experimental)</i>
-----------------	--

Description

Layout the flow diagram of a model specification so that each row is one of the paths through the model (ignoring loops).

Usage

```
mp_layout_paths(
  spec,
  sort_paths = TRUE,
  combine_columns = TRUE,
  deduplicate_edges = TRUE,
  loops = "^$",
  ignore = "^$",
  x_gap = 0.3,
  y_gap = 0.3,
  north_south_sep = 0,
  east_west_sep = 0,
  trim_states = character()
)
```

Arguments

spec	A model specification made with mp_tmb_model_spec or related function.
sort_paths	Should the paths/rows be sorted to minimize the number times an edge must go through a node that it is not connected with?
combine_columns	Should each state/node get its own column in the layout (FALSE) or should the algorithm try to place branching states in the same column (TRUE, default).
deduplicate_edges	Should each row have all of the edges in the path or should duplicate edges be removed?
loops	Regular expression for matching the names of flows that cause loops in the flow model, and so should be ignored when building the layout.
ignore	Regular expression for matching the names of flows that should be removed from the layout analysis entirely. These will be isolated in a data frame for custom drawing of 'difficult' edges.
x_gap	Size of the gap to the left and right of the 1-by-1 space provided for a node.
y_gap	Size of the gap above and below the 1-by-1 space provided for a node.
north_south_sep	Horizontal separation between north and south flow arrows.
east_west_sep	Vertical separation between east and west flow arrows.
trim_states	List of states to remove from the diagram

mp_ledgers

Bundle up Ledgers

Description

Bundle up several ledgers (see [LedgerDefinition](#)) to pass to [mp_dynamic_model](#).

Usage

```
mp_ledgers(...)
```

Arguments

... Ledgers to bundle up.

mp_linear	<i>Linear Chain Product</i>
-----------	-----------------------------

Description

TODO: what does this mean?

Usage

mp_linear(x, y_labelling_column_names)

Arguments

x An index.
y_labelling_column_names
 TODO

See Also

Other functions that take products of index tables and return one index tables [mp_cartesian\(\)](#), [mp_square\(\)](#), [mp_symmetric\(\)](#), [mp_triangle\(\)](#)

mp_lookup	<i>Lookup</i>
-----------	---------------

Description

Lookup a subset or factor index associated with a symbol, and return the index associated with that symbol.

Usage

mp_lookup(index, symbol)

Arguments

index Index table (see [mp_index](#)).
symbol Character string that could possibly be associated with a subset or factor of index.

mp_model_docs

Browse Model Docs

Description

Open a browser at the current version of a particular model in an online macpan2 model library.

Usage

```
mp_model_docs(model_name, macpan_library = "starter_models")
```

Arguments

model_name Name of a model in the macpan_library.

macpan_library Name of a library. Currently, the default value of macpan_library = "starter_models" is the only recommended option.

Value

This function returns the URL of the library model, but the main purpose is the side-effect of automatically opening a web browser at this URL.

mp_model_starter

Copy Existing Model as a Starting Point

Description

Create a directory with a template model definition.

Usage

```
mp_model_starter(starter_name, dir)
```

Arguments

starter_name Currently can only be sir.

dir String giving the path to a directory for copying the template model definition.

mp_opt_attempted	<i>Optimization Attempted</i>
------------------	-------------------------------

Description

Has an attempt been made to calibrate model parameters through optimization of a likelihood function or posterior density, probably by using [mp_optimize](#)?

Usage

```
mp_opt_attempted(model)
```

Arguments

model	A model that can be calibrated, probably produced using mp_tmb_calibrator .
-------	---

Value

Either TRUE or FALSE.

mp_optimize	<i>Optimize Simulation Model</i>
-------------	----------------------------------

Description

Calibrate a model that has been parameterized, typically by using [mp_tmb_calibrator](#) to produce such a model.

Usage

```
mp_optimize(
  model,
  optimizer = c("nlminb", "optim", "DEoptim", "optimize", "optimise"),
  ...
)
```

Arguments

model	A model object capable of being optimized. Typically this object will be produced using mp_tmb_calibrator .
optimizer	Name of an implemented optimizer. See below for the options and details on using each option.
...	Arguments to pass to the optimizer.

Details

The `mp_tmb_calibrator` models that get passed to `mp_optimize` will remember both their original default parameters set at the time the model was created, and their currently best parameters that get updated when `mp_optimize` is run. The optimization is started from the currently best parameter set. Therefore, if you find yourself in a local minimum you might need to either recreate the model using `mp_tmb_calibrator` or use `optimizer = "DEoptim"`, which is more robust to objective functions with multiple optima.

Value

The output of the optimizer. The model object is modified and saves the history of optimization outputs. These outputs can be obtained using `mp_optimizer_output`.

Details on Using Each Type of Optimizer

nlminb:

The default optimizer is `nlminb`. This optimizer uses gradients computed by the **Template Model Builder** engine of `macpan2`. This optimizer is efficient at local optimization by exploiting the gradient information computed by automatic differentiation. However, like many nonlinear optimizers, it can struggle if the objective function has multiple optima.

To set control parameters (e.g., maximum number of iterations), you can use the control argument:

```
mp_optimize(model, "nlminb", control = list(iter.max = 800))
```

See the `nlminb` help page for the complete list of control parameters and what the output means.

optim:

The `optim` optimizer lets you choose from a variety of optimization algorithms. The default, `method = "Nelder-Mead"`, does not use second derivatives (compare with the description of `nlminb`), and so could be less efficient at taking each step. However, we find that it can be somewhat better at getting out of local optima, although the "DEoptim" optimizer is designed for objective functions with multiple optima (see description below).

To set control parameters (e.g., maximum number of iterations), one may use the following.

```
mp_optimize(model, "nlminb", control = list(maxit = 800))
```

See the `optim` help page for the complete list of control parameters and what the output means.

If your model is parameterized by only a single parameter, you'll get a warning asking you to use `'method = "Brent"'` or `optimizer = 'optimize()'`. You can ignore this warning if you are happy with your answer, or can do either of the suggested options as follows.

```
mp_optimize(model, "optim", method = "Brent", lower = 0, upper = 1.2)
mp_optimize(model, "optimize", interval = c(0, 1.2))
```

In this case you have to specify lower and upper bounds for the optimization.

DEoptim:

The `DEoptim` optimizer comes from the `DEoptim` package; you'll need to have that package installed to use this option. It is designed for objective functions with multiple optima. Use this

method if you do not believe the fit you get by other methods. The downsides of this method are that it doesn't use gradient information and evaluates the objective function at many different points, so is likely to be much slower than gradient-based optimizers such as the default `nlminb` optimizer, or `optim` with `method = "BFGS"`.

Because this optimizer starts from multiple points in the parameter space, you need to specify lower and upper bounds for each parameter in the parameter vector.

```
mp_optimize(model, "DEoptim", lower = c(0, 0), upper = c(1.2, 1.2))
```

In this example we have two parameters, and therefore need to specify two values each for lower and upper

`optimize` **and** `optimise`:

This optimizer can only be used for models parameterized with a single parameter. You need to specify lower and upper bounds, e.g.

```
mp_optimize(model, "optimize", c(0, 1.2))
```

Examples

```
spec = ("starter_models"
  |> mp_tmb_library("seir", package = "macpan2")
  |> mp_tmb_update(default = list(beta = 0.6))
)
sim = mp_simulator(spec, 50, "infection")

## simulate data to fit to, but remove the start of the
## simulated epidemic in order to make it more difficult
## to fit.
data = mp_trajectory(sim)
data = data[data$time > 24, ]
data$time = data$time - 24

## time scale object that accounts for the true starting time
## of the epidemic (in this example we are not trying to estimate
## the initial number infected, so the starting time strongly
## affects the fitting procedure)
time = mp_sim_offset(24, 0, "steps")

## model calibrator, estimating only the transmission parameter
cal = mp_tmb_calibrator(spec
  , data = data
  , traj = "infection"
  , par = "beta"
  , default = list(beta = 1)
  , time = time
)

## From the starting point at beta = 1 this takes us into a
## local optimum at beta = 0.13, far from the true value of beta = 0.6.
mp_optimize(cal)

## In this case, one-dimensional optimization finds the true value.
```

```
mp_optimize(cal, "optimize", c(0, 1.2))
```

mp_optimized_spec	<i>Optimized Model Specification</i>
-------------------	--------------------------------------

Description

Create a new model specification using parameter values that have been optimized.

Usage

```
mp_optimized_spec(model, spec_structure = c("original", "modified"))
```

Arguments

model	A model object that can be optimized/calibrated. Currently, only models produced by mp_tmb_calibrator are valid.
spec_structure	A character string identifying the structure of the returned model specification. Use "original" for the specification originally passed to the mp_tmb_calibrator function, and "modified" for the one that was used to fit to data. See the details below for more information on these modifications.

Details

The following is a list of additional model specification structure that was or could have been added to the "modified" specification by [mp_tmb_calibrator](#) for the purposes of fitting to data.

- Observed data in a format that can be compared with simulated data.
- Expressions preparing simulated data so that they can be compared with the observed data.
- Distributional parameters for likelihood and prior components of the objective function (e.g., a dispersion parameter for a negatively binomial likelihood component, or a prior standard deviation for a transmission rate parameter).
- Data, parameters, and expressions for modelling time-variation of parameters (e.g., basis functions and weights for a spline determining the time-variation of transmission rate).

Value

A copy of the model specification used to produce the calibrator model, with calibrated default values.

mp_optimizer_output	<i>Optimizer Output</i>
---------------------	-------------------------

Description

Get the output from an optimizer used in model calibration.

Usage

```
mp_optimizer_output(model, what = c("latest", "all"))
```

Arguments

model	An object that has been optimized.
what	A string indicating whether to return the results of the "latest" optimization attempt or a list with "all" of them.

Details

When objects created by [mp_tmb_calibrator](#) are successfully passed to [mp_optimize](#), they build up an optimization history. This history is recorded as a list of the output produced by the underlying optimizer (e.g. [nlminb](#)). This `mp_optimizer_output` function returns the latest output by default or the entire history list.

mp_par	<i>Fit Parameters</i>
--------	-----------------------

Description

Define the prior distributions for parameters and random effects to be passed to `par` argument of the [mp_tmb_calibrator](#) function.

Usage

```
mp_par(params, random)
```

Arguments

params	Named list of distributional specifications for the fixed effects.
random	Named list of distributional specifications for the random effects.

mp_parameterization	<i>Description of Model Parameterization</i>
---------------------	--

Description

Description of Model Parameterization

Usage

```
mp_parameterization(model, types = c("fixed", "random"))
```

Arguments

- model Parameterized model, probably produced using [mp_tmb_calibrator](#).
- types Vector indicating what kinds of parameters should be included, "fixed" for fixed-effect-type parameters and "random" for random-effect-type.

Examples

```
cal = si_example_object("optimized_calibrator")
mp_parameterization(cal)
```

mp_per_capita_flow	<i>Specify Flow Into, Out Of, and Between Compartments</i>
--------------------	--

Description

Specify different kinds of flows into, out of, and between compartments.

Usage

```
mp_per_capita_flow(from, to, rate, flow_name = NULL, abs_rate = NULL)
mp_per_capita_inflow(from, to, rate, flow_name = NULL, abs_rate = NULL)
mp_per_capita_outflow(from, rate, flow_name = NULL, abs_rate = NULL)
mp_inflow(to, rate, flow_name = NULL, abs_rate = NULL)
mp_outflow(from, rate, flow_name = NULL, abs_rate = NULL)
```

Arguments

from	String giving the name of the compartment from which the flow originates.
to	String giving the name of the compartment to which the flow is going.
rate	String giving the expression for the per-capita flow rate. Alternatively, and for back compatibility, a two-sided formula with the left-hand-side giving the name of the absolute flow rate per time-step and the right-hand-side giving an expression for the per-capita rate of flow from from to to.
flow_name	String giving the name of the flow
abs_rate	Deprecated synonym for flow_name. Please use flow_name in all future work.

Details

The examples below can be mixed and matched in `mp_tmb_model_spec()` to produce compartmental models. The symbols used below must be used in an appropriate context (e.g., if `N` is used for total population size, then there must be an expression like $N \sim S + I + R$ somewhere in the model or for models with constant population size there must be a default variable, `N`, with a numerical value).

Functions

- `mp_per_capita_inflow()`: Only flow into the `to` compartment, and do not flow out of the `from` compartment. The `from` compartment can even be a function of a set of compartments, because it will not be updated. A common construction is `mp_per_capita_inflow("N", "S", "birth_rate", "birth")` for adding a birth process, which involves the total population size, `N`, rather than a single compartment.
- `mp_per_capita_outflow()`: Only flow out of the `from` compartment, without going anywhere. This is useful for removing individuals from the system (e.g., death). To keep track of the total number of dead individuals one can use `mp_per_capita_flow` and set `to` to be a compartment for these individuals (e.g., `to = "D"`).
- `mp_inflow()`: Only flow into the `to` compartment. For adding a birth or immigration process.
- `mp_outflow()`: Only flow out of the `from` compartment. For adding an absolute removal process that goes to 'nowhere': dangerous! The reason it is dangerous is that this flow can easily lead to negative values of state variables when the rate is high relative to the size of the `from` compartment. Often `mp_per_capita_outflow` will be a better choice, given that the size of the outflow will be scaled to the size of the `from` compartment by measuring rates on a per-capita basis.

See Also

[mp_absolute_flow\(\)](#)

Examples

```
# infection by mass action
# https://github.com/canmod/macpan2/blob/main/inst/starter_models/si
mp_per_capita_flow("S", "I", "beta * I / N", "infection")
```

```

# recovery
# https://github.com/canmod/macpan2/blob/main/inst/starter_models/sir
mp_per_capita_flow("I", "R", "gamma", "recovery")

# disease progression with different severity
# https://github.com/canmod/macpan2/blob/main/inst/starter_models/macpan_base
mp_per_capita_flow("E", "I_mild", "alpha * phi", "progression_mild")
mp_per_capita_flow("E", "I_sev", "alpha * (1 - phi)", "progression_sev")

# birth
# https://github.com/canmod/macpan2/blob/main/inst/starter_models/sir_demog
mp_per_capita_inflow("N", "S", "nu", "birth")

# death
# https://github.com/canmod/macpan2/blob/main/inst/starter_models/sir_demog
mp_per_capita_outflow("S", "mu", "death_S")
mp_per_capita_outflow("I", "mu", "death_I")
mp_per_capita_outflow("R", "mu", "death_R")

# vaccination
# https://github.com/canmod/macpan2/blob/main/inst/starter_models/shiver
mp_per_capita_flow("S", "V", "(vrate * S / (vrate + S)) / S", "vaccination")

# importation
# mp_inflow("I", "delta", "importation")

```

mp_positions

Position Vectors

Description

Return an integer vector of positions of `x` in `table`. Currently this is a simple wrapper around [match](#).

Usage

```
mp_positions(x, table, zero_based = TRUE)
```

Arguments

<code>x</code>	Character vector
<code>table</code>	Character vector
<code>zero_based</code>	Use zero-based indexing? Defaults to TRUE, otherwise standard R one-based indexing is used.

mp_print_obj_fn	<i>Print Objective Function</i>
-----------------	---------------------------------

Description

Print Objective Function

Usage

```
mp_print_obj_fn(model)
```

Arguments

model	Model object with an objective function, probably a calibrator produced using mp_tmb_calibrator .
-------	---

Value

Called to print the objective function for humans to read. Invisibly returns the underlying objective function object that is of limited utility.

mp_print_spec	<i>Print Model Specification</i>
---------------	----------------------------------

Description

Print Model Specification

Usage

```
mp_print_spec(model)

mp_print_before(model)

mp_print_during(model)

mp_print_after(model)
```

Arguments

model	A model produced by mp_tmb_model_spec .
-------	---

Functions

- `mp_print_before()`: Print just the expressions executed before the simulation loop.
- `mp_print_during()`: Print just the expressions executed during each iteration of the simulation loop.
- `mp_print_after()`: Print just the expressions executed after the simulation loop.

mp_rbf

Fit a Time-Varying Parameter with Radial Basis Functions

Description

Pass the output of this function to the `tv` argument of [mp_tmb_calibrator](#) to model time variation of a parameter with flexible radial basis functions.

Usage

```
mp_rbf(
  tv,
  dimension,
  initial_weights,
  seed,
  prior_sd = 1,
  fit_prior_sd = TRUE,
  sparse_tol = 0.01
)
```

Arguments

<code>tv</code>	String giving the name of the parameter.
<code>dimension</code>	Number of bases.
<code>initial_weights</code>	Optional vector with dimensions elements. These are the parameters that are fitted and determine how <code>tv</code> varies with time.
<code>seed</code>	Optional random seed to use to generate the <code>initial_weights</code> if they are not provided.
<code>prior_sd</code>	Prior standard deviation default value for radial basis function coefficients, defaults to 1.
<code>fit_prior_sd</code>	Should the prior sd be fitted.
<code>sparse_tol</code>	Tolerance below which radial basis function outputs are set exactly to zero. Small values are more accurate but slower. Lack of accuracy can be visually apparent as jumps in graphs of the time-varying parameter.

mp_read_rds	<i>Read Serialized Model Specification</i>
-------------	--

Description

Uses [readRDS](#) to read in a saved model specification created using a function like [mp_tmb_model_spec](#), and updates this specification using [mp_version_update](#) so that it is compatible with the installed version of macpan2. To save a model specification, just use the base R function [saveRDS](#).

Usage

```
mp_read_rds(filename)
```

Arguments

filename	Path to a saved model specification object.
----------	---

mp_reference	<i>Reference Index</i>
--------------	------------------------

Description

Extract the index used as a reference for generating position vectors.

Usage

```
mp_reference(x, dimension_name)
```

Arguments

x	Object
dimension_name	Name of a dimension used in a ledger if applicable.

mp_rename	<i>Rename Index Columns</i>
-----------	-----------------------------

Description

Rename Index Columns

Usage

```
mp_rename(x, ...)
```

Arguments

x	An index with columns to be renamed.
...	Name-value pairs. The name gives the new name and the value is a character vector giving the old name.

See Also

Other functions that return index tables [mp_cartesian\(\)](#), [mp_index\(\)](#), [mp_subset\(\)](#), [mp_union\(\)](#)

mp_show_models	<i>Print a Table of Existing Models</i>
----------------	---

Description

Collects information from the headers of the README files in model directories and returns the results as a data frame

Usage

```
mp_show_models(
  dir = system.file("starter_models", package = "macpan2"),
  show_missing = FALSE,
  for_markdown = FALSE
)

show_models(
  dir = system.file("starter_models", package = "macpan2"),
  show_missing = FALSE,
  for_markdown = FALSE
)

mp_list_models(dir = system.file("starter_models", package = "macpan2"))
```

Arguments

dir	directory to list
show_missing	(logical) include entries for models with no README information?
for_markdown	(logical) format for rendering the table with markdown-formatted links to model readme files?

Value

a data frame containing entries Directory (model directory), Title (model title), Description (short description)

Functions

- `show_models()`: Synonym for `mp_show_models`, which is preferred. Present for back-compatibility.
- `mp_list_models()`: Return a character vector containing model names, instead of a data frame with more information about each model.

Examples

```
mp_show_models(show_missing = TRUE)
```

mp_sim_bounds	<i>Simulation Bounds</i>
---------------	--------------------------

Description

Set the simulation bounds (start time and end time) for a calibration. This is used to override the default simulation bounds taken from the observed data passed to `mp_tmb_calibrator`.

Usage

```
mp_sim_bounds(sim_start, sim_end, time_scale = "steps", time_column = "time")
```

Arguments

sim_start	Start time of each simulation.
sim_end	End time of each simulation.
time_scale	Qualitative description of the size of a time step. The only valid values are 'steps' and 'daily'. If you would like each time step in the model to represent one day, then you should consider using 'daily'. Note that using 'daily' will require that your data represent time using a (1) Date vector, (2) character vector in YYYY-MM-DD format, or (3) integer vector that counts the number of days since some reference. Otherwise please choose 'steps', the default, and convert your time column into integer values that represent the time step that you would like in the model.
time_column	Name of the column that will identify the time at which particular values were observed.

Value

An object to be passed to the time argument of `mp_tmb_calibrator`.

See Also

`mp_sim_offset()`

mp_sim_offset

Simulation Offsets

Description

Offset the starting and ending times of the simulation, from the start and end time of the data used in calibration. This is used to override the default offsets of zero taken from the observed data passed to `mp_tmb_calibrator`.

Usage

```
mp_sim_offset(
  sim_start_offset,
  sim_end_offset,
  time_scale = "steps",
  time_column = "time"
)
```

Arguments

sim_start_offset	Number of time steps before the first data point to start each simulation.
sim_end_offset	Number of time steps after the last data point to end each simulation.
time_scale	Qualitative description of the size of a time step. The only valid values are 'steps' and 'daily'. If you would like each time step in the model to represent one day, then you should consider using 'daily'. Note that using 'daily' will require that your data represent time using a (1) Date vector, (2) character vector in YYYY-MM-DD format, or (3) integer vector that counts the number of days since some reference. Otherwise please choose 'steps', the default, and convert your time column into integer values that represent the time step that you would like in the model.
time_column	Name of the column that will identify the time at which particular values were observed.

Value

An object to be passed to the time argument of `mp_tmb_calibrator`.

See Also

`mp_sim_bounds()`

`mp_simulator`*Create a Simulator*

Description

Construct a simulator from a model specification object.

Usage

```
mp_simulator(model, time_steps, outputs, default = list(), inits = list())
```

Arguments

<code>model</code>	A model specification object.
<code>time_steps</code>	How many time steps should be simulated when simulations are requested?
<code>outputs</code>	Character vector of names of model quantities that will be outputted when simulations are requested.
<code>default</code>	Named list of numerical objects that will update the default values defined in the model specification object. Any number of objects can be updated or not.
<code>inits</code>	An optional list of initial values for the state variables. These initial values can be added to the default list with identical results, but adding them to <code>inits</code> is better practice because it makes it clear that they are initial values that will change as the state updates.

`mp_slices`*Slice an index*

Description

Slice an index

Usage

```
mp_slices(index, unpack = c("no", "maybe", "yes"))
```

Arguments

<code>index</code>	Index to slice up.
<code>unpack</code>	Place factors in the global environment?

mp_square	<i>Self Cartesian Product</i>
-----------	-------------------------------

Description

Self Cartesian Product

Usage

```
mp_square(x, suffixes = c("A", "B"))
```

Arguments

x	An index.
suffixes	Length-2 character vector giving suffixes that disambiguate the column names in the output.

See Also

Other functions that take products of index tables and return one index tables [mp_cartesian\(\)](#), [mp_linear\(\)](#), [mp_symmetric\(\)](#), [mp_triangle\(\)](#)

mp_state_dependence_frame	<i>Data Frame Describing State Dependent Per-Capita Flow Rates</i>
---------------------------	--

Description

Data frame giving states that per-capita flow rates directly depend on. This is intended for plotting diagrams and not for mathematical analysis, in that it does not describe indirect dependence for flow rates on state variables.

Usage

```
mp_state_dependence_frame(spec)
```

Arguments

spec	Model specification from spec .
------	---

mp_structured_vector	<i>Structured Vectors</i>
----------------------	---------------------------

Description

This documentation was originally in `mp_index()` and should be cleaned up See issue #131. Also this is an experimental feature.

Usage

```
mp_structured_vector(x, ...)
```

```
mp_set_numbers(vector, ...)
```

Arguments

x	An index.
...	Passed on to S3 methods.
vector	An index.

Functions

- `mp_set_numbers()`: Update numerical values of a structured vector. TODO: details on syntax.

Examples

```
state = mp_index(  
  Epi = c("S", "I", "S", "I"),  
  Age = c("young", "young", "old", "old")  
)  
state_vector = (state  
  |> mp_structured_vector()  
  |> mp_set_numbers(Epi = c(S = 1000))  
  |> mp_set_numbers(Epi = c(I = 1), Age = "old")  
)  
print(state_vector)
```

<code>mp_subset</code>	<i>Subset of Indexes</i>
------------------------	--------------------------

Description

Take a subset of the rows of an index table (see [mp_index](#)) to produce another index table. The `mp_subset` function gives rows that match a certain criterion and `mp_setdiff` gives rows that do not match.

Usage

```
mp_subset(x, ...)

mp_setdiff(x, ...)
```

Arguments

- `x` Model index.
- `...` Name-value pairs. The names are columns (or sets of columns using dot-concatenation) in `x` and the values are character vectors that refer to labels with respect to those columns. These values determine the resulting subset.

See Also

Other functions that return index tables [mp_cartesian\(\)](#), [mp_index\(\)](#), [mp_rename\(\)](#), [mp_union\(\)](#)

<code>mp_symmetric</code>	<i>Symmetric Self Cartesian Product</i>
---------------------------	---

Description

Symmetric Self Cartesian Product

Usage

```
mp_symmetric(x, y_labelling_column_names, exclude_diag = TRUE)
```

Arguments

- `x` An index.
- `y_labelling_column_names` TODO
- `exclude_diag` Should 'diagonal' components be excluded from the output.

See Also

Other functions that take products of index tables and return one index tables [mp_cartesian\(\)](#), [mp_linear\(\)](#), [mp_square\(\)](#), [mp_triangle\(\)](#)

mp_time_scale	<i>Time Scale</i>
---------------	-------------------

Description

Time Scale

Usage

```
mp_time_scale(start, end, time_step_scale = c("steps", "daily", "weekly"), ...)
```

Arguments

- | | |
|-----------------|---|
| start | First date or time in the first time step |
| end | Last date or time in the last time step |
| time_step_scale | TODO |
| ... | TODO |

mp_tmb	<i>Get Underlying TMB Object</i>
--------	----------------------------------

Description

Get the result of TMB: :MakeADFun underlying a TMB-based model in macpan2.

Usage

```
mp_tmb(model)
```

Arguments

- | | |
|-------|-------------------------|
| model | An object based on TMB. |
|-------|-------------------------|

mp_tmb_calibrator	<i>Make a Calibrator</i>
-------------------	--------------------------

Description

Construct an object that can get used to calibrate an object produced by [mp_tmb_model_spec](#) or [mp_tmb_library](#), and possibly modified by [mp_tmb_insert](#) or [mp_tmb_update](#). Note that the output of this function is not a model that has been calibrated to data, but rather an object that contains a model that can be calibrated. To actually attempt a calibration one needs the [mp_optimize](#) function (see examples below).

Usage

```
mp_tmb_calibrator(
  spec,
  data = empty_trajectory,
  traj = character(),
  par = character(),
  tv = character(),
  outputs = traj,
  default = list(),
  inits = list(),
  time = NULL,
  save_data = TRUE,
  optimize = FALSE
)
```

Arguments

spec	An TMB model spec to fit to data. Such specs can be produced by mp_tmb_model_spec or mp_tmb_library , and possibly modified with mp_tmb_insert and mp_tmb_update .
data	A data frame containing trajectories to fit to and possibly time-varying parameters. The data must be of the same format as that produced by mp_trajectory .
traj	A character vector giving the names of trajectories to fit to data, or a named list of likelihood distributions specified with a distribution for each trajectory. Transformations of trajectories cannot be named implicitly unless they are also transformed through the outputs argument and their transformed version appears in the data.
par	A character vector giving the names of parameters (either time-varying or not) to fit using trajectory match, or a named list of prior distributions specified with a distribution for each parameter. Parameters can be implicitly fitted on a transformed scale by prefixing the name of the parameter with the name of the transformation (e.g., <code>log_beta</code> will fit beta on the log-transformed scale, and mp_tmb_coef will report estimates on the original scale by default. The mp_tmb_implicit_backtrans function is used internally. See that help page for available transformations.

tv	A character vector giving the names of parameters to make time-varying according to the values in data, or a radial basis function specified with <code>mp_rbf</code> .
outputs	A character vector of outputs that will be generated when <code>mp_trajectory</code> , <code>mp_trajectory_sd</code> , or <code>mp_trajectory_ensemble</code> are called on the optimized calibrator. By default it is just the trajectories listed in <code>traj</code> . Outputs can be implicitly transformed by prefixing the name of the output with the name of the transformation (e.g., <code>log_infection</code> will output <code>log(infection)</code> , but <code>mp_trajectory_sd</code> will report infection and its confidence interval on the original scale). The <code>mp_tmb_implicit_trans</code> function is used internally. See that help page for available transformations.
default	A list of default values to use to update the defaults in the spec. By default nothing is updated. Alternatively one could use <code>mp_tmb_update</code> to update the spec outside of the function. Indeed such an approach is necessary if new expressions, in addition to default updates, need to be added to the spec (e.g. seasonally varying transmission).
inits	An optional list of initial values for the state variables. These initial values can be added to the default list with identical results, but adding them to <code>inits</code> is better practice because it makes it clear that they are initial values that will change as the state updates.
time	Specify the start and end time of the simulated trajectories, and the time period associated with each time step. The default is <code>NULL</code> , which takes simulation bounds from the data. You can use <code>mp_sim_bounds</code> and <code>mp_sim_offset</code> to be more specific. See the example on the <code>mp_optimize</code> help file for an illustration the use of <code>mp_sim_offset</code> .
save_data	Save a copy of the data in the calibrator object that is returned, so that you do not need to pass the data manually to downstream functions like <code>mp_forecaster</code> . If the resulting calibrator object is so large that it causes you problems, consider not saving the data in the calibrator object and manually passing it to the data argument of <code>mp_forecaster</code> .
optimize	Call <code>mp_optimize</code> on the resulting calibrator object, before returning it. The default is <code>FALSE</code> so that you can control when you would like to spend time optimizing.

Examples

```
spec = mp_tmb_library("starter_models", "sir", package = "macpan2")
sim = mp_simulator(spec, 50, "infection")
data = mp_trajectory(sim)
cal = mp_tmb_calibrator(
  spec
  , data
  , traj = "infection"
  , par = "beta"
  , default = list(beta = 0.25)
)
mp_optimize(cal)
if (suppressPackageStartupMessages(require(broom.mixed))) {
  print(mp_tmb_coef(cal))
}
```

mp_tmb_coef

TMB Model Coefficient Table

Description

TMB Model Coefficient Table

Usage

```
mp_tmb_coef(model, back_transform = TRUE, ...)
```

Arguments

model	Object that contains information about model coefficients.
back_transform	A boolean to indicate if model coefficients should be back transformed to display their defaults, estimates, and confidence intervals on the original scale. Coefficient names are also stripped of their transformation identifier. Currently, this back transformation only applies to log transformed coefficients that have been named with "log_" prefix or logit transformed coefficients that have been named with "logit_" prefix. Back transformation also applies to time varying parameters and distributional parameters that get automatic prefixes when used. back_transform defaults to TRUE.
...	Arguments to pass onto the broom.mixed::tidy.TMB method. To get confidence intervals, use conf.int = TRUE. Note that there is currently an issue when using effects = "random."

Value

A data frame that describes the fitted coefficients.

mp_tmb_expr_list

Expression List

Description

Create a list of expressions for defining a compartmental model in TMB.

Usage

```
mp_tmb_expr_list(
  before = list(),
  during = list(),
  after = list(),
  .simulate_exprs = character(0L)
)
```

Arguments

- before** List of formulas to be evaluated in the order provided before the simulation loop begins. Each **formula** must have a left hand side that gives the name of the matrix being updated, and a right hand side giving an expression containing only the names of matrices in the model, functions defined in `macpan2.cpp`, and numerical literals (e.g. `3.14`). The available functions are described in [engine_functions](#). Names can be provided for the components of **before**, and these names do not have to be unique. These names are used by the `.simulate_exprs` argument.
- during** List of formulas to be evaluated at every iteration of the simulation loop, with the same rules as **before**.
- after** List of formulas to be evaluated after the simulation loop, with the same rules as **before**.
- .simulate_exprs** Character vector of names of expressions to be evaluated within TMB simulate blocks. This is useful when an expression cannot be evaluated during the computation of the objective function and its gradients (e.g. if the expression contains randomness or other discontinuities that will break the automatic differentiation machinery of TMB).

Value

Object of class `ExprList` with the following methods.

Methods:

- `$data_arg(...)`: Return the following components of the data structure to pass to C++.
 - `expr_output_id` – Indices into the list of matrices identifying the matrix being produced.
 - `expr_sim_block` – Identified whether or not the expression should be evaluated inside a `simulate` macro within TMB.
 - `expr_num_p_table_rows` – Number of rows associated with each expression in the parse table (`p_table_*`)
 - `eval_schedule` – Vector giving the number of expressions to evaluate in each phase (before, during, or after) of the simulation.
 - `p_table_x` – Parse table column giving an index for looking up either function, matrix, or literal.
 - `p_table_n` – Parse table column giving the number of arguments in functions.
 - `p_table_i` – Parse table column giving indices for looking up the rows in the parse table corresponding with the first argument of the function.

Method Arguments:

- `...`: Character vector containing the names of the matrices in the model.

mp_tmb_fixef_cov	<i>Covariance of Fixed Effect Estimates</i>
------------------	---

Description

Covariance of Fixed Effect Estimates

Usage

```
mp_tmb_fixef_cov(model)
```

Arguments

model	Object that contains information about fitted model parameters.
-------	---

Value

A covariance matrix.

mp_tmb_insert	<i>Transform a TMB Model Specification</i>
---------------	--

Description

Insert, update, or delete elements of a TMB model spec, produced using [mp_tmb_library](#) or [mp_tmb_model_spec](#). The only difference between `mp_tmb_insert` and `mp_tmb_update` is that the former shifts the positions of existing expressions to make room for the new expressions, whereas the latter overwrites existing expressions using the new expressions. The treatment of new default values and integers is the same. The examples below clarify this difference. Note that `mp_tmb_delete` does not contain an `expressions` argument, because it is not necessary to specify new expressions in the case of deletion.

Usage

```
mp_tmb_insert(
  model,
  phase = "during",
  at = 1L,
  expressions = list(),
  default = list(),
  inits = list(),
  integers = list(),
  must_save = character(),
  must_not_save = character(),
  sim_exprs = character()
)
```

```

mp_tmb_update(
  model,
  phase = "during",
  at = 1L,
  expressions = list(),
  default = list(),
  inits = list(),
  integers = list(),
  must_save = character(),
  must_not_save = character(),
  sim_exprs = character()
)

mp_tmb_delete(
  model,
  phase,
  at,
  default = character(),
  integers = character(),
  must_save = character(),
  must_not_save = character(),
  sim_exprs = character()
)

```

Arguments

model	TMB model spec object produced using mp_tmb_library or mp_tmb_model_spec .
phase	At what phase should expressions be inserted, updated, or deleted.
at	Expression number, which can be identified by printing out model, at which the expressions should be inserted or updated. If inserted then the existing expressions with number at and higher are shifted after the new expressions are added. If updated, then the existing expressions with number from at to at + length(expressions) - 1 are replaced with the new expressions. For <code>mp_tmb_delete</code> , a numeric vector of integers identifying expressions to delete from the model.
expressions	Expressions to insert into the model spec or to replace existing expressions.
default	Named list of objects, each of which can be coerced into a numeric matrix . The names refer to variables that appear in before, during, and after. For <code>mp_tmb_delete</code> , a character vector of such objects to delete from the model.
inits	An optional list of initial values for the state variables. These initial values can be added to the default list with identical results, but adding them to inits is better practice because it makes it clear that they are initial values that will change as the state updates.
integers	Named list of vectors that can be coerced to integer vectors. These integer vectors can be used by name in model formulas to provide indexing of matrices and as grouping factors in group_sums . For <code>mp_tmb_delete</code> , a character vector of such objects to delete from the model.

<code>must_save</code>	Character vector of the names of variables that must have their values stored at every iteration of the simulation loop. For example, a variable that you do not want to be returned, but that impacts dynamics with a time lag, must be saved and therefore must be in this list.
<code>must_not_save</code>	Character vector of the names of variables that must not have their values stored at every iteration of the simulation loop. For example, the user may ask to return a very large matrix that would create performance issues if stored at each iteration. The creator of the model can mark such variables making it impossible for the user of the model to save their full simulation history.
<code>sim_exprs</code>	Character vector of the names of before, during, and after expressions that must only be evaluated when simulations are being produced and not when the objective function is being evaluated. For example, expressions that generate stochasticity should be listed in <code>sim_exprs</code> because TMB objective functions must be continuous.

Details

These modifications do not update the model specification in-place. Instead the output of `mp_tmb_insert`, `mp_tmb_update`, and `mp_tmb_delete` define a new model specification and should be saved if you want to use the new model (e.g., `new_model = mp_tmb_insert(model, ...)`).

Value

A new model spec object with updated and/or inserted information.

Examples

```
si = mp_tmb_library("starter_models", "si", package = "macpan2")
print(si)

## Update the mixing process to include
## optional phenomenological heterogeneity.
## We need mp_tmb_update here so that
## the previous infection expression is
## overwritten.
mp_tmb_update(si, phase = "during"
, at = 1
, expressions = list(infection ~ beta * I * (S/N)^zeta)
, default = list(zeta = 1)
)

## Parameterize with log_beta in place of beta.
## We need mp_tmb_insert here so that the
## existing expression for computing the initial
## number of susceptible individuals is not
## overwritten.
mp_tmb_insert(si, phase = "before"
, at = 1
, expressions = list(beta ~ exp(log_beta))
, default = list(log_beta = log(0.5))
)
```

`mp_tmb_insert_backtrans`*Insert Back Transformations of Model Parameters*

Description

Insert Back Transformations of Model Parameters

Usage

```
mp_tmb_insert_backtrans(  
  model,  
  variables = character(),  
  transformation = mp_log  
)  
  
mp_tmb_implicit_backtrans(model, variables = character())
```

Arguments

<code>model</code>	TMB model spec object produced using mp_tmb_library or mp_tmb_model_spec .
<code>variables</code>	Character vector of parameters to back transform.
<code>transformation</code>	A transformation object such as mp_log , which is the default. See the help page for mp_log for available options.

Value

A new model specification object with expressions for the untransformed (or back transformed) parameters at the beginning of the before phase. The transformed version of the parameter is also added to the defaults and are identified with a prefixed name (e.g., `log_beta` if `beta` is log transformed).

Functions

- `mp_tmb_implicit_backtrans()`: Insert parameter transformations implicitly by pre-pending the name of the transformations in front of the names of the variables that you ask for (e.g., `"log_case_reports"`) even if this variable is not in the model as long as the base name (e.g., `"case_reports"`) is.

See Also

[mp_tmb_insert_trans\(\)](#)

Examples

```

init_si = ("starter_models"
  |> mp_tmb_library("si", package = "macpan2")
  |> mp_tmb_insert_backtrans("beta", mp_log)
  |> mp_initial_list()
)
print(init_si$log_beta)
print(log(init_si$beta))

```

mp_tmb_insert_glm_timevar

Insert GLM Time Variation

Description

Insert GLM Time Variation

Usage

```

mp_tmb_insert_glm_timevar(
  model,
  parameter_name,
  design_matrix,
  timevar_coef,
  link_function = mp_log,
  matrix_coef_name = sprintf("matrix_coef_%s", parameter_name),
  matrix_row_name = sprintf("matrix_row_%s", parameter_name),
  matrix_col_name = sprintf("matrix_col_%s", parameter_name),
  linear_pred_name = sprintf("linear_pred_%s", parameter_name),
  timeseries_name = sprintf("timeseries_%s", parameter_name),
  timevar_coef_name = sprintf("time_var_%s", parameter_name),
  time_index_name = sprintf("time_index_%s", parameter_name),
  sparsity_tolerance = 0,
  engine_function = c("sparse_mat_mult", "group_sums")
)

```

Arguments

model	A model specification (see mp_tmb_model_spec).
parameter_name	Character string giving the name of the parameter to make time-varying.
design_matrix	Matrix of time variation.
timevar_coef	Initial coefficient matrix with the same number of rows as there are columns of the design_matrix.
link_function	Link function given by functions like mp_log .

matrix_coef_name	Name of the vector containing values of the non-zero elements of the design matrix.
matrix_row_name	Name of the vector containing row indices of the non-zero elements of the design matrix.
matrix_col_name	Name of the vector containing column indices of the non-zero elements of the design matrix.
linear_pred_name	Name of the vector containing the linear predictor.
timeseries_name	Name of the vector containing the time-series of the varying parameter.
timevar_coef_name	Name of the vector containing the time-varying parameter coefficients.
time_index_name	Name of the index at which the time varying parameter changes.
sparsity_tolerance	Make design matrix coefficients exactly zero when they are below this tolerance.
engine_function	Which of two engine_functions , sparse_mat_mult or group_sums , should be used to compute the matrix multiplication. The only reason to use anything other than the default is for back-compatibility with an existing script.

mp_tmb_insert_log_linear

Insert Log Linear Model of Time Variation (Experimental)

Description

Insert Log Linear Model of Time Variation (Experimental)

Usage

```
mp_tmb_insert_log_linear(
  model,
  parameter_name,
  design_matrices,
  time_var_parameters,
  window_names = names(time_var_parameters),
  baseline_functions = c(list(TimeVarBaselineParameter()),
    rep(list(TimeVarBaselineNumeric(0)), length(design_matrices) - 1)),
  link_functions = rep(list(mp_identity), length(design_matrices)),
  full_series_name = sprintf("time_var_output_%s", parameter_name),
  baseline_names = sprintf("baseline_%s", window_names),
```

```

matrix_coef_names = sprintf("matrix_coef_%s", window_names),
matrix_row_names = sprintf("matrix_row_%s", window_names),
matrix_col_names = sprintf("matrix_col_%s", window_names),
linear_pred_names = sprintf("linear_pred_%s", window_names),
time_var_names = sprintf("time_var_%s", window_names),
time_index_name = sprintf("time_index_%s", parameter_name),
sparsity_tolerance = 0
)

```

Arguments

model	A model specification (see mp_tmb_model_spec).
parameter_name	Character string giving the name of the parameter to make time-varying.
design_matrices	List of matrices, one for each time window, describing the model of time variation.
time_var_parameters	Named list of parameter vectors for each window, with names giving the window names.
window_names	Names for each window.
baseline_functions	It is complicated – this is a joke – I’m tired.
link_functions	List of objects representing link functions.
full_series_name	Name of variable storing the full time series.
baseline_names	Names of variables containing the baseline in each window.
matrix_coef_names	Names of vectors containing values of the non-zero elements of the design matrices.
matrix_row_names	Names of the vectors containing row indices of the non-zero elements of the design matrices.
matrix_col_names	Names of the vectors containing column indices of the non-zero elements of the design matrices.
linear_pred_names	Names of the vectors containing the linear predictors in each window.
time_var_names	Names of the time-varying parameter in each window.
time_index_name	Name of the index at which the time varying parameter changes.
sparsity_tolerance	Make design matrix coefficients exactly zero when they are below this tolerance.

mp_tmb_insert_reports *Transform a TMB Model Specification to Account for Reporting Bias*

Description

A version of `mp_tmb_insert` making it more convenient to transform an incidence variable into a reports variable, which accounts for reporting delays and under-reporting. This new reports variable is a convolution of the simulation history of an incidence variable with a kernel that is proportional to a Gamma distribution of reporting delay times.

Usage

```
mp_tmb_insert_reports(
  model,
  incidence_name,
  report_prob,
  mean_delay,
  cv_delay,
  reports_name = sprintf("reported_%s", incidence_name),
  report_prob_name = sprintf("%s_report_prob", incidence_name),
  mean_delay_name = sprintf("%s_mean_delay", incidence_name),
  cv_delay_name = sprintf("%s_cv_delay", incidence_name)
)
```

Arguments

<code>model</code>	A model produced by <code>mp_tmb_model_spec</code> .
<code>incidence_name</code>	Name of the incidence variable to be transformed.
<code>report_prob</code>	Value to use for the reporting probability; the proportion of cases that get reported.
<code>mean_delay</code>	Mean of the Gamma distribution of reporting delay times.
<code>cv_delay</code>	Coefficient of variation of the Gamma distribution of reporting delay times.
<code>reports_name</code>	Name of the new reports variable.
<code>report_prob_name</code>	Name of the variable containing <code>report_prob</code> .
<code>mean_delay_name</code>	Name of the variable containing <code>mean_delay</code> .
<code>cv_delay_name</code>	Name of the variable containing <code>cv_delay</code> .

mp_tmb_insert_trans *Insert Basic Transformations of Model Variables*

Description

Insert Basic Transformations of Model Variables

Usage

```
mp_tmb_insert_trans(model, variables = character(), transformation = mp_log)
```

```
mp_tmb_implicit_trans(model, variables = character())
```

Arguments

model	TMB model spec object produced using mp_tmb_library or mp_tmb_model_spec .
variables	Character vector of variables to transform.
transformation	A transformation object such as mp_log , which is the default. See the help page for mp_log for available options.

Value

A new model specification object with expressions for the transformed variables at the end of the simulation loop. The transformed variables are identified with a prefixed name (e.g., log_incidence if incidence is log transformed).

Functions

- `mp_tmb_implicit_trans()`: Insert variable transformations implicitly by pre-pending the name of the transformations in front of the names of the variables that you ask for (e.g., "log_case_reports") even if this variable is not in the model as long as the base name (e.g., "case_reports") is.

See Also

[mp_tmb_insert_backtrans\(\)](#)

Examples

```
("starter_models"
  |> mp_tmb_library("si", package = "macpan2")
  |> mp_tmb_insert_trans("infection", mp_log)
  |> mp_simulator(time_steps = 5L, outputs = "log_infection")
  |> mp_trajectory()
)
```

mp_tmb_library	<i>Read Item from a Model Library</i>
----------------	---------------------------------------

Description

Get a TMB model specification from a model library.

Usage

```
mp_tmb_library(..., package = NULL, alternative_specs = FALSE)
```

```
mp_tmb_entire_library()
```

```
mp_official_library(model_name)
```

Arguments

...	File path components pointing to a directory that contains an R script that creates an object called spec, which is produced by mp_tmb_model_spec .
package	If NULL, file.path is used to put together the ... components but if package is the name of a package (as a character string) then system.file is used to put together the ... components.
alternative_specs	If TRUE, return a list of alternative specification objects. For models without alternatives this will cause the return value to be a list with one element containing a spec object.
model_name	Character string giving the name of a single model in the official starter_models library .

Details

This function executes the R code in the tmb.R file of a directory in a model library. To be a valid model, this file should produce an object called spec (containing a model specification produced by the [mp_tmb_model_spec\(\)](#) function) or specs (containing a list of such specifications). This [mp_tmb_library\(\)](#) function returns this spec or specs object (see the [alternative_specs](#) argument), but does not expose any other object produced by tmb.R.

Functions

- [mp_tmb_entire_library\(\)](#): List of model specifications; one for each model in the library.
- [mp_official_library\(\)](#): Get a model specification from the official [macpan2 starter_models library](#).

See Also

[mp_show_models\(\)](#)

Examples

```
mp_tmb_library(
  "starter_models"
, "si"
, package = "macpan2"
)
```

mp_tmb_model_spec	Create TMB Model Specification
-------------------	--------------------------------

Description

Specify a simulation model in the TMB engine. A detailed explanation of this function is covered in `vignette("quickstart")`.

Usage

```
mp_tmb_model_spec(
  before = list(),
  during = list(),
  after = list(),
  default = list(),
  inits = list(),
  integers = list(),
  must_save = character(),
  must_not_save = character(),
  sim_exprs = character(),
  state_update = c("euler", "rk4", "discrete_stoch", "hazard", "rk4_old",
    "euler_multinomial")
)
```

Arguments

before	List of formulas to be evaluated (in the order provided) before the simulation loop begins. These formulas must be standard two-sided R formula objects. See details below for the rules for these formulas.
during	List of formulas or calls to flow functions (e.g., <code>mp_per_capita_flow</code>) to be evaluated at every iteration of the simulation loop.
after	List of formulas to be evaluated (in the order provided) before the simulation loop begins. These formulas must be standard two-sided R formula objects. See details below for the rules for these formulas.
default	Named list of objects, each of which can be coerced into a numeric matrix . The names refer to quantities that appear in before, during, and after. Each quantity that is used in before, during, or after before it is defined must be given a numerical value in this default list.

<code>inits</code>	Named list of initial values of state variables. These initial values can be added to the default list with identical results, but adding them to <code>inits</code> is better practice because it makes it clear that they are initial values that will change as the state updates.
<code>integers</code>	Named list of vectors that can be coerced to integer vectors. These integer vectors can be used by name in model formulas to provide indexing of matrices and as grouping factors in group_sums .
<code>must_save</code>	Character vector of the names of variables that must have their values stored at every iteration of the simulation loop. For example, a variable that you do not want to be returned, but that impacts dynamics with a time lag, must be saved and therefore must be in this list.
<code>must_not_save</code>	Character vector of the names of variables that must not have their values stored at every iteration of the simulation loop. For example, the user may ask to return a very large matrix that would create performance issues if stored at each iteration. The creator of the model can mark such variables making it impossible for the user of the model to save their full simulation history.
<code>sim_exprs</code>	Character vector of the names of before, during, and after expressions that must only be evaluated when simulations are being produced and not when the objective function is being evaluated. For example, expressions that generate stochasticity should be listed in <code>sim_exprs</code> because TMB objective functions must be continuous.
<code>state_update</code>	Optional character vector for how to update the state variables when it is relevant. Options include "euler" (the default), "rk4", and "discrete_stoch".

Details

Expressions in the before, during, and after lists can be standard R [formula](#) objects for defining variables in the model. These formulas must have a left hand side that gives the name of the (possibly matrix-valued) variable being updated, and a right hand side giving an expression containing only (1) the names of quantities in the model, (2) numerical literals (e.g., 3.14), or (3) functions defined in the TMB engine (described in [engine_functions](#)). For example, the expression `N ~ S + I + R` updates the value of `N` to be the sum of the variables `S`, `I`, and `R`.

Names can be provided for the components of the before, during, and after lists, and these names do not have to be unique. These names are used by the `sim_exprs` argument.

Examples

```
## A simple SI model.
spec = mp_tmb_model_spec(
  during = mp_per_capita_flow("S", "I", "beta * I / N", "infection")
  , default = list(N = 100, S = 99, I = 1, beta = 0.2)
)
(spec
 |> mp_simulator(time_steps = 5L, output = "infection")
 |> mp_trajectory()
)

## The `` can be used for flexibly defining dynamical variables.
```

```

spec2 = mp_tmb_model_spec(
  during = list(
    force_of_infection ~ beta * I / N
    , mp_per_capita_flow("S", "I", "force_of_infection", "infection")
  )
  , default = list(N = 100, S = 99, I = 1, beta = 0.2)
)
(spec2
  |> mp_simulator(time_steps = 5L, output = "force_of_infection")
  |> mp_trajectory()
)

## The `before` argument can be used to pre-compute quantities before
## the simulation loop begins. Here we compute `S` from `N` and `I`,
## instead of specifying `S` in the `default` list. The potential
## benefit here is that one could make `I` a parameter to be fitted,
## while ensuring consistent values for `S`.
spec3 = mp_tmb_model_spec(
  before = S ~ N - I
  , during = mp_per_capita_flow("S", "I", "beta * I / N", "infection")
  , default = list(N = 100, I = 1, beta = 0.2)
)
(spec3
  |> mp_simulator(time_steps = 5L, output = "infection")
  |> mp_trajectory()
)

```

mp_tmb_objective

Value of the Objective Function of a Model

Description

Value of the Objective Function of a Model

Usage

```

mp_tmb_objective(
  model,
  parameter_updates = list(),
  baseline = c("recommended", "default", "optimized")
)

```

Arguments

model	A model with an objective function, probably one produced using mp_tmb_calibrator .
parameter_updates	Named list of a subset of model variables with the values to use when simulating the trajectory using the <code>mp_trajectory_par</code> function. In the future we plan to

allow this variable to be a data frame with one row for each scalar value (which would be useful if only certain elements of a vector or matrix are parameters) and a string giving the name of a file containing parameter information. But for now, only a list is allowed.

baseline Models can contain several alternative sets of parameters, and this baseline argument is used to choose which of these should be updated using the `parameter_updates` passed to `mp_trajectory_par`. The current options are "recommended", "optimized", and "default". The "recommended" option will be used if neither of the other two options are selected. If model is capable of being optimized (e.g., it was created using `mp_tmb_calibrator`) then "recommended" is equivalent to "optimized", which use the best set of parameters found by `mp_optimize`. If `mp_optimize` has not yet been called on model then a warning will be issued. If model is not capable of being optimized then "recommended" is equivalent to "default", which uses the original set of parameters available when model was created.

mp_tmb_profile	<i>TMB Likelihood Profiling</i>
----------------	---------------------------------

Description

Use `TMB::tmbprofile` to compute the profile likelihood of a calibrator produced using `mp_tmb_calibrator`.

Usage

```
mp_tmb_profile(model, param, ...)
```

Arguments

<code>model</code>	A TMB model probably produced using <code>mp_tmb_calibrator</code> .
<code>param</code>	The name of a fixed effect parameter set through the <code>par</code> argument of the call to <code>mp_tmb_calibrator</code> used to create <code>model</code> . You can find the names of these parameters after the model is created using the <code>mp_parameterization</code> function.
<code>...</code>	Arguments to pass to <code>TMB::tmbprofile</code> .

Value

The output of `TMB::tmbprofile`.

mp_tmbstan_coef	<i>Model Coefficient Table with stan</i>
-----------------	--

Description

Leverages the `tmbstan` and `broom.mixed` packages to generate MCMC-based coefficient tables.

Usage

```
mp_tmbstan_coef(model, tmbstan_args = list(), ...)
```

Arguments

<code>model</code>	Object that contains information about model coefficients.
<code>tmbstan_args</code>	Arguments to pass on to <code>tmbstan</code> , which is used to generate an <code>rstan</code> object from the underlying TMB object.
<code>...</code>	Arguments to pass onto the <code>broom.mixed::tidy.stanfit</code> method.

mp_traj	<i>Trajectory Specification</i>
---------	---------------------------------

Description

Specify a set of trajectories to fit. The output of this function is intended to be passed to the `traj` argument of [mp_tmb_calibrator](#).

Usage

```
mp_traj(likelihood = empty_named_list(), condensation = empty_named_list())
```

Arguments

<code>likelihood</code>	List of likelihood components. The names of the list identify the trajectory associated with each likelihood component.
<code>condensation</code>	List of condensation methods. The names of the list identify the trajectories produced by each condensation method.

mp_trajectory *Simulate Dynamical Model Trajectories*

Description

Return simulations of the trajectory of the output variables of a dynamical model simulator. To see this functionality in context, please see `vignette("quickstart")`.

Usage

```
mp_trajectory(model, include_initial = FALSE)

mp_trajectory_par(
  model,
  parameter_updates = list(),
  include_initial = FALSE,
  include_final = FALSE,
  baseline = c("recommended", "default", "optimized")
)

mp_trajectory_sd(
  model,
  conf.int = FALSE,
  conf.level = 0.95,
  include_initial = FALSE,
  back_transform = TRUE
)

mp_trajectory_ensemble(model, n, probs = c(0.025, 0.975))

mp_trajectory_sim(model, n, probs = c(0.025, 0.25, 0.5, 0.75, 0.975))

mp_trajectory_replicate(
  model,
  n,
  parameter_updates = list(),
  include_initial = FALSE,
  include_final = FALSE,
  baseline = c("recommended", "default", "optimized")
)
```

Arguments

<code>model</code>	A dynamical model simulator produced by <code>mp_simulator</code> .
<code>include_initial</code>	Should the initial values of the simulation be included in the output? If TRUE this will include outputs for <code>time == 0</code> associated with the initial values. See

[mp_initial](#) for another approach to getting the initial values.

<code>parameter_updates</code>	Named list of a subset of model variables with the values to use when simulating the trajectory using the <code>mp_trajectory_par</code> function. In the future we plan to allow this variable to be a data frame with one row for each scalar value (which would be useful if only certain elements of a vector or matrix are parameters) and a string giving the name of a file containing parameter information. But for now, only a list is allowed.
<code>include_final</code>	Should the final values of the simulation, after the post-simulation processing steps in the after stage of a model, be included in the output? If TRUE this will include outputs for <code>time == time_steps + 1</code> , associated with the values of the variables after the full trajectory has been post-processed in the after stage. See mp_final for another approach to getting the final values.
<code>baseline</code>	Models can contain several alternative sets of parameters, and this baseline argument is used to choose which of these should be updated using the <code>parameter_updates</code> passed to <code>mp_trajectory_par</code> . The current options are "recommended", "optimized", and "default". The "recommended" option will be used if neither of the other two options are selected. If model is capable of being optimized (e.g., it was created using mp_tmb_calibrator) then "recommended" is equivalent to "optimized", which use the best set of parameters found by mp_optimize . If mp_optimize has not yet been called on model then a warning will be issued. If model is not capable of being optimized then "recommended" is equivalent to "default", which uses the original set of parameters available when model was created.
<code>conf.int</code>	Should confidence intervals be produced?
<code>conf.level</code>	If <code>conf.int</code> is TRUE, what confidence level should be used? For example, the default of 0.95 corresponds to 95% confidence intervals.
<code>back_transform</code>	A boolean to indicate if trajectories, standard deviations, and confidence intervals should be back transformed to the original scale. Variable names are also stripped of their transformation identifier. Currently, this back transformation only applies to log transformed coefficients that have been named with "log_" prefix or logit transformed coefficients that have been named with "logit_" prefix. Back transformation also applies to time varying parameters and distributional parameters that get automatic prefixes when used. <code>back_transform</code> defaults to TRUE.
<code>n</code>	Number of random trajectories to simulate.
<code>probs</code>	Numeric vector of probabilities corresponding to quantiles for summarizing the results over the random realizations.

Value

A data frame with one row for each simulated value and the following columns.

matrix Name of the variable in the model. All variables are matrix-valued in `macpan2` (scalars are technically 1-by-1 matrices), which explains the name of this field. In hindsight I would have called it `variable`.

time Time index of the simulated value, with `time = 0` indicating initial values.

- row** The 0-based index of the row of the matrix, or the name of the row of the matrix if row names (or names for column vectors) are supplied for the default value of the matrix.
- col** The 0-based index of the column of the matrix, or the name of the column of the matrix if column names are supplied for the default value of the matrix. It is also possible that this column is blank if everything is either a scalar or column vector (a common case).
- value** (mp_trajectory and mp_trajectory_sd) Simulation values.
- sd** (for mp_trajectory_sd only) The standard deviations of the simulated values accounting for parameter estimation uncertainty.
- conf.low** (for mp_trajectory_sd only) The lower bounds of the confidence interval for the simulated values.
- conf.high** (for mp_trajectory_sd only) The upper bounds of the confidence interval for the simulated values.
- n%** (for mp_trajectory_[ensemble|sim]) The n-th quantiles of the simulation values over repeated simulations.

Functions

- `mp_trajectory_par()`: Produce a trajectory, after updating the baseline set of parameters with values in `parameter_updates`.
- `mp_trajectory_sd()`: Simulate a trajectory that includes uncertainty information provided by the `sreport` function in TMB with default settings.
- `mp_trajectory_ensemble()`: Simulate a trajectory that includes uncertainty information provided by repeatedly sampling from a normal approximation to the distribution of the fitted parameters, and generating one trajectory for each of these samples. The quantiles of the empirical distribution of these trajectories can be used to produce a confidence interval for the fitted trajectory.
- `mp_trajectory_sim()`: Generate quantiles over `n` realizations of the trajectory. Instead of a value column in the output data frame, there is one column for each of the quantiles defined in `probs`.
- `mp_trajectory_replicate()`: Generate a list of `n` simulation results.

Examples

```
spec = mp_tmb_library("starter_models"
  , "si"
  , package = "macpan2"
)
simulator = mp_simulator(spec
  , time_steps = 10L
  , outputs = c("infection", "I")
)
trajectory = mp_trajectory(simulator)
print(trajectory)
```

mp_triangle	<i>Self Cartesian Product Excluding One Off-Diagonal Side</i>
-------------	---

Description

Self Cartesian Product Excluding One Off-Diagonal Side

Usage

```
mp_triangle(
    x,
    y_labelling_column_names,
    exclude_diag = TRUE,
    lower_tri = FALSE
)
```

Arguments

x	An index.
y_labelling_column_names	TODO
exclude_diag	Should 'diagonal' components be excluded from the output.
lower_tri	Should the lower triangular components be include from the output. If FALSE the result is upper triangular.

See Also

Other functions that take products of index tables and return one index tables [mp_cartesian\(\)](#), [mp_linear\(\)](#), [mp_square\(\)](#), [mp_symmetric\(\)](#)

mp_uncertainty_estimated	<i>Uncertainty Estimated</i>
--------------------------	------------------------------

Description

Does a model contain estimates of parameter uncertainty, probably through a covariance matrix estimated using [mp_optimize](#)?

Usage

```
mp_uncertainty_estimated(model)
```

Arguments

model	A model that can be calibrated, probably produced using mp_tmb_calibrator .
-------	---

Value

Either TRUE or FALSE.

mp_union	<i>Union of Indexes</i>
----------	-------------------------

Description

Union of Indexes

Usage

```
mp_union(...)
```

Arguments

... Indexes.

See Also

Other functions that return index tables [mp_cartesian\(\)](#), [mp_index\(\)](#), [mp_rename\(\)](#), [mp_subset\(\)](#)

mp_vars	<i>Dynamic Variable Names</i>
---------	-------------------------------

Description

Get the state, flow, and other dynamic variables in a model specification. Dynamic variables are any variable that is updated every time step. State and flow variables are special dynamic variables involved in compartmental models that have been explicitly represented using functions like [mp_per_capita_flow](#) that define flows among compartments (i.e., states).

Usage

```
mp_state_vars(spec, topological_sort = FALSE, loops = "^$", trans = "")
```

```
mp_flow_vars(spec, topological_sort = FALSE, loops = "^$", trans = "")
```

```
mp_state_flow_vars(spec, topological_sort = FALSE, loops = "^$", trans = "")
```

```
mp_dynamic_vars(spec)
```

```
mp_other_dynamic_vars(spec)
```

Arguments

spec	Model specification (mp_tmb_model_spec).
topological_sort	Should the states and flows be topologically sorted to respect the main direction of flow? The default is no topological sorting, which differs from mp_flow_frame .
loops	Pattern for matching the names of flows that make the flow model not a DAG, which is a critical assumption when topologically sorting the order of states and flows in the output. This is only relevant if <code>topological_sort</code> is used.
trans	Add a prefix to the names for indicating if a transformed version of the variables is preferred.

Details

State and flow variables will be identical regardless of the state update method (e.g., [mp_rk4](#)), but other dynamic variables might appear for one particular state update method that does not appear for another. For example, the first Runge Kutta step for a state in a model with an [mp_rk4](#) updater, will not appear with an [mp_euler](#) updater.

Value

Character vector of names of all requested variables.

Functions

- `mp_state_vars()`: Return character vector of all state variables.
- `mp_flow_vars()`: Return the names of all variables that contain the absolute flow between compartments. The absolute flow is the magnitude of a flow per time step.
- `mp_state_flow_vars()`: Union of `mp_state_vars()` and `mp_flow_vars()`.
- `mp_dynamic_vars()`: All variables that are updated once per time-step.
- `mp_other_dynamic_vars()`: All variables that are updated once per time-step, excluding those that are state and flow variables.

Examples

```
si = mp_tmb_library("starter_models", "si", package = "macpan2")
(si
  |> mp_simulator(time_steps = 5L, mp_state_vars(si))
  |> mp_trajectory()
)
```

mp_version	<i>Version of macpan2</i>
------------	---------------------------

Description

Version of macpan2

Usage

```
mp_version(model)
```

Arguments

model	macpan2 model object.
-------	-----------------------

Value

Object of type [package_version](#) giving the version of macpan2 that produced model.

mp_version_update	<i>Version Update</i>
-------------------	-----------------------

Description

Update a model specification so that it is compatible with the installed version of macpan2.

Usage

```
mp_version_update(spec)
```

Arguments

spec	Object produced by mp_tmb_model_spec or another function that produces the same type of object.
------	---

mp_zero_vector	<i>Zero Vector</i>
----------------	--------------------

Description

Create a **numeric** vector of all zeros with names given by x

Usage

```
mp_zero_vector(x, ...)
```

Arguments

x	Object representing the names of the output vector. Most commonly this will be a character vector.
...	Passed on to S3 methods.

names_and_labels	<i>Names and Labels</i>
------------------	-------------------------

Description

This page describes functions for giving names and labels to entities in structured models.

Usage

```
to_labels(x)
to_names(x)
to_name(x)
to_name_pairs(x)
to_values(x)
```

Arguments

x	Object from which to extract its name, names, labels, name-pairs, or values. Not all types of objects will work for all functions.
---	--

Value

Character vector (or numeric vector in the case of `to_values`) that describes x.

Functions

- `to_labels()`: Extract a vector for describing the rows of a data frame or values of a numeric vector.
- `to_names()`: Extract a character vector for describing the character-valued columns in a data frame or the flattened structure of a numeric vector. Names obey the following restrictions: (1) they cannot have dots, (2) all values must start with a letter, (3) all characters must be letters, numbers, or underscore.
- `to_name()`: Extract a string (i.e. length-1 character vector) for describing the character-valued columns in a data frame or the flattened structure of a numeric vector. The name of an object is the dot-concatenation of its names.
- `to_name_pairs()`: A character vector with all possible pairwise dot-concatenations of a set of names.
- `to_values()`: Extract the `numeric` column from a data frame with only a single numerical column. This data frame might have more than one column, but only one of them can be numeric. This function will also turn numeric `matrix` and `array` objects with `dimnames` into a flattened numeric vector with labels produced by appropriately dot-concatenating the `dimnames`.

Context

A goal of `macpan2` is to provide a mechanism for representing structured compartmental models. An example of such a model is to have each compartment in an SEIR model split into a set of spatial locations and into a set of age groups. It is crucial but difficult to assign meaningful and consistent names to the compartments, flow rates, transmission rates, contact rates, sub-population sizes, and other parameters determining these quantities. Such names should convey how the different quantities relate to one another. For example, the names should make clear that the rate of flow between two compartments is specific to, say, the age group and location of those compartments. The naming system should facilitate identifying model quantities and sets of quantities. For example, in a spatially structured model we might want to refer to all states in a particular location (e.g. Toronto) and a specific state within that location (e.g. susceptible individuals in Toronto).

Model entities (e.g. states, flow rates, transmission rates), can be described using a data frame of string-valued columns. The rows of these data frames represent the entities being represented. The columns of the data frame represent different ways to describe the rows.

```
EpiSympVax = data.frame(
  Epi = c(rep(c("S", "E", "I", "I", "R", "beta"), 2), "alpha", "gamma", "gamma", "infectiousness", "infe"),
  Symp = c(rep(c("", "", "mild", "severe", "", ""), 2), "", "mild", "severe", "mild", "severe", ""),
  Vax = c(rep(c("unvax", "vax"), each = 6), "", "", "", "", "", "dose_rate")
)
EpiSympVax
#>           Epi    Symp      Vax
#> 1           S              unvax
#> 2           E              unvax
#> 3           I    mild      unvax
#> 4           I severe      unvax
#> 5           R              unvax
#> 6         beta              unvax
```

```

#> 7          S          vax
#> 8          E          vax
#> 9          I   mild    vax
#> 10         I severe    vax
#> 11         R          vax
#> 12        beta          vax
#> 13        alpha
#> 14        gamma   mild
#> 15        gamma severe
#> 16 infectiousness mild
#> 17 infectiousness severe
#> 18                                dose_rate

```

Non-empty values in each cell must contain only letters, numbers, underscores, and must start with a letter. Empty values are zero-length strings that can be used to indicate that some partitions are not applicable to some variables. The purpose for these restrictions is to facilitate the construction of strings and character vectors that summarize different aspects of the data frame. When taken together, these summaries can be inverted to restore the full labelled partition and so they represent zero information loss. This equivalence allows us to go back-and-forth between the two representations without losing information, but perhaps gaining convenience.

There are three types of summaries: the names, the name, and the labels. The names of a data frame are the names of the string-valued columns.

```

to_names(EpiSympVax)
#> [1] "Epi" "Symp" "Vax"

```

The name of a data frame is the dot-concatenation of the names.

```

to_name(EpiSympVax)
#> [1] "Epi.Symp.Vax"

```

The labels of a data frame is the row-wise dot-concatenation of the string-valued columns.

```

to_labels(EpiSympVax)
#> [1] "S..unvax"          "E..unvax"          "I.mild.unvax"
#> [4] "I.severe.unvax"    "R..unvax"          "beta..unvax"
#> [7] "S..vax"            "E..vax"            "I.mild.vax"
#> [10] "I.severe.vax"      "R..vax"            "beta..vax"
#> [13] "alpha.."           "gamma.mild."        "gamma.severe."
#> [16] "infectiousness.mild." "infectiousness.severe." "..dose_rate"

```

These labels give a unique single character string for referring to each variable. With only the labels and one of either the names or the name, one may recover the labelled partition. The labels provide convenient names for the variables – i.e. rownames. By convention we use **UpperCamelCase** for partition names and a modified form of **snake_case** for variable labels. Our modification of snake case allows for single uppercase letters in order to accommodate the convention in epidemiology for using single uppercase letters to refer to state variables. For example, S, I, and R, as well as I_mild and I_severe, would be consistent with our modified snake case style.

nlist	<i>Self Naming List</i>
-------	-------------------------

Description

Self Naming List

Usage

```
nlist(...)
```

Arguments

... Objects to put into the list

rbf	<i>Radial Basis Functions</i>
-----	-------------------------------

Description

Compute a set of radial basis functions (dimension of them).

Usage

```
rbf(time_steps, dimension, scale = time_steps/dimension)
```

Arguments

time_steps	number of time steps in the model
dimension	number of gaussians in the basis
scale	width of the gaussians

Examples

```
matplot(rbf(100, 5), type = "l")
```

Reader	<i>Reader</i>
Description	
Construct objects for reading data.	
Usage	
Reader(...)	
CSVReader(...)	
JSONReader(...)	
TXTReader(...)	
RReader(...)	
NULLReader(...)	
Arguments	
...	Character vectors giving path components to the file to be read.
Functions	
• CSVReader(): Read CSV files. • JSONReader(): Read JSON files. • TXTReader(): Read TXT files. • RReader(): Read R files. • NULLReader(): Placeholder reader that always returns NULL.	
si_example	<i>SI Example</i>

Description

The `si_example_object` function efficiently generates objects associated with the **example SI model**. It either creates these objects or extracts them from a pre-computed cache if they have already been created in the current session. This function is predominantly used to simplify examples in the package documentation – it allows package developers to efficiently get these basic example objects to illustrate many different concepts, without having to spend time computing them over and over again when generating the documentation. The `si_example_code` function displays code that could be used to generate these objects, and clarifies what these functions do.

Usage

```

si_example_object(
  object = c("specification", "simulator", "data", "calibrator", "optimized_calibrator")
)

si_example_code(
  object = c("specification", "simulator", "data", "calibrator", "optimized_calibrator")
)

```

Arguments

object Type of object associated with the example SI model. Can be one of the following: "specification", "simulator", "data", "calibrator", "optimized_calibrator". The functions used to produce each of these objects, respectively, are [mp_tmb_library](#), [mp_simulator](#), [mp_trajectory](#), [mp_tmb_calibrator](#), and [mp_tmb_calibrator](#).

Functions

- `si_example_object()`: Return an object associated with the example SI model.
- `si_example_code()`: Show code that could be used to generate an object associated with the example SI model.

Examples

```

# Get SI calibrator in two different states
cal1 = si_example_object("calibrator")
cal2 = si_example_object("optimized_calibrator")

# Optimize cal1 so that it is equal to cal2
mp_optimize(cal1)
mp_tmb_coef(cal1)
mp_tmb_coef(cal2)

# Inspect the code used to produce all objects in this example SI model.
si_example_code("specification")
si_example_code("simulator")
si_example_code("data")
si_example_code("calibrator")
si_example_code("optimized_calibrator")

```

Description

Simple Iterated Simulation

Usage

```
simple_sims(iteration_exprs, time_steps, int_vecs = list(), mats = list())
```

Arguments

iteration_exprs	List of expressions to pass to the engine . The expressions are only allowed to use valid engine_functions . Each expression is evaluated in order, once for each iteration. The number of iterations is controlled by the time_steps argument.
time_steps	Number of time steps to iterate.
int_vecs	Named list of integer vectors.
mats	Named list of matrices.

Value

A data frame with the simulation results.

sparse_matrix_notation

Extract Sparse Matrix Notation from a Dense Matrix

Description

Converts a dense matrix to a sparse representation by extracting its non-zero entries and their indices. Entries with absolute value less than or equal to tol are treated as zeros.

Usage

```
sparse_matrix_notation(M, zero_based = TRUE, tol = 1e-04)
```

Arguments

M	A numeric matrix or object coercible to a matrix.
zero_based	Logical; if TRUE (default), the returned row and column indices are zero-based (starting at 0). If FALSE, indices are one-based (as in standard R matrices).
tol	Numeric tolerance (default 1e-4). Entries with absolute value less than or equal to tol are treated as zero.

Value

A named list with components:

row_index Integer vector of row indices for non-zero entries (adjusted by zero_based).

col_index Integer vector of column indices for non-zero entries (adjusted by zero_based).

values Numeric vector of the non-zero entries.

M The original input matrix, coerced to a dense matrix.

Msparse A copy of M with near-zero entries (as determined by tol) replaced by exact zeros.

Examples

```
M <- matrix(c(5, 0, 0,
              0, 0, 3,
              0, 2, 0), nrow = 3, byrow = TRUE)
sparse_matrix_notation(M)
```

state_updates

Change How State Variables are Updated

Description

These functions return a modified version of a model specification, such that the state variables are updated each time step according to different numerical methods.

Usage

```
mp_euler(model)

mp_rk4(model)

mp_rk4_old(model)

mp_euler_multinomial(model)

mp_discrete_stoch(model)

mp_hazard(model)
```

Arguments

`model` Object with quantities that have been explicitly marked as state variables.

Details

By choosing one of these functions, one is able to convert

To see the computations that update the state variables under these modified specifications, one may use the [mp_expand](#) function (see examples).

The default update method for model specifications produced using [mp_tmb_model_spec](#) is `mp_euler`. This update method yields a difference-equation model where the state is updated once per time-step using the absolute flow rate as the difference between steps.

Functions

- `mp_euler()`: ODE solver using the Euler method, which is equivalent to treating the model as a set of discrete-time difference equations. This is the default method used by `mp_tmb_model_spec`, but this default can be changed using the functions described below.
- `mp_rk4()`: ODE solver using Runge-Kutta 4. Any formulas that appear before model flows in the during list will only be updated with RK4 if they do contain functions in `getOption("macpan2_non_iterable_fun")` and if they do not make any state variable assignments (i.e., the left-hand-side does not contain state variables). Each formula that does not meet these conditions will be evaluated only once at each time-step before the other three RK4 iterations are taken. By default, the `time_var` function and functions that generate random numbers (e.g., `rbinom`) are not iterable. Functions that generate random numbers will only be called once with state update methods that do not repeat expressions more than once per time-step (e.g., `mp_euler`), and so repeating these functions with RK4 could make it difficult to compare methods. If you really do want to regenerate random numbers at each RK4 iteration, you can do so by setting the above option appropriately. The `time_var` function assumes that it will only be called once per time-step, and so it should never be removed from the list of non-iterable functions. Although in principle it could make sense to update state variables manually, it currently causes us to be confused. We therefore require that all state variable updates are set explicitly (e.g., with `mp_per_capita_flow`).
- `mp_rk4_old()`: Old version of `mp_rk4` that doesn't keep track of absolute flows through each time-step. As a result this version is more efficient but makes it more difficult to compute things like incidence over a time scale.
- `mp_euler_multinomial()`: Original and deprecated name for `mp_discrete_stoch`. In all new projects please use `mp_discrete_stoch`.
- `mp_discrete_stoch()`: Update state such that the probability of moving from box *i* to box *j* in one time step is given by $(1 - \exp(-\sum(r_{ij}))) * (r_{ij} / r_i)$, where r_{ij} is the per-capita rate of flow from box *i* to box *j*, and r_i is the sum of all r_{ij} for a particular *i*. These probabilities from box *i* are used together in a multinomial distribution that determines how many individuals go to each *j* box and how many stay in *i*.
- `mp_hazard()`: Update state with hazard steps, which is equivalent to taking the step given by the expected value of the Euler-multinomial distribution.

Examples

```
sir = mp_tmb_library("starter_models", "sir", package = "macpan2")
sir
sir |> mp_euler() |> mp_expand()
sir |> mp_rk4() |> mp_expand()
sir |> mp_discrete_stoch() |> mp_expand()
```

Description

Create objects for representing names and labels in a dynamical model.

Usage

```
StringDataFromFrame(data)

StringDataFromDotted(labels, name)

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

Arguments

data	Data frame with names given by column names and labels by the elements of the columns.
labels	Character vector with (dot-separated) partition labels.
name	Character scalar with (dot-separated) partition name.
x	StringData object
...	Not used but present for S3 method consistency.

Methods (by generic)

- `print(StringData)`: Print out a StringData object.

Functions

- `StringDataFromFrame()`: Construct object from a data frame without any dots in either the names or the values.
- `StringDataFromDotted()`: Construct object from a character scalar with (dot-separated) partition names and a character vector with (dot-separated) partition labels.

Examples

```
vars = (mp_cartesian(
  mp_index(Epi = c("S", "I", "R"))
  , mp_index(Age = c("young", "old"))
)
|> as.data.frame()
|> StringDataFromFrame()
)
vars
vars$dot()
```

to_positions

To Positions

Description

Return position vector of indices corresponding to the input object.

Usage

```
to_positions(x)
```

Arguments

x An object of a class that can be converted to a position vector.

See Also

[mp_positions\(\)](#)

to_string*To String*

Description

Convert an object to a string.

Usage

```
to_string(x)
```

Arguments

x Object to convert to a string.

Value

A length-1 character vector.

Transform

Transform

Description

Transform

Usage

```
Transform(variable, default = NULL, trans_variable = variable)
```

```
Identity(variable, default = NULL, trans_variable = variable)
```

```
Log(variable, default = NULL, trans_variable = sprintf("log_%s", variable))
```

```
Logit(
  variable,
  default = NULL,
  trans_variable = sprintf("logit_%s", variable)
)
```

Arguments

variable Character string giving the name of a variable in the model.

default Default value for the untransformed variable. If NULL (the default) this value is taken from the initial value in the model containing the transformation.

trans_variable Character string to use as the name of the transformed version of the variable.

Functions

- Identity(): Identity transformation.
- Log(): Log transformation.
- Logit(): Logit transformation.

transform_distr_param *Link Functions and Transformation*

Description

- mp_identity - Identity transformation
- mp_log - Log transformation
- mp_log1p - Log1p transformation (i.e., $\log(1 + x)$)
- mp_logit - Logit transformation
- mp_sqrt - Square-root transformation

Usage`mp_identity``mp_log``mp_log1p``mp_logit``mp_sqrt`

Index

- * **create-model-calibrator-args**
 - mp_par, [53](#)
 - mp_rbf, [58](#)
 - mp_traj, [86](#)
- * **create-model-calibrator**
 - mp_tmb_calibrator, [68](#)
- * **create-model-simulator**
 - mp_simulator, [63](#)
- * **create-model-spec**
 - mp_model_starter, [48](#)
 - mp_tmb_library, [81](#)
 - mp_tmb_model_spec, [82](#)
- * **datasets**
 - empty_matrix, [9](#)
 - empty_trajectory, [10](#)
 - transform_distr_param, [105](#)
- * **distributional-assumptions**
 - distribution, [6](#)
 - fit_distr_params, [26](#)
- * **indexes**
 - mp_cartesian, [29](#)
 - mp_index, [39](#)
 - mp_rename, [60](#)
 - mp_subset, [66](#)
 - mp_union, [91](#)
- * **ledgers**
 - mp_aggregate, [29](#)
 - mp_join, [41](#)
- * **population-structure**
 - LedgerDefinition, [27](#)
 - mp_cartesian, [29](#)
 - mp_extract, [35](#)
 - mp_index, [39](#)
 - mp_join, [41](#)
 - mp_linear, [47](#)
 - mp_positions, [56](#)
 - mp_reference, [59](#)
 - mp_rename, [60](#)
 - mp_square, [64](#)
 - mp_symmetric, [66](#)
 - mp_triangle, [90](#)
- * **products**
 - mp_cartesian, [29](#)
 - mp_linear, [47](#)
 - mp_square, [64](#)
 - mp_symmetric, [66](#)
 - mp_triangle, [90](#)
- * **transform-model-spec**
 - mp_tmb_insert, [72](#)
 - mp_tmb_insert_reports, [79](#)
- (, [15](#)
- :, [14](#)
- ‘*’ (engine_functions), [11](#)
- ‘+’ (engine_functions), [11](#)
- ‘-’ (engine_functions), [11](#)
- ‘/’ (engine_functions), [11](#)
- ‘:’ (engine_functions), [11](#)
- ‘[’ (engine_functions), [11](#)
- ‘%*%’ (engine_functions), [11](#)
- ‘%x%’ (engine_functions), [11](#)
- ‘^’ (engine_functions), [11](#)
- all_consistent (comparison), [5](#)
- all_equal (comparison), [5](#)
- all_not_equal (comparison), [5](#)
- array, [95](#)
- assign, [24](#)
- assign (engine_functions), [11](#)
- BinaryOperator, [4](#)
- block (engine_functions), [11](#)
- c (engine_functions), [11](#)
- cbind, [16](#)
- cbind (engine_functions), [11](#)
- cbind_lag (engine_functions), [11](#)
- cbind_time (engine_functions), [11](#)
- character, [5](#), [39](#), [61](#), [62](#), [94](#)
- check_finite (engine_functions), [11](#)

- clamp (engine_functions), 11
- col_sums (engine_functions), 11
- colSums, 17
- comparison, 5
- convolution (engine_functions), 11
- cos (engine_functions), 11
- CSVReader (Reader), 98
- cumsum (engine_functions), 11
- data.frame, 39
- Date, 61, 62
- dbinom (engine_functions), 11
- dimnames, 95
- distribution, 6, 68
- divide_safe (engine_functions), 11
- dnbinom (engine_functions), 11
- dnorm (engine_functions), 11
- dot_layout, 8
- dpois (engine_functions), 11
- empty_matrix, 9, 23, 24
- empty_trajectory, 10
- engine_eval, 10, 11
- engine_functions, 10, 11, 71, 77, 83, 100
- exp (engine_functions), 11
- file.path, 81
- find_all_paths, 25
- fit_distr_params, 8, 26
- formula, 71, 82, 83
- from_diag (engine_functions), 11
- graphAM, 8, 9
- group_sums, 73, 77, 83
- group_sums (engine_functions), 11
- Identity (Transform), 105
- integer, 61, 62
- invlogit (engine_functions), 11
- JSONReader (Reader), 98
- last (engine_functions), 11
- layoutGraph, 8
- LedgerDefinition, 27, 29, 41, 46
- Log (Transform), 105
- log (engine_functions), 11
- Logit (Transform), 105
- logit (engine_functions), 11
- match, 56
- matrix, 9, 16, 28, 73, 82, 95
- matrix (engine_functions), 11
- mean (engine_functions), 11
- mp_absolute_flow, 28
- mp_absolute_flow(), 55
- mp_add_effects_descr
(mp_effects_descr), 33
- mp_adjacency, 28
- mp_aggregate, 29, 42
- mp_binary_operator (BinaryOperator), 4
- mp_cartesian, 29, 40, 47, 60, 64, 66, 67, 90, 91
- mp_change_frame, 30
- mp_default, 31
- mp_default_list (mp_default), 31
- mp_discrete_stoch (state_updates), 101
- mp_dot_layout (dot_layout), 8
- mp_dynamic_model, 27, 32, 33, 46
- mp_dynamic_simulator, 32
- mp_dynamic_vars (mp_vars), 91
- mp_effects_descr, 33
- mp_euler, 92, 102
- mp_euler (state_updates), 101
- mp_euler_multinomial (state_updates), 101
- mp_expand, 34, 101
- mp_extract, 35
- mp_factors, 35
- mp_final, 36, 88
- mp_final_list (mp_final), 36
- mp_fit (fit_distr_params), 26
- mp_flow_frame, 25, 36, 92
- mp_flow_vars (mp_vars), 91
- mp_forecaster, 37, 69
- mp_functions_used, 38
- mp_generates_randomness
(mp_functions_used), 38
- mp_group, 38
- mp_hazard (state_updates), 101
- mp_identity (transform_distr_param), 105
- mp_index, 29, 39, 41, 42, 47, 60, 66, 91
- mp_index(), 65
- mp_inflow (mp_per_capita_flow), 54
- mp_initial, 41, 88
- mp_initial_list (mp_initial), 41
- mp_join, 27, 29, 41
- mp_kronecker_operator, 43

mp_labels, 44
 mp_layout_grid, 44
 mp_layout_paths, 45
 mp_ledgers, 46
 mp_linear, 29, 47, 64, 67, 90
 mp_list_models (mp_show_models), 60
 mp_lnorm (distribution), 6
 mp_log, 26, 75, 76, 80
 mp_log (transform_distr_param), 105
 mp_loglp (transform_distr_param), 105
 mp_log_normal (distribution), 6
 mp_logit (transform_distr_param), 105
 mp_logit_normal (distribution), 6
 mp_logitnorm (distribution), 6
 mp_lookup, 47
 mp_model_docs, 48
 mp_model_starter, 48
 mp_nbinom (distribution), 6
 mp_neg_bin (distribution), 6
 mp_nofit (fit_distr_params), 26
 mp_norm (distribution), 6
 mp_normal, 26
 mp_normal (distribution), 6
 mp_official_library (mp_tmb_library), 81
 mp_opt_attempted, 49
 mp_optimize, 31, 49, 49, 53, 68, 69, 85, 88, 90
 mp_optimized_spec, 52
 mp_optimizer_output, 50, 53
 mp_other_dynamic_vars (mp_vars), 91
 mp_outflow (mp_per_capita_flow), 54
 mp_par, 53
 mp_parameterization, 54, 85
 mp_per_capita_flow, 28, 34, 36, 54, 82, 91, 102
 mp_per_capita_flow(), 28
 mp_per_capita_inflow, 8, 28, 36
 mp_per_capita_inflow
 (mp_per_capita_flow), 54
 mp_per_capita_outflow, 8, 28
 mp_per_capita_outflow
 (mp_per_capita_flow), 54
 mp_pois (distribution), 6
 mp_poisson (distribution), 6
 mp_positions, 56
 mp_positions(), 104
 mp_print_after (mp_print_spec), 57
 mp_print_before (mp_print_spec), 57
 mp_print_during (mp_print_spec), 57
 mp_print_obj_fn, 57
 mp_print_spec, 57
 mp_rbf, 58, 69
 mp_read_rds, 59
 mp_reduce (mp_expand), 34
 mp_reference, 59
 mp_rename, 29, 40, 60, 66, 91
 mp_rk4, 92
 mp_rk4 (state_updates), 101
 mp_rk4_old (state_updates), 101
 mp_set_numbers (mp_structured_vector), 65
 mp_set_numbers(), 40
 mp_setdiff (mp_subset), 66
 mp_show_models, 60
 mp_show_models(), 81
 mp_sim_bounds, 61, 69
 mp_sim_bounds(), 62
 mp_sim_offset, 62, 69
 mp_sim_offset(), 62
 mp_simulator, 32, 63, 87, 99
 mp_slices, 63
 mp_sqrt (transform_distr_param), 105
 mp_square, 29, 47, 64, 67, 90
 mp_state_dependence_frame, 64
 mp_state_flow_vars (mp_vars), 91
 mp_state_vars (mp_vars), 91
 mp_structured_vector, 65
 mp_structured_vector(), 40
 mp_subset, 29, 40, 60, 66, 91
 mp_symmetric, 29, 47, 64, 66, 90
 mp_time_scale, 67
 mp_tmb, 67
 mp_tmb_calibrator, 31, 37, 49, 50, 52–54, 57, 58, 61, 62, 68, 84–86, 88, 90, 99
 mp_tmb_coef, 34, 68, 70
 mp_tmb_delete (mp_tmb_insert), 72
 mp_tmb_entire_library (mp_tmb_library), 81
 mp_tmb_expr_list, 70
 mp_tmb_fixef_cov, 72
 mp_tmb_implicit_backtrans, 68
 mp_tmb_implicit_backtrans
 (mp_tmb_insert_backtrans), 75
 mp_tmb_implicit_trans, 69
 mp_tmb_implicit_trans
 (mp_tmb_insert_trans), 80
 mp_tmb_insert, 31, 68, 72, 79

- mp_tmb_insert_backtrans, 75
- mp_tmb_insert_backtrans(), 80
- mp_tmb_insert_glm_timevar, 76
- mp_tmb_insert_log_linear, 77
- mp_tmb_insert_reports, 79
- mp_tmb_insert_trans, 80
- mp_tmb_insert_trans(), 75
- mp_tmb_library, 68, 72, 73, 75, 80, 81, 99
- mp_tmb_model_spec, 30, 32, 34, 36, 45, 46, 57, 59, 68, 72, 73, 75, 76, 78–81, 82, 92, 93, 101, 102
- mp_tmb_objective, 84
- mp_tmb_profile, 85
- mp_tmb_update, 31, 68, 69
- mp_tmb_update(mp_tmb_insert), 72
- mp_tmbstan_coef, 34, 86
- mp_traj, 86
- mp_trajectory, 10, 37, 68, 69, 87, 99
- mp_trajectory_ensemble, 69
- mp_trajectory_ensemble(mp_trajectory), 87
- mp_trajectory_par(mp_trajectory), 87
- mp_trajectory_replicate
(mp_trajectory), 87
- mp_trajectory_sd, 37, 69
- mp_trajectory_sd(mp_trajectory), 87
- mp_trajectory_sim(mp_trajectory), 87
- mp_triangle, 29, 47, 64, 67, 90
- mp_uncertainty_estimated, 90
- mp_unif(distribution), 6
- mp_uniform(distribution), 6
- mp_union, 29, 40, 60, 66, 91
- mp_vars, 91
- mp_version, 93
- mp_version_update, 59, 93
- mp_zero_vector, 94

- names.Index(mp_index), 39
- names_and_labels, 94
- nlist, 97
- nlminb, 50, 53
- not_all_equal(comparison), 5
- NULLReader(Reader), 87
- numeric, 9, 73, 82, 94, 95

- optim, 50

- package_version, 93
- pgamma(engine_functions), 11

- pnorm(engine_functions), 11
- print(engine_functions), 11
- print.Index(mp_index), 39
- print.StringData(StringData), 102
- proportions(engine_functions), 11

- rbf, 97
- rbind, 16
- rbind(engine_functions), 11
- rbind_lag(engine_functions), 11
- rbind_time(engine_functions), 11
- rbinom(engine_functions), 11
- Reader, 98
- readRDS, 59
- recycle(engine_functions), 11
- renderGraph, 8
- rep(engine_functions), 11
- reulermultinom(engine_functions), 11
- rnbinom(engine_functions), 11
- rnorm(engine_functions), 11
- round(engine_functions), 11
- row_sums(engine_functions), 11
- rowSums, 17
- rpois(engine_functions), 11
- RReader(Reader), 98

- saveRDS, 59
- sd(engine_functions), 11
- seq, 14
- seq(engine_functions), 11
- show_models(mp_show_models), 60
- si_example, 98
- si_example_code(si_example), 98
- si_example_object(si_example), 98
- simple_sims, 11, 99
- sin(engine_functions), 11
- sparse_mat_mult, 77
- sparse_mat_mult(engine_functions), 11
- sparse_matrix_notation, 100
- spec, 64
- sqrt(engine_functions), 11
- state_updates, 101
- StringData, 102
- StringDataFromDotted(StringData), 102
- StringDataFromFrame(StringData), 102
- sum(engine_functions), 11
- system.file, 81

- t, 16, 17

`t(engine_functions)`, 11
`time_group(engine_functions)`, 11
`time_step(engine_functions)`, 11
`time_var(engine_functions)`, 11
`to_diag(engine_functions)`, 11
`to_labels(names_and_labels)`, 94
`to_name(names_and_labels)`, 94
`to_name_pairs(names_and_labels)`, 94
`to_names(names_and_labels)`, 94
`to_positions`, 104
`to_string`, 104
`to_values(names_and_labels)`, 94
`Transform`, 105
`transform_distr_param`, 7, 105
`TXTRReader(Reader)`, 98

`unpack(engine_functions)`, 11